TARTALOM

MI AZ ANGULAR?

Az Angular egy **nyílt forráskódú**, **TypeScript-alapú frontend keretrendszer**, amelyet a **Google** fejleszt és tart fenn. Elsődleges célja, hogy megkönnyítse összetett, jól strukturált, skálázható webalkalmazások fejlesztését. Az Angular számos beépített funkciót kínál, mint például komponensalapú architektúra, irányított útvonalkezelés (routing), formkezelés, HTTP-kommunikáció, dependency injection, és egy erőteljes CLI eszközkészlet.

Az **első stabil verzió (Angular 2) 2016-ban jelent meg**, az AngularJS (1.x) utódjaként, amely egy teljes újraírással és új szemléletmóddal érkezett. Azóta az Angular folyamatosan fejlődik; évente több verziófrissítés jelenik meg. Az Angular 17 és 18 verziók bevezették a standalone komponensalapú fejlesztést és új optimalizációs lehetőségeket (pl. deferred loading, SSR integráció egyszerűsítése), míg a 20-as verzió már alapértelmezetten standalone szemléletet használ.

Az Angular napjainkban is **az egyik legnépszerűbb frontend keretrendszer** a nagyvállalati és komplex alkalmazásfejlesztés világában. Erősségei közé tartozik a teljes ökoszisztéma, a típusbiztonság, valamint a hosszú távú Google támogatás, amely biztosítja a folyamatos fejlődést és a robusztus alkalmazások építésének lehetőségét.

ANGULAR ELŐNYEI

STRUKTURÁLT ÉS JÓL SZERVEZETT KERETRENDSZER

- **Modularitás:** Az Angular moduláris felépítése lehetővé teszi a kód jobb szervezését és újrahasznosíthatóságát. A modulok segítenek a kód különböző részeinek elkülönítésében.
- **MVC architektúra:** Az Angular követi a Model-View-Controller (MVC) mintát, amely segíti a kód strukturáltságát és karbantarthatóságát.

ERŐTELJES ESZKÖZKÉSZLET

- **Angular CLI:** Az Angular CLI (Command Line Interface) megkönnyíti a projekt létrehozását, fejlesztését, tesztelését és telepítését. Automatizálja a gyakori fejlesztési feladatokat.
- **Széleskörű integráció:** Az Angular könnyen integrálható más eszközökkel és könyvtárakkal, mint például RxJS a reaktív programozáshoz és Angular Material a felhasználói felületekhez.

TÍPUSBIZTONSÁG

- **TypeScript:** Az Angular TypeScript alapú, ami statikus típusellenőrzést és fejlettebb fejlesztői eszközöket biztosít. Ez segíti a hibák korai észlelését és a kód olvashatóságát.

KERESŐOPTIMALIZÁLÁS (SEO)

- **Angular Universal:** Az Angular Universal lehetővé teszi a szerver oldali renderelést (SSR), ami javítja a keresőoptimalizálást és a kezdeti betöltési időt.

AKTÍV KÖZÖSSÉG ÉS TÁMOGATÁS

- **Nagy közösség:** Az Angular rendelkezik egy nagy és aktív közösséggel, ami segíti a fejlesztők problémáinak gyors megoldását. Számos forrás áll rendelkezésre, mint dokumentációk, fórumok és blogok.
- **Hivatalos támogatás:** Az Angular a Google támogatásával fejlődik, ami stabilitást és hosszú távú támogatást biztosít.

ANGULAR HÁTRÁNYAI

MEREDEK TANULÁSI GÖRBE

- **Komplexitás:** Az Angular komplexitása és funkcionalitása miatt a tanulási görbéje meredek lehet, különösen azoknak a fejlesztőknek, akik nem ismerik a TypeScript-et vagy az MVC architektúrát.
- **Számos fogalom és eszköz:** Az Angular használatához számos fogalmat és eszközt kell elsajátítani, mint például a Dependency Injection, az RxJS, a szolgáltatások és a modulok.

TELJESÍTMÉNY

- **Nagyobb alkalmazások:** Nagyobb alkalmazások esetén az Angular teljesítménye lassabb lehet más könnyebb keretrendszerekhez képest. A bonyolultabb alkalmazások optimalizálása időigényes lehet.
- **Bundle méret:** Az Angular alkalmazások nagyobb csomagmérettel rendelkezhetnek, ami befolyásolhatja a betöltési időt.

KONFIGURÁCIÓS IGÉNY

- **Beállítások és konfigurációk:** Az Angular projektek beállítása és konfigurálása bonyolultabb lehet, különösen a fejlett funkciók, mint az SSR vagy a Lazy Loading használatakor.

VISSZAMENŐLEGES KOMPATIBILITÁS

- **Frissítések:** Bár az Angular rendszeresen frissül, néha a nagyobb verzióváltások (major releases) esetén a visszamenőleges kompatibilitás problémát okozhat, és a meglévő projektek frissítése időigényes lehet.

NAGYOBB FEJLESZTÉSI IDŐ

- **Kód mennyisége:** Az Angular alkalmazások fejlesztése több időt vehet igénybe, mivel több boilerplate (ismétlődő, sablon) kódot kell írni és karbantartani a modulok, szolgáltatások és egyéb komponensek miatt.

ÖSSZEFOGLALÁS

Az Angular egy erőteljes és robusztus keretrendszer, amely ideális nagy és komplex webalkalmazások fejlesztéséhez. Előnyei közé tartozik a modularitás, a típusbiztonság, az aktív

közösség és az eszközkészlet. Ugyanakkor a meredek tanulási görbe, a teljesítményproblémák és a nagyobb konfigurációs igények hátrányt jelenthetnek, különösen kisebb projektek esetén. A megfelelő választás attól függ, hogy a projekt igényei és a fejlesztői csapat preferenciái milyen irányba mutatnak.

SZÜKSÉGES ISMERETEK

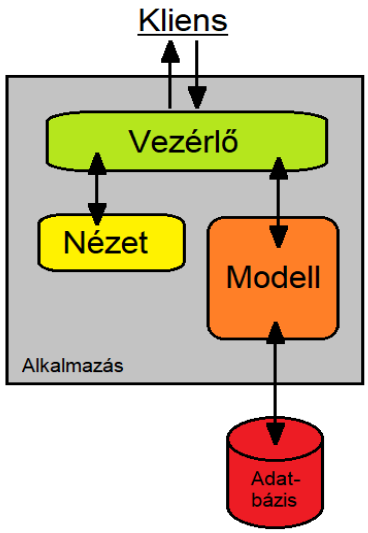
- HTML – HTML váz készítése, azok valid felépítése
- JavaScript – Vezérlési szerkezetek, változók használata, főleg az objektumok kezelése

AJÁNLOTT ISMERETEK

- CSS – Design szempontjából
- TypeScript – A hibák észlelése miatt
- OOP alapok – A kód újrahasznosíthatóságának megírása miatt

MVC MODELL

A **modell-nézet-vezérlő** (MNV) (angolul: ***model-view-controller***) a szoftvertervezésben használatos programtervezési minta. Összetett, sok adatot a felhasználó elé táró számítógépes alkalmazásokban gyakori fejlesztői kívánalom az **adathoz (modell)** és a **felhasználói felülethez (nézet)** tartozó dolgok szétválasztása, hogy a felhasználói felület ne befolyásolja az adatkezelést, és az adatok átszervezhetők legyenek a felhasználói felület változtatása nélkül. A modell-nézet-vezérlő ezt úgy éri el, hogy elkülöníti az **adatok elérését** és az **üzleti logikát** az **adatok megjelenítésétől** és a felhasználói interakciótól egy közbülső összetevő, a vezérlő bevezetésével.



ELŐNYEI

- ***Egyidejű fejlesztés*** – Több fejlesztő tud egyszerre külön a modellen, vezérlőn és a nézeteken dolgozni.
- ***Magas szintű összetartás*** – MNV segítségével az összetartozó funkciók egy vezérlőben csoportosíthatóak. Egy bizonyos modell nézetei is csoportosíthatóak.
- ***Függetlenség*** – MNV mintában az elemek alapvetően nagy részben függetlenek egymástól
- ***Könnyen változtatható*** – Mivel a felelősségek szét vannak választva a jövőbeli fejlesztések könnyebbek lesznek
- ***Több nézet egy modellhez*** – Modelleknek több nézetük is lehet
- ***Tesztelhetőség*** - mivel a felelősségek tisztán szét vannak választva, a külön elemek könnyebben tesztelhetőek egymástól függetlenül

HÁTRÁNYAI

- ***Kód olvashatósága*** – A keretrendszer új rétegeket ad a kódhoz, ami megnöveli a bonyolultságát
- ***Sok boilerplate kód*** – Mivel a programkód 3 részre bomlik, ebből az egyik végzi a legtöbb munkát, a másik kettő pedig az MNV minta kielégítése miatt létezik.
- ***Nehezebben tanulható*** – A fejlesztőnek több különböző technológiát is ismernie kell a MNV használatához.

DEPENDENCY INJECTION

A számítógép-programozásban a **dependency injection** egy technika, aminek lényege, hogy egy objektum más objektumok függőségeit elégíti ki. A függőséget felhasználó objektum szolgáltatást nyújt, az injekció pedig ennek a függőségnek az átadása a kliens részére. A szolgáltatás a kliens állapotának része. A minta alapkövetelménye a szolgáltatás kliensnek való átadása ahelyett, hogy a szolgáltató objektumot a kliens hozná létre.

A szolgáltató osztály szempontjából ez azt jelenti, hogy a kliens nem hívhat rajta konstruktort vagy statikus metódust. Paramétereit más osztályoktól kapja, azok állítják be. A függőséget előállítja valaki más, például a kontextus vagy a konténer problémája lesz.

A minta célja, hogy annyira leválassza a szolgáltató objektumot a kliensről, hogy ha kicserélik, akkor ne kelljen módosítani a klienst.

TYPESCRIPT

- Microsoft által fejlesztett úgynevezett JavaScript szuperszett, a JavaScript fejlesztett verziója
- Transpiler – Nem futtat, hanem fordít, minden benne megírt kódot JavaScripté fordítja le
- Mivel JS kódot generál ezért könnyen integrálható más JS projektekbe
- Összetett projektek esetén használják, mivel már fordításkor figyelmeztet az előforduló hibákra

TYPESCRIPT ELŐNYEI

STATIKUS TÍPUSELLENŐRZÉS

- **Hibák korai észlelése:** A típusok használatával a fordítási idő alatt sok hibát lehet észrevenni, amelyek futásidőben jelentkeznének, ha csak JavaScript-et használnánk.
- **Kódolási hibák megelőzése:** A típusok használata segít elkerülni a hibákat, mint például a típusok helytelen használata vagy a nem létező mezők elérése.

FEJLESZTŐI ÉLMÉNY

- **IntelliSense és autocompletion:** A fejlesztői környezetek, mint például a Visual Studio Code, jobban támogatják a TypeScript-et, részletesebb autocompletion és IntelliSense funkciókkal.
- **Könnyebb kódolvasás és karbantartás:** A típusok és interfészek használata világosabbá teszi a kódot, ami megkönnyíti a kód olvasását és karbantartását.

SKÁLÁZHATÓSÁG

- **Nagyobb projektek kezelése:** A TypeScript jobban alkalmas nagyobb projektek kezelésére, mivel a típusok segítenek a kód szervezésében és karbantartásában.
- **Refaktorálás támogatása:** A típusbiztonság segíti a refaktorálást, mivel a fordító figyelmeztet a típusok megsértésére.

MODERN JAVASCRIPT FUNKCIÓK

- **ES6+ funkciók használata:** A TypeScript támogatja a legújabb JavaScript funkciókat, még akkor is, ha a célböngészők nem támogatják azokat natívan, mivel a TypeScript lefordítja a kódot kompatibilis verzióra.

KÖZÖSSÉGI TÁMOGATÁS ÉS ESZKÖZKÉSZLET

- **Kiterjedt ökoszisztéma:** A TypeScript köré épült nagy közösség és eszközkészlet segíti a fejlesztést és a hibák gyors megoldását.

TYPESCRIPT HÁTRÁNYAI

TANULÁSI GÖRBE

- **További tanulás szükséges:** A JavaScript fejlesztőknek időt kell fordítaniuk a TypeScript nyelvi funkcióinak és típusrendszerének elsajátítására.

KOMPLEXITÁS NÖVEKEDÉSE

- Bonyolultabb kód: A típusdefiníciók és a szigorúbb szintaxis bonyolultabbá teheti a kódot, különösen kisebb projektek esetében, ahol a JavaScript egyszerűsége előnyt jelenthet.

FORDÍTÁS SZÜKSÉGESSÉGE

- **Fordítási lépés:** A TypeScript kódot először le kell fordítani JavaScript-re, mielőtt futtatható lenne, ami extra lépést és időt igényel a fejlesztési folyamatban.

KOMPILÁCIÓS IDŐ

- **Hosszabb build idő:** Nagyobb projektek esetén a fordítási idő hosszabb lehet, különösen akkor, ha sok fájlt és komplex típusokat használunk.

KIEGÉSZÍTŐ TÍPUSDEFINÍCIÓK

- **Külső könyvtárak:** Ha külső JavaScript könyvtárakat használunk, szükség lehet további típusdefiníciós fájlok (DefinitelyTyped) beszerzésére vagy megírására, ami extra munkát jelenthet.

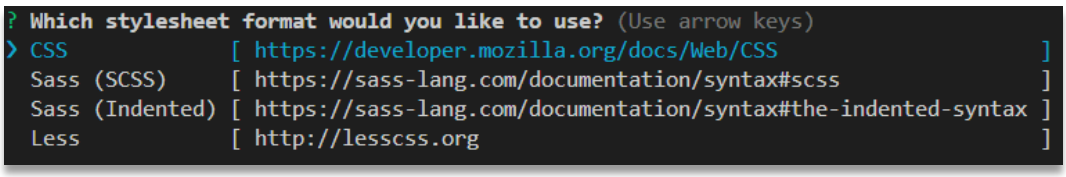
ÖSSZEFOGLALÁS

A TypeScript számos előnyt kínál, különösen **nagyobb, komplex projektek esetén**, ahol a **típusbiztonság, a kód olvashatósága** és a fejlesztői eszközök támogatása nagy értéket képvisel. Ugyanakkor kisebb projektek vagy egyszerűbb fejlesztési feladatok esetén a TypeScript komplexitása és a fordítási szükségesség hátrányt jelenthet. A megfelelő választás attól függ, hogy a projekt igényei és a fejlesztői csapat preferenciái milyen irányba mutatnak.

Az Angular 20 már a TypeScript 5.x verzióit használja.

ELSŐ HASZNÁLAT, PROJEKT LÉTREHOZÁSA

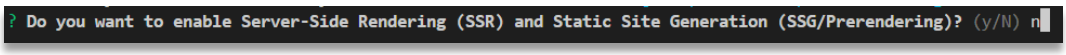
- Létrehozunk egy új mappát.
- Megnyitjuk Visual Studio Code-ban.
- Ezután új terminálban használjuk a Node Package Managert.
  Akinek nincs telepítve az a továbblépés előtt mindenképp telepítse a Node.js-t.
- Telepítjük az angulart az alábbi paranccsal.
  - `npm install -g @angular/cli`
- Ellenőrizhetjuk, az **angular verzióját** az alábbi paranccsal:
  - `ng v`
- Új saját alkalmazás/app létrehozása:
  - `ng new Alapok --no-standalone`
    Hagyományos Angular modulos megközelítést alkalmaz, ahol minden komponens, direktíva és egyéb elem modulokban kerül deklarálásra. (Visszafelé kompatibilitás miatt szokták alkalmazni, modul alapú komponensekhez.)
  - `ng new Alapok`
    Standalone komponens megközelítést alkalmaz, ami egyszerűbbé és modulmentessé teszi a komponensek használatát és deklarálását. Ilyenkor nincs „Appmodule” rész sem. **Az Angular 20-ban a standalone verzió az alapértelmezett.**
- Az alkalmazás főként standalone komponensekből fog állni, amelyek nem igényelnek külön modult a deklaráláshoz.
- A komponensek közvetlenül használhatók és importálhatók más komponensekbe vagy szolgáltatásokba.
- A projekt struktúrája egyszerűbbé válik, mivel nem szükséges modulokat létrehozni és fenntartani a komponensek és direktívák számára.
- Ezután az alábbi kérdést teszi fel:



Itt érdemes a CSS-t választani, mivel ez a legelterjedtebb stílusformátum.

Később bármikor átállítható más formátumra is (pl. SCSS).

- Később, további kérdést is feltesz, itt a 'no' opciót (n betű) válaszd a billentyűzetről.



SERVER-SIDE RENDERING (SSR)

MI AZ SSR?

Az SSR során a weboldal tartalmát a szerver oldalon renderelik, mielőtt azt elküldenék a kliens (böngésző) számára. Ez azt jelenti, hogy a szerver a HTML oldalt dinamikusan generálja a kérések feldolgozásakor.

Angular 17-től kezdődően az SSR támogatása egyszerűsödött: a `ng add @angular/ssr` paranccsal néhány másodperc alatt szerveroldali renderelésre képes alkalmazás hozható létre.

ELŐNYÖK

- **Gyorsabb kezdeti betöltési idő:** Mivel az oldal tartalma már a szerveren előre van renderelve, a böngésző gyorsabban jeleníthető meg.
- **SEO barát:** A keresőmotorok könnyebben tudják indexelni az előre renderelt HTML tartalmat, ami jobb keresőoptimalizálást (SEO) eredményezhet.
- **Dinamikus tartalom:** Az SSR jól működik azokban az esetekben, amikor az oldal tartalma gyakran változik vagy felhasználói interakciók alapján dinamikusan generálódik.

HÁTRÁNYOK

- **Szerver terhelés:** A szerver oldali renderelés, nagyobb terhelést ró a szerverre, mivel minden kérést feldolgozni kell.
- **Bonyolultabb infrastruktúra:** Az SSR megvalósítása és fenntartása bonyolultabb lehet, különösen nagyobb alkalmazások esetén.

STATIC SITE GENERATION (SSG)

MI AZ SSG?

Az SSG során a weboldal statikus HTML fájljait előre generálják a build folyamat során. Ezek a statikus fájlok kerülnek kiszolgálásra a felhasználók számára.

ELŐNYÖK

- **Gyors kiszolgálás:** A statikus fájlok kiszolgálása gyors, mivel nincs szükség dinamikus szerver oldali renderelésre.
- **Skálázhatóság:** Könnyen skálázható, mivel a statikus fájlokat CDN-en (**Content Delivery Network**) keresztül lehet terjeszteni.
- **Biztonság:** A statikus oldalak kevésbé sebezhetők, mivel nincs szükség adatbázisra vagy szerver oldali logikára.

HÁTRÁNYOK

- **Kevésbé dinamikus:** Az SSG nem alkalmas olyan oldalakra, ahol a tartalom gyakran változik vagy felhasználói interakciók alapján dinamikusan frissül.
- **Hosszabb build idő:** Nagyobb oldalak esetén a build folyamat hosszabb időt vehet igénybe, mivel minden oldalt előre generálni kell.

MIKOR MELYIKET ÉRDEMES HASZNÁLNI?

- **SSR:** Ha az alkalmazásod tartalma gyakran változik, dinamikusan generált adatokra van szükséged, vagy fontos a SEO, akkor az SSR a megfelelő választás.
- **SSG:** Ha az alkalmazásod tartalma ritkán változik, és fontos a gyors betöltési idő és a könnyű skálázhatóság, akkor az SSG a megfelelő megoldás.

Ezután elkezdődik a terminálban a telepítés, és a végeztével létrejött az projekt alapszerkezete!

MIK AZOK A MODULOK

Az Angular modulok alapvető építőkövei az Angular alkalmazásoknak. Egy modul olyan mechanizmus, amely segítségével csoportosíthatók az alkalmazás különböző részei, például komponensek, direktívák, szolgáltatások és egyéb funkciók, amelyek egy logikai egységet alkotnak. Minden Angular alkalmazásnak van legalább egy modulja, az úgynevezett **root** modul, amely általában az **AppModule**.

Az Angular modulok tehát segítenek abban, hogy az alkalmazás kódját szervezetté, újrahasznosíthatóvá és karbantarthatóvá tegyék.

MIK AZOK A KOMPONENSEK

A projektedben komponenseket hozol létre, melyek az MVC modell felépítésének megfelelően, külön kezelik a HTML, a CSS, illetve a TYPESCRIPT(JavaScript) részt...

Az Angular komponensek olyan egységek, amelyek egy alkalmazásban található nézeteket és logikát képviselik. A komponensek segítségével az Angular alkalmazásokat modulárisan és könnyen kezelhető módon fejleszthetjük.

AZ ANGULAR KOMPONENSEK ELŐNYEI

- A moduláris fejlesztés: Az Angular komponensek segítségével modulárisan fejleszthetjük az alkalmazást, ami javítja a fejlesztési és karbantartási folyamatot.
- A könnyű újra felhasználhatóság: Az Angular komponensek könnyen újra felhasználhatók az alkalmazás különböző részein, ami javítja a kód újrahasznosítását.

- A könnyű tesztelhetőség: Az Angular komponensek könnyen tesztelhetők, mert a függőségi befecskendezés segítségével a függőségeket könnyen cserélni lehet, ami javítja a tesztelhetőséget és a karbantarthatóságot.

Az Angular komponensek előnyöket kínálnak, a natív HTML, CSS és JavaScript kódú fájlokkal szemben is:

- Az Angular komponensek használatával könnyebben tudunk kezelni a komponensállapotot, mivel az Angular automatikusan figyeli a változásokat, és frissíti a nézetet.
- Az Angular komponensek segítségével könnyebben kezelhető a komponens logikája, különösen az eseménykezelést és az állapotváltozásokat.

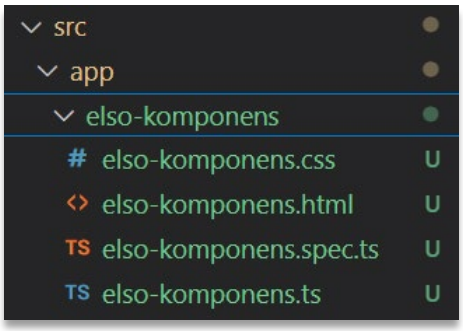
Létrehozni őket az alábbi paranccsal lehet:

- `ng generate component KomponensNeve`
- `ng g c KomponensNeve`

A példából jól látható a **`generate`** és a **`component`** kifejezések akár 1 betűsre is rövidíthetőek.

ÚJ KOMPONENS VÁZA

- A **`*.html`** kiterjesztésű a frontend vázát tartalmazza.
- A **`*.css`** kiterjesztésű a stíluslap formázási tulajdonságait.
- A **`*.ts`** állomány tartalmazza a komponens logikáját.
- A **`*.spec.ts`** pedig a unit teszteket foglalja magába.



HOGYAN HASZNÁLJUK

Mi az app mappában fogunk dolgozni/kódolni, de jönnek létre az általunk generált komponensek.

Az Angular alkalmazások egyéb forrásfájljait, pl. képek, az **`src/assets/`** mappába helyezzük. A projekt futtatásakor bele kell lépnünk az azt tartalmazó mappába. **`cd ProjektNeve`**. Majd ki lehet adni az alábbi parancsok: **`ng serve`**

Ezután a lefordított kódot a **`localhost:4200`** címe megtaláljuk a böngészőben!

Ha a projektet megjelenítését szeretnéd leállítani, használd a **`Ctrl + C`** billentyűkombinációt…

Miután kiadtuk a leállítás parancsot, létrehozhatunk új komponenst, telepíthetünk új modulokat vagy egyéb szükséges parancsokat adhatunk ki, amik a projektünk számára fontosak!

FELHASZNÁLT FORRÁSOK

https://angular.io/

https://angular.io/guide/dependency-injection

https://hu.wikipedia.org/wiki/A_f%C3%BCgg%C5%91s%C3%A9g_befecskendez%C3%A9se

https://hu.wikipedia.org/wiki/Modell-n%C3%A9zet-vez%C3%A9rl%C5%91

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

https://en.wikipedia.org/wiki/TypeScript