Forrás: https://hackr.io/blog/angular-interview-questions

Kérdés: Mi az Angular?

Válasz: Az Angular egy TypeScript-alapú nyílt forráskódú webalkalmazás-keretrendszer, amelyet a Google fejlesztett és tart fenn. Könnyű és hatékony módszert kínál a webes frontend alkalmazások fejlesztésére.

Az Angular számos olyan funkciót integrál, mint a declarative templates, dependency injection, end-to-end tooling stb., Amelyek megkönnyítik a webalkalmazások fejlesztését.

Kérdés: Miért vezették be az Angulart kliensoldali keretrendszerként?

Válasz: Hagyományosan a VanillaJS-t és a jQuery-t használták a fejlesztők dinamikus weboldalak fejlesztésére. Mivel a weboldalak összetettebbé váltak a hozzáadott funkciókkal és funkcionalitással, a fejlesztők számára nehéz volt fenntartani a kódot. Tehát az Angulart e problémák megoldására építették, így megkönnyítve a fejlesztők számára a kódot kisebb információrészekre osztva, amelyeket Components (komponensek) néven ismerünk.

Kérdés: n-content Directive?

Válasz: A hagyományos HTML elemek tartalmaznak némi tartalmat. Például:

```
<p> Tedd ide a bekezdést </p>
```

Nézzük most meg a következőt:

```
<app-work> Ez addig nem fog működni, amíg nem használjuk az ng-content direktívát </app-work>
```

Ez azonban nem fog működni úgy, ahogy a HTML elemeknél. Ahhoz, hogy a fent említett HTML példához hasonlóan működjön, az ng-content direktívát kell használnunk. Sőt, hasznos az újrafelhasználható komponensek felépítésében.

Kérdés: Angular különféle jellemzőit.

Válasz: Az Angular számos funkcióval rendelkezik, amelyek ideális frontend JavaScript keretrendszerré teszik. Ezek közül a legfontosabbakat a következőképpen írják le:

1.) Angular CLI

Az Angular támogatja a parancssori interfész eszközöket. Ezek az eszközök használhatók komponensek hozzáadására, tesztelésre, deploy-olásra stb.

2.) Cross-Platform App Development

Az Angular hatékony natív mobil és webalkalmazások felépítésére használható. Az Angular támogatást nyújt a natív mobilalkalmazások Cordova, Ionic vagy NativeScript használatával történő létrehozásához.

Az Angular lehetővé teszi a nagy teljesítményű, progresszív webalkalmazások létrehozását a modern webplatform képességek felhasználásával. A népszerű JS keretrendszer asztali alkalmazások készítéséhez is használható Linux, macOS és Windows rendszerekhez.

3.) Kódgenerálás

Az Angular képes a sablonokat nagymértékben optimalizált kódokká konvertálni a modern JavaScript virtuális gépek számára.

4.) Kódfelosztás

Az új Component Router segítségével az Angular alkalmazások gyorsan betöltődnek. A Component Router automatikus kódfelosztást kínál, így csak a felhasználó által kért nézet megjelenítéséhez szükséges kód kerül betöltésre.

5.) Template

Lehetővé teszi felhasználói felület nézetek (UI views) létrehozását egyszerű és hatékony sablonszintaxissal.

6.) Tesztelés

Az Angular lehetővé teszi a Karma használatával végzett unit test-et.

Kérdés: Mutassa be az Angular néhány előnyét

Válasz:: Számos beépített szolgáltatás, például az routing, state management, rxjs library, és a HTTP service-ek.

Deklaratív felhasználói felület: Az Angular HTML-t használ az alkalmazás felhasználói felületének megjelenítéséhez, mivel ez deklaratív nyelv, és sokkal könnyebben használható, mint a JavaScript.

Hosszú távú Google-támogatás: A Google azt tervezi, hogy ragaszkodik az Angularhoz, és tovább bővíti ökoszisztémáját, mivel hosszú távú támogatást kínált az Angular számára.

Kérdés: Mi a lifecycle hook Angularban?

Válasz: Angular componensek életük ciklusába lépnek a létrehozásától a megsemmisüléséig. Angular hooks lehetővé teszik ezeknek a fázisoknak a bekapcsolását és a változások kiváltását az életciklus egyes szakaszaiban.

ngOnChanges (): Ezt a módszert akkor hívják meg, amikor a komponens egy vagy több input properties-je megváltozik. A hook egy SimpleChanges objektumot kap, amely tartalmazza a property előző és aktuális értékeit.

ngOnInit (): Ezt a hook-ot egyszer hívják meg, az ngOnChanges hook után. Inicializálja a komponenst és beállítja a komponens bemeneti tulajdonságait.

ngDoCheck (): Meghívásra kerül az ngOnChanges és az ngOnInit után, és olyan változások észlelésére szolgál, amelyeket az Angular nem tud felismerni. Ebben a hook-ban tudjuk megvalósítani a változás detektálási algoritmusunkat.

ngAfterContentInit (): Az első ngDoCheck hook után hívják meg. Ez a hook válaszol, miután a tartalom kirajzolódik a komponens belsejébe.

ngAfterContentChecked (): Meghívásra kerül az ngAfterContentInit és minden további ngDoCheck után. A kirajzolódott tartalom ellenőrzése után válaszol.

ngAfterViewInit (): Egy komponens nézetét követően hívódik, vagy egy gyermek komponens nézetét inicializálják az után hívódik meg.

ngAfterViewChecked (): Meghívásra kerül az ngAfterViewInit után, és válaszol az komponens nézete vagy a gyermek komponens nézetének ellenőrzése után.

ngOnDestroy (): Pont azelőtt hívják meg, mielőtt az Angular megsemmisítené a komponenst. Ez a hook használható a kód megtisztítására és az eseménykezelők leválasztására.

Kérdés: Magyarázza el Dependency Injection-t?

Válasz: A Dependency Injection egy olyan alkalmazás-tervezési minta, amelyet az Angular valósít meg, és ez képezi az Angular alapfogalmait.

Az Angular dependenyc-jei service-ekkel rendelkeznek. Az alkalmazás különböző komponenseinek és direktíváinak szükségük lehet a szolgáltatás ezen funkcióira. Az Angular sima mechanizmust biztosít, amellyel ezeket a függőségeket a komponensekbe és direktívákba injektálják.

Kérdés: Írja le az MVVM architektúráját.

Válasz: Az MVVM architektúra eltávolítja az egyes komponensek szoros összekapcsolását. Az MVVM architektúra három részből áll:

Modell

View

ViewModel

Lehetővé teszi, hogy a gyerek komponenseknek legyenek referenciák observables-ön keresztül, (és nem közvetlenül) a szülő komponensükre.

Modell: Egy alkalmazás adatait és üzleti logikáját képviseli, vagy mondhatjuk, hogy egy entitás felépítését tartalmazza. Ez az üzleti logikából áll - helyi és távoli adatforrás, modellosztályok.

Nézet (View): A nézet az alkalmazás vizuális rétege, és így a felhasználói felület kódjából áll. A felhasználói műveletet elküldi a ViewModelnek, de a választ közvetlenül nem kapja meg. Fel kell iratkoznia az observables-ön keresztül, amelyeket a ViewModel képvisel, hogy megkapja a választ.

ViewModel: Az alkalmazás egy elvont rétege, és hídként működik a Nézet és a Modell (üzleti logika) között. Nincsenek nyomai, amelyeket a Nézetnek (View) használnia kell, mivel nincs közvetlen hivatkozása a Nézetre (View). A View és a ViewModel adat-összerendeléssel van összekapcsolva, így a ViewModel nézet bármilyen módosítása tudomásul veszi és megváltoztatja a modellen belüli adatokat. Kölcsönhatásba lép a Modellel.

Kérdés: Mutassa be a navigációt egy Angular alkalmazásban.

Válasz: A következő kód bemutatja, hogyan lehet navigálni a különböző oldalak között:

```
import from "@angular/router";
.
.
.
@Component({
  selector: 'app-header',
  template: `
<nav class="navbar navbar-light bg-faded">
  <a class="navbar-brand" (click)="goHome()">Some Search App</a>
  <ul class="nav navbar-nav">
    <li class="nav-item">
      <a class="nav-link" (click)="goHome()">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" (click)="goSearch()">Search</a>
    </li>
  </ul>
</nav>
  `
})
class HeaderComponent {
  constructor(private router: Router) {}
  goHome() {
    this.router.navigate(['']);
  }
  goSearch() {
    this.router.navigate(['search']);
  }
}
```

Kérdés: Mi az AOT (Ahead-Of-Time) fordítás ? Milyen előnyei vannak?

Válasz: Az Angular alkalmazás olyan komponensekből és template-ekből áll, amelyeket a böngésző nem ért. Ezért minden Angular alkalmazást le kell fordítani, mielőtt a böngészőben futtatna. Az Angular fordító beveszi a JS kódot, lefordítja, majd előállít néhány JS kódot. AOT-fordításnak hívják, és felhasználónként alkalmanként csak egyszer fordul elő.

Az Angular kétféle fordítást kínál:

JIT (Just-in-Time) fordítás: az alkalmazás futás közben fordít a böngészőben

AOT (Ahead-of-Time) fordítás: az alkalmazás build közben fordít.

Az AOT fordítás előnyei:

Gyors renderelés: A böngésző betölti a futtatható kódot, és azonnal megjeleníti az alkalmazás fordításakor, mielőtt a böngészőben futna.

Kevesebb Ajax kérés: A fordító elküldi a külső HTML és CSS fájlokat az alkalmazással együtt, megszüntetve az AJAX kéréseket ezekhez a forrásfájlokhoz.

A hibák minimalizálása: A build fázisban könnyen észlelhetők és kezelhetők a hibák.

Jobb biztonság: Mielőtt egy alkalmazás futna a böngészőben, az AOT fordító HTML-t és sablonokat ad hozzá a JS fájlokhoz, így nincsenek extra HTML fájlok, így jobb biztonságot nyújtva az alkalmazás számára.

Kérdés: Service-ek angularban

Válasz: A Singleton objektumokat Angularban, amelyek az alkalmazás élettartama alatt csak egyszer példányosodnak, Service-eknek nevezzük. Az Angular service-ek olyan metódusokat tartalmaznak, amelyek az adatokat az alkalmazás élettartama alatt fenntartják.

Az Angular service-ek elsődleges célja az üzleti logika, modellek, adatok és funkciók szervezése és megosztása az Angular alkalmazás különböző komponenseivel.

Angular service által kínált funkciókat bármely Angular komponensből hívhatjuk, például controller or directive.

Kérdés: Angular használatának előnyeit és hátrányait?

Válasz: Az alábbiakban bemutatjuk az Angular használatának különféle előnyeit:

Képesség egyedi direktíva (custom directive) elkészítéséhez

Nagy közösségi támogatás

Megkönnyíti a kliens és szerver kommunikációt

Olyan erős funkciókkal rendelkezik, mint az animáció és az eseménykezelők

Az MVC minta architektúráját követi

Támogatást nyújt a static template és Angular template-hez

Kétirányú adatkötés (two-way data-binding) támogatása

Támogatja a függőség-injektálást, a RESTful szolgáltatásokat és a validációkat

Az Angular használatának hátrányait a következőképpen soroljuk fel:

A komplex SPA-k méretük miatt kényelmetlenek és sok idő míg belölthetnek

A dinamikus alkalmazások nem mindig jó a teljesítményük

Az Angular tanulás erőfeszítést és időt igényel

Kérdés: Sorolja fel az Angular 7 néhány kiemelkedő jellemzőjét.

Válasz: Az Angular korábbi verzióival ellentétben a 7. fő kiadás az @ angular / core felosztásával jár. Ez annak érdekében történik, hogy csökkentsék a méretét. Általában nem minden modulra van szükség egy Angular fejlesztő számára. Ezért a 7-es angularban a @ angular / core minden egyes részének legfeljebb 418 modulja lesz.

Az Angular 7 behozta a drag-and-drop és a virtuális görgetést is. Ez utóbbi lehetővé teszi az elemek be- és kirakodását a DOM-ból. Virtuális görgetéshez az Angular legújabb verziója tartozik a csomaghoz. Ezenkívül az Angular 7 az ng-compiler új és továbbfejlesztett változatával érkezik.

Kérdés: Mi a string interpoláció?

Válasz: moustache syntax is nevezik, a sztringek interpolációja az Angular-ban egy speciális szintaxis-típusra utal, amely sablonkifejezéseket használ az komponensadatok megjelenítéséhez. Ezek a sablonkifejezések kettős zárójelbe vannak zárva, azaz {{}}.

Az Angular által végrehajtandó JavaScript kifejezéseket hozzáadják a zárójelekhez, és a megfelelő kimenetet beágyazzák a HTML kódba.

Kérdés: Scope hierarchia fogalma?

Válasz: Angular $scope objektumokat hierarchiába rendezi, amelyet általában a nézetek használnak. Ez scope hierachiának nevezhető. Gyökere van, amely tartalmazhat egy vagy több hatókört (scope), úgynevezett gyermek hatókört (scope).

A hatókör (scope) hierarchiában minden controller-nek megvan a saját $ scope-ja. Ezért a nézet nézetvezérlője által beállított változók rejtve maradnak a többi controller előtt. Az alábbiakban bemutatjuk a hatókör hierarchiájának tipikus ábrázolását:

Root $ scope

$ scope az 1. controller-hez

$ scope a 2. controller-hez

…

..

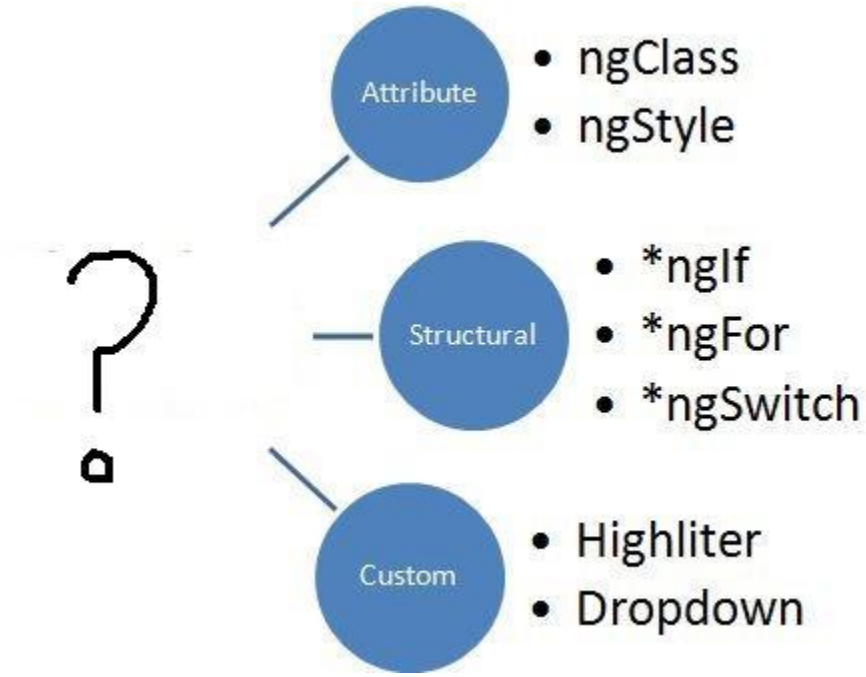
.

$ scope a n controller-hez

Kérdés: Miben különböznek az Observable a Promise-tól?

Válasz: Amint a promise létrehozódik, meghívódik is. Az observable-ök azonban nem így vannak, mert lusták (lazy). Ez azt jelenti, hogy semmi sem történik addig, amíg a subscription meg nem történik. Míg a promise-ok egyetlen eseményt kezelnek, az observable egy olyan adatfolyam, amely több esemény átadását is lehetővé teszi. Minden eseményről az observable-ben callback történik.

Kérdés: Nézzük meg az alábbi ábrát, mi kerülhet a ? helyére



Választ: direktívák.

Kérdés: Meg tudná magyarázni a sablonok (template) fogalmát Angularban?

Válasz: HTML-el írva a Angular sablonok Angular specifikus tulajdonságokat és elemeket tartalmaznak. A controller-től és a modell-től származó információkkal kombinálva a sablonokat (template) ezután tovább renderelik, hogy a felhasználónak megfeleljen a dinamikus nézet.

Kérdés: Magyarázza el a különbséget az Annotation és a Decorator között Angularban?

Válasz: Az Angular alkalmazásban az annotációkat egy annotációs tömb létrehozásához használják. Csak azok az osztály metaadatkészletei, amelyek a Reflect Metadata könyvtárat használják.

Az Angular dekorátorok olyan tervezési minták, amelyeket az osztályok dekorálására vagy módosításának elválasztására használnak az eredeti forráskód megváltoztatása nélkül.

Kérdés: Mik a direktívák?

Válasz: A direktívák az Angular egyik alapvető jellemzője. Lehetővé teszik egy Angular fejlesztő számára, hogy új, alkalmazás-specifikus HTML-szintaxist írjon. Valójában a direktívák olyan függvények, amelyeket az Angular fordító akkor hajt végre, amikor ugyanazok megtalálják őket a DOM-ban. A direktíváknak három típusa van:

Attribútum direktívák

Custom direktívák

Strukturális direktívák

Kérdés: Melyek az Angular építőkövei?

Válasz: Az Angular alkalmazás lényegében 9 építőelemet tartalmaz. Ezek:

Komponensek - Egy komponens egy vagy több nézetet vezérel. Minden nézet a képernyő bizonyos része. Minden Angular alkalmazásnak van legalább egy komponense, gyökér komponensként ismert. Egy komponens egy osztályon belül definiált alkalmazáslogikát tartalmaz.

Adatkötés - Az a mechanizmus, amellyel a template egyes részei koordinálják a komponens részeit adatkötésnek nevezik. Annak érdekében, hogy az Angular tudja, hogyan kell összekapcsolni mindkét oldalt (template-et és annak komponensét), a binding markup-ot hozzáadjuk a HTML sablonhoz.

Dependency Injection (DI) - Az Angular a DI-t használja az új komponensek szükséges függőségeinek biztosítására. Általában a komponens által igényelt függőségek szolgáltatások. Egy komponens konstruktorának paraméterei elmondják az Angularnak azokat a szolgáltatásokat, amelyekre egy komponensnek szüksége van.

Directives- Az Angular által használt template-ek dinamikus jellegűek. A direktívák feladata, hogy a template renderelésekor az Angular-nak megmondják, hogyan alakítsák át a DOM-ot. Valójában a komponensek template-el rendelkező direktívák. Az direktívák egyéb típusai az attribútum és a strukturális direktívák.

Metaadatok - Annak érdekében, hogy az Angular megtudja, hogyan kell feldolgozni egy osztályt, a metaadatokat csatolnak az osztályhoz. Ehhez decorator-t használnak.

Modulok - más néven NgModules, a modul egy szervezett kódblokk, amely meghatározott képességekkel rendelkezik. Meghatározott alkalmazási tartományt vagy munkafolyamatot tartalmaz. Minden Angular alkalmazásnak van legalább egy modulja. Ez gyökérmodul néven ismert. Általában egy Angular alkalmazásnak több modulja van.

Routing – Routing felelős azért, hogy a böngésző URL-jét view beöltéshez, navigáláshoz. A router egy oldal linkjeihez van kötve, hogy az Angular utasítsa az alkalmazási nézetet, ha a felhasználó rákattint.

Szolgáltatások (services) - Nagyon tág kategória, a szolgáltatás bármi lehet, az értéktől és a funkciótól kezdve egy olyan funkcióig, amelyre egy Angular alkalmazásnak szüksége van. Technikailag a Service egy jól meghatározott célú osztály.

Sablon (Template)- Minden komponens nézete társított sablonnal társul. Az Angularban található sablon a HTML-címkék egy olyan formája, amely tájékoztatja az Angular-t arról, hogy hogyan kell az komponenst megjeleníteni.

Kérdés: Angular és a jQuery közötti különbségeket?

Válasz: Az egyetlen legnagyobb különbség az Angular és a jQuery között az, hogy míg az előbbi JS frontend keretrendszer, az utóbbi egy JS library. Ennek ellenére van némi hasonlóság a kettő között, például mindkét DOM-manipulációval rendelkezik, és támogatja az animációt.

Ennek ellenére figyelemre méltó különbségek vannak az Angular és a jQuery között:

Az Angular kétirányú adatkötéssel (two-way data bindig) rendelkezik, a jQuery nem

Az Angular támogatja a RESTful API-t, míg a jQuery nem

A jQuery nem kínál routing-ot, az Angular támogatja

A jQuery-ben nincs formaellenőrzés, míg az Angular-ban van

Kérdés: Meg tudná magyarázni a különbséget az Angular kifejezések és a JavaScript kifejezések között?

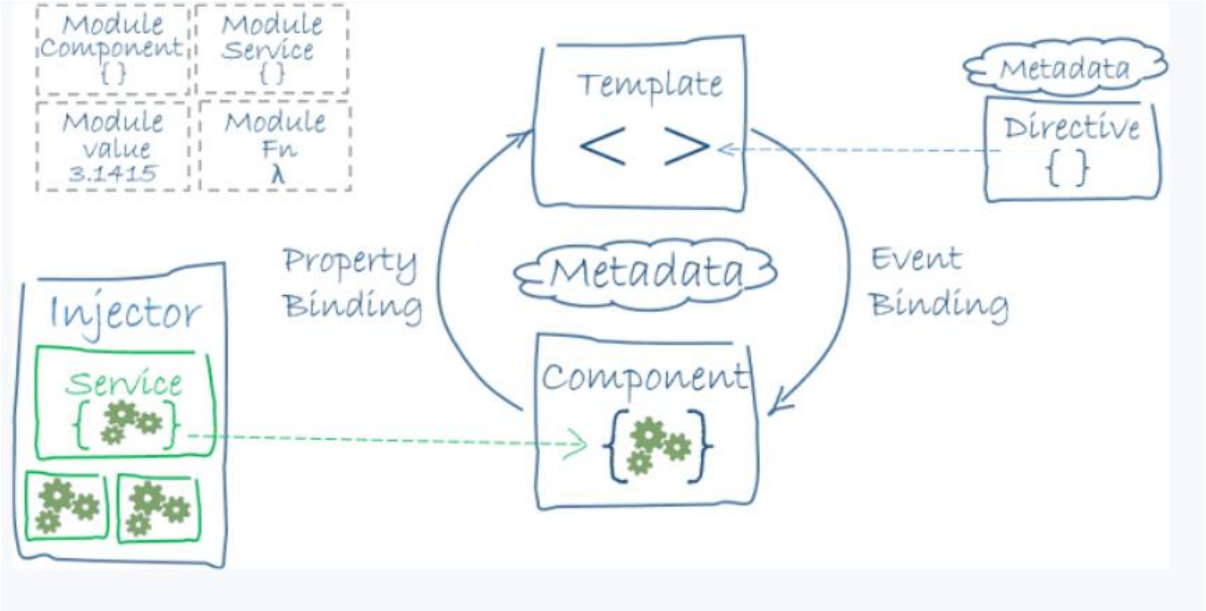
Válasz: Bár mind az Angular, mind a JavaScript kifejezések tartalmazhatnak literálokat, operátorokat és változókat, a kettő között vannak figyelemre méltó különbségek. Az alábbiakban felsoroljuk az Angular és a JavaScript kifejezések közötti fontos különbségeket:

Az Angular kifejezések támogatják a filter-eket, míg a JavaScript kifejezések nem

Lehetséges Angular kifejezéseket írni a HTML tag-ekbe. A JavaScript-kifejezéseket ezzel ellentétben nem lehet a HTML-tag-ekbe írni

Míg a JavaScript kifejezések támogatják a feltételeket, kivételeket és ciklusokat, addig az Angular kifejezések nem

Kérdés: Angular architektúra?



Kérdés: Mi az az Angular Material?

Válasz: Ez egy UI komponens könyvtár. Az Angular Material segít szép weboldalak, valamint webalkalmazások létrehozásában..

Kérdés: Mi az AOT (Ahead-Of-Time) fordítás?

Válasz: Minden Angular alkalmazást belsőleg fordítanak le. Az Angular fordító befogadja a JS kódot, lefordítja, majd előállít néhány JS kódot. Ez felhasználónként alkalmanként csak egyszer.

Kérdés: Mi az adatkötés? Hányféleképpen lehet megtenni?

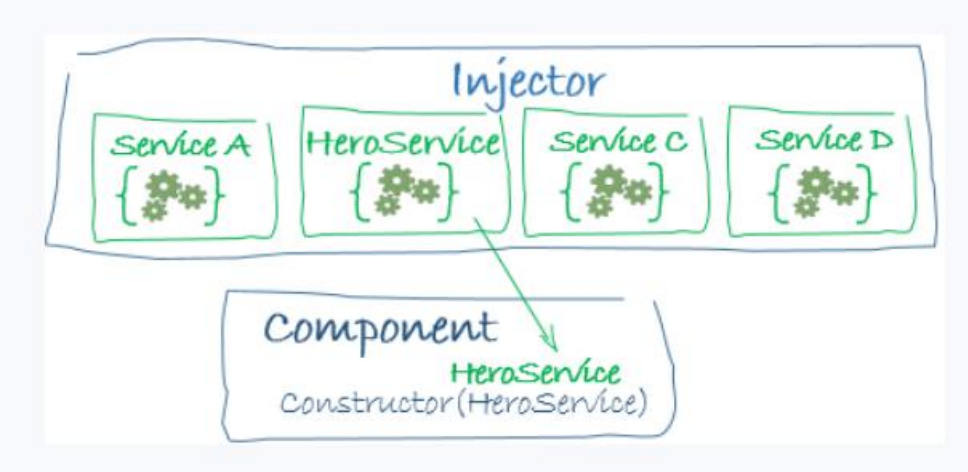
Válasz: Az alkalmazásadatok és a DOM (adatobjektum-modell) összekapcsolásához adatkötést használnak. A sablon (HTML) és a komponens (TypeScript) között történik. Az adatkötés elérésének 3 módja van:

Eseménykötés (Event Binding) - Lehetővé teszi az alkalmazás számára, hogy válaszoljon a felhasználói környezetben a felhasználói adatokra

Tulajdonságkötés (Property Binding)- Lehetővé teszi az alkalmazások adataiból kiszámított értékek interpolációját a HTML-be

Kétirányú kötés (Two-way Binding)- Az alkalmazás állapotában végrehajtott változtatások automatikusan tükröződnek a nézetben és fordítva. Az ngModel direktíva használható az ilyen típusú adatkötések elérésére.

Kérdés: Mit mutat a nyíl a következő képen?



Ez egy függőségi injekciót azaz DI-t jelent.

Kérdés: Összehasonlítás a service() és a factory() függvények között?

Válasz: Az alkalmazás üzleti rétegéhez használt service() függvény konstruktorként működik. A függvény futás közben meghívásra kerül az new kulcsszó használatával.

Bár a factory() függvény nagyjából ugyanúgy működik, mint a service() függvény, az előbbi rugalmasabb és erőteljesebb. Valójában a factory() függvények olyan tervezési minták, amelyek segítenek az objektumok létrehozásában.

Kérdés: Mi az ngOnInit ()? Hogyan definiálható?

Válasz: Az ngOnInit () egy lifecycly-hook, amelyet akkor hívnak meg, amikor az Angular befejezte az irányelv összes data-bound properties of a directive (adat kötési tulajdonságát a direktívának) inicializálását. Meghatározása:

```
Interface OnInit {

 ngOnInit() : void

 }
```

Kérdés: Mi az SPA (egyoldalas alkalmazás) az Angularban? SPA technológia a hagyományos web technológiával összehasonítva.

Válasz: A SPA technológiában csak egyetlen oldal, az index.HTML van fenntartva, bár az URL folyamatosan változik. A hagyományos web technológiával ellentétben a SPA technológia gyorsabb és könnyebben fejleszthető.

A hagyományos webes technológiában, amint a kliens weboldalt kér, a szerver elküldi az erőforrást. Amikor azonban a kliens ismét egy másik oldalt kér, a kiszolgáló ismét válaszol a kért erőforrás elküldésével. Ezzel a technológiával az a probléma, hogy sok időt igényel.

Kérdés: Mi a kód a decorator létrehozásához?

Válasz: Dummy nevű decoratort hozunk létre:

```
function Dummy(target) {

 dummy.log('This decorator is Dummy', target);

 }
```

Kérdés: Hogyan hívják azt a folyamatot, amellyel a TypeScript kódot JavaScript kódokká konvertálják?

Válasz: Transpilingnek hívják. Annak ellenére, hogy a TypeScript-t kódok írására használják az Angular alkalmazásokban, belsőleg átkerül egy egyenértékű JavaScript-be.

Kérdés: Mi a ViewEncapsulation, és hányféleképpen lehet ezt megtenni Angularban?

Válasz: Leegyszerűsítve: a ViewEncapsulation meghatározza, hogy az adott komponensben meghatározott stílusok befolyásolják-e az egész alkalmazást. Az Angular 3 típusú ViewEncapsulation-ot támogat:

Emulált - más HTML-ben használt stílusok elterjedtek a komponensre

Natív - A más HTML-ben használt stílusok nem terjednek át a komponensre

Nincs – A komponensben meghatározott stílusok az alkalmazás összes komponens számára láthatók

Kérdés: Miért használják inkább a TypeScript-et a JavaScript helyett Angular-ban?

Válasz: A TypeScript a Javascript egy halmaza, mivel ez a Javascript + típusok. Olyan extra funkciók, mint a változók típusai, változók hatóköre és még sok más. A TypeScript-et úgy tervezték, hogy kiküszöbölje a Javascript hiányosságait, például a változók, osztályok, dekorátorok, változó hatókör stb. Ezenkívül a Typescript pusztán objektum-orientált programozás, amely egy "fordítót" kínál, amely képes Javascript-ekvivalens kóddá konvertálni.