https://github.com/FAQGURU/FAQGURU/blob/master/topics/en/angular.md

Mi az egyenértékű az "ngShow" és az "ngHide" -al?

```
[hidden]="!myVar"
```

Mi a különbség a * ngIf és a [hidden] között?

* ngIf hatékonyan eltávolítja a tartalmat a DOM-ból, míg a [hidden] módosítja a megjelenítési tulajdonságot, és csak arra utasítja a böngészőt, hogy ne jelenítse meg a tartalmat, de a DOM még mindig tartalmazza.

Mi a különbség a "@Component" és a "@Directive" között Angular-ban?

Az direktívák viselkedést adnak egy meglévő DOM elemhez vagy egy meglévő komponens példányhoz.

A komponens a viselkedés hozzáadása / módosítása helyett valójában saját nézetet (a DOM elemek hierarchiáját) hoz létre csatolt viselkedéssel.

Írjon egy komponenst, ha egyéni viselkedéssel szeretne újrafelhasználható felhasználói felület DOM-elemeket létrehozni. Írjon direktívát, amikor újrahasználható viselkedést szeretne írni a meglévő DOM-elemek kiegészítésére.

Hogyan védené a router-en keresztül az activated komponenseket?

A Angular router a guard-al szállít. Ezek lehetőséget nyújtanak alkalmazásunk folyamatának ellenőrzésére. Megállíthatjuk a felhasználót bizonyos útvonalak felkeresésében, megakadályozhatjuk a felhasználók útvonalak elhagyását és még sok mást. Az Angular routes védelmének általános folyamata:

- Hozzon létre guard service-t: ng g guard auth
- Hozzon létre canActivate () vagy canActivateChild () metódusokat
- Az útvonalak meghatározásakor használja a guard-ot

```
// import the newly created AuthGuard
const routes: Routes = [
  {
    path: 'account',
    canActivate: [AuthGuard]
  }
];
```

Néhány egyéb rendelkezésre álló guard:

- CanActivate: Ellenőrizze, hogy van-e hozzáférése a felhasználónak

- CanActivateChild: Ellenőrizze, hogy a felhasználó hozzáfér-e valamelyik gyermek útvonalhoz
- CanDeactivate: A felhasználó elhagyhat egy oldalt? Például nem fejezték be a bejegyzések szerkesztését
- Resolve: Az adatok megragadása az útvonal elkészítése előtt
- CanLoad: Ellenőrizze, hogy be tudjuk-e tölteni az útvonalakat

Mit csinál a @HostBinding?

A @HostBinding segítségével property-ket állíthat be element or component that hosts the directive.

Hogyan futtatná az egység tesztet?

```
ng test
```

Mikor használná az eager modulbetöltést?

Eager betöltés: Egy funkció modul eager betöltéséhez be kell importálnunk az alkalmazás modulba a @NgModule decorator -t felhasználásával. Az eager betöltés hasznos kis méretű alkalmazásokban. Eager betöltéskor az összes funkciómodul betöltődik az alkalmazás elindítása előtt. Ezért az alkalmazás későbbi kérése gyorsabb lesz.

How to bundle an Angular app for production?

OneTime Setup

- `npm install -g @angular/cli`
- `ng new projectFolder` creates a new application

Bundling Step

- `ng build --prod` (run in command line when directory is `projectFolder`)
- flag `prod` _bundle for production

bundles are generated by default to ***projectFolder/dist/***

Deployment

```
ng serve –prod
```

Ekkor a http://localhost:4200 –on lesz elérhető.

Hogyan injektálhatjuk a base href-et?

Build-kor lehet módosítani a base href-et(<base href = "/">) az --base-href your-url opcióval.

```
# Sets base tag href to /myUrl/ in your index.html
ng build --base-href /myUrl/
```

Meg tudja magyarázni a Promise és Observable különbségeket az Angular-ban? Milyen forgatókönyv esetén használhatjuk az egyes eseteket?

Promise

A Promise egyetlen eseményt kezel, ha az aszinkron művelet befejeződik vagy sikertelen. Vannak Promise könyvtárak, amelyek támogatják a cancel-t, de az ES6 Promise nem támogatja.

Observable

Az Observable olyan, mint egy ** Stream ** (sok nyelven), és nulla vagy több olyan esemény átadását teszi lehetővé, ahol minden eseményhez callback-et hívnak.

Gyakran az Observable-t részesítik előnyben a Promise-al szemben, mert ez biztosítja a Promise és még sok más tulajdonságait. Az Observable segítségével nem számít, hogy 0, 1 vagy több eseményt akar-e kezelni. Minden esetben ugyanazt az API-t használhatja.

Az Observable törölhető, a Promise nem. Ha a szerverhez intézett HTTP-kérelem vagy más drága aszinkron művelet eredményére már nincs szükség, akkor az Observable subscription lehetővé teszi a subscription törlését, míg a Promise akkor is hívja a success vagy az error callback-et, ha nem már nincs szüksége az értesítésre vagy az általa nyújtott eredményre.

Az Observable olyan operátorokat nyújt, mint a map, a forEach, a reduce, ... hasonló egy tömbhöz. Vannak olyan hatékony operátorok is, mint például a retry () vagy a replay (), ... amelyek gyakran nagyon hasznosak.

Miért kell használni az ngOnInit-t, ha már van konstruktorunk?

A Konstruktor az osztály alapértelmezett módszere, amelyet az osztály példányosításakor hajtanak végre, és biztosítja az osztály és annak alosztályainak mezőinek megfelelő inicializálását.

Az ngOnInit egy life cycle hook, amelyet az Angular2 hív meg, jelezve, hogy az Angular elkészült a komponens létrehozásával.

Leginkább az ngOnInit-t használjuk az összes inicializáláshoz / deklarációhoz, és elkerüljük, hogy a konstruktorban működjenek dolgok. A konstruktort csak az osztálytagok inicializálására szabad használni, de nem végezhet tényleges "munkát". Tehát a constructor () segítségével kell beállítania a Dependency Injection beállítást, és nem sok mást. Az ngOnInit () jobb hely az "indításra".

Mi a különbség a "declarations ", a "providers " és az "import " között az NgModule-ban?

Az import az egyéb modulok exportált deklarációit teszi elérhetővé az aktuális modulban

A deklarációknak az aktuális modulról szóló direktívákat (beleértve a komponenseket és pipe-okat is) elérhetővé kell tenniük a jelenlegi modul más direktívái számára. Az direktívák, komponensek vagy pipe-ok selector-jai csak akkor illeszkednek a HTML-hez, ha azokat deklarálják vagy importálják.

A provider-ek service-eket készítenek és az értékeket a DI tudomására kell hozniuk. Hozzáadják őket a root scope-hoz, és más service-ekhez vagy direktívákhoz juttatják őket, amelyek függőségként tekintenek rájuk.

A provider-ek számára különleges eset a lazy loaded modulok, amelyek saját gyermekinjektort kapnak. a lazy loaded modul provider-jei alapértelmezés szerint csak ennek a lazy loaded modulnak biztosítják (nem a teljes alkalmazást, mint más modulokkal).

Mi az a reaktív programozás és hogyan használható az Angularban?

A reaktív programozás aszinkron adatfolyamokkal történő programozás. Az Angular alkalmazás egy reaktív rendszer.

Az eseményfolyamok (event streams) megvalósításának számos módja van. Angular alkalmazza az RxJS-t, és az EventEmitter osztály az RxJS / Observable megvalósítása.

Az RxJS-ben aszinkron adatfolyamokat (asynchronous data streams) ábrázol observable szekvenciák vagy egyszerűen csak observables segítségével.

Hogyan állítsunk be header-t minden egyes Angular kéréshez?

You could provide a service that wraps the original `Http` object from Angular.

```
import { Injectable } from '@angular/core';
import { Http, Headers } from '@angular/http';

@Injectable() export class HttpClient {
    constructor(private http: Http) {}

    createAuthorizationHeader(headers: Headers) {
        headers.append('Authorization', 'Basic ' + btoa('username:password'));
    }

    get(url) {
        let headers = new Headers();
        this.createAuthorizationHeader(headers);
        return this.http.get(url, {
            headers: headers
        });
    }

    post(url, data) {
        let headers = new Headers();
        this.createAuthorizationHeader(headers);
        return this.http.post(url, data, {
            headers: headers
        });
    }
}
```

And instead of injecting the `Http` object you could inject this one (`HttpClient`).

```
import { HttpClient } from './http-client';
```

Mikor tölt be egy lazy loaded modult?

A lazy loaded felgyorsítja az alkalmazás betöltési idejét azáltal, hogy több kötegre osztja, és igény szerint betölti őket, miközben a felhasználó az alkalmazásban navigál. Ennek eredményeként a kezdeti csomag sokkal kisebb, ami javítja a bootstrap időt.

A lazy loading az Angular nyelvű technika, amely lehetővé teszi a JavaScript-komponensek aszinkron betöltését egy adott útvonal aktiválásakor. Ez hozzáadhat némi kezdeti teljesítményt a kezdeti betöltés során, különösen, ha sok összetett útvonallal rendelkező komponens van.

Hogyan lehet észlelni az útvonal változását ?

Az Angular részben feliratkozhat (Rx esemény) egy Router példányra. Tehát például:

```
class MyClass {
    constructor(private router: Router) {
        router.subscribe((val) => /*whatever*/ )
    }
}
```

Vannak-e előnyei / hátrányai (különösen a teljesítmény szempontjából) abban, hogy a local storage-t a cookie funkcióinak helyettesítésére használják?

A cookie és a local storage különböző célokat szolgál.

- A cookie-t elsősorban a szerver oldali olvasásra szolgálnak,
- A local storage csak a kliens oldala olvashatja el.

Tehát az a kérdés, hogy az alkalmazásában kinek kellenek ezek az adatok – a kliensnek vagy a szervernek? Ha csak kliensnek(a JavaScript), akkor mindenképpen local storage. A sávszélességet pazarolja azzal, hogy az összes adatot elküldi az egyes HTTP fejlécekben.

Ha a szervernek, akkor a local storage nem jó, mert valahogy tovább kellene adnod az adatokat (az Ajax vagy a rejtett űrlapmezők, vagy valami mással együtt). Ez rendben lehet, ha a kiszolgálónak minden kéréshez csak a teljes adat egy kis részhalmazára van szüksége.

Nevezzen meg néhány biztonsági bevált gyakorlatot az Angular alkalmazásban

Az XSS hibák szisztematikus blokkolásához az Angular az összes értéket alapértelmezés szerint nem megbízhatóként kezeli (sanitation)

Az Angular sablonok megegyeznek a futtatható kóddal: a sablonokban a HTML, az attribútumok és a binding expressions biztonságosak. A biztonsági rések megelőzése érdekében használja az offline sablonfordítót, más néven template injection-t.

Kerülje a közvetlen interakciót a DOM-mal, és ehelyett használja az Angular sablonokat.

A template code injektálása egy Angular alkalmazásba megegyezik a futtatható kód beadásával az alkalmazásba. Tehát ellenőrizze az összes adatot a szerveroldali kódon, hogy megakadályozza a kiszolgáló XSS-biztonsági réseit.

Az Angular HttpClient beépített támogatást nyújt az XSRF támadások megakadályozásához a kliens oldalon.

A szerverek megakadályozhatják az XSSI-támadást azáltal, hogy az összes JSON-választ elõtaggal feltüntetik, hogy egyezmény szerint futtathatatlanná tegyék a jól ismert ""]} ', \ n "karakterláncot. Az Angular HttpClient könyvtára felismeri ezt a konvenciót, és a további elemzés előtt automatikusan lecsupaszítja a ")]} ', \ n" karakterláncot az összes válaszról.

Miért használ az Angular URL szegmenst?

Az UrlSegment az URL része a két perjel között. Tartalmaz egy elérési utat és a szegmenshez társított mátrixparamétereket.

A mátrixparaméterek egy path szakaszhoz, míg a lekérdezési paraméterek az URL-hez vannak kötve. Különböző szemantikájúak.

```
localhost:3000/heroes;id=15;foo=foo/bar/baz
// instead of localhost:3000/heroes/bar/baz?id=15&foo=foo
```

```
this.route.url.subscribe((url: UrlSegment[]) => {
  let heroes = url[0];
  let heroesMatrix = heroes.parameters();
  // heroes should contain id=15, foo=foo
  let bar = url[1].path; // 15
  let baz = url[2].path; //foo
})
```

A mátrixparamétereknél feliratkozhat a paraméterekre is ahelyett, hogy kihúzná őket az URL-ből.

```
this.paramSubscription = this.activeRoute.params.subscribe(params => {
  const bar = params['bar'];
  const baz = params['baz'];
});
```

Mikor kell használni a query paramétereket a mátrix paraméterekkel szemben?

A Mátrix paraméterek és a query paraméterek közötti különbségek sokkal többek, mint puszta konvenció.

A fő különbségek a következők:

- A query paraméterekkel rendelkező URL-ekre a válaszok nem kerülnek tárolásra közvetítők / proxyk között (jelenleg)
- a mátrix paraméterek bárhol megjelenhetnek az útvonalon
- a relatív uri kiszámítása különbözik
- a mátrixparaméterek nem erőforrások, hanem olyan szempontok, amelyek segítenek hivatkozni egy erőforrásra egy információs térben, amelyet nehéz a hierarchiában megjeleníteni