2FJ.io

Kurz: Angular Development

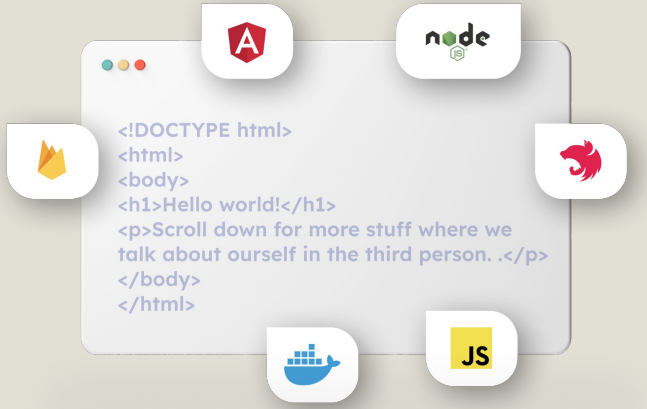
Dvoudenní intenzivní kurz Angularu

Autor: Furát Ismail

2FJ.io

O mně

2FJ.io



O mně

- Lead Frontend Developer
- Aktuálně pracuji s Angular, NestJS, Firestore, Docker a Kubernetes
- Předchozí zkušenosti s React, NextJS, PHP, SQL, MongoDB, VueJS a NodeJS

2FJ.io

Osnova dnešního kurzu

- Rekapitulace znalostí
- Stavební bloky Angularu
- Servisy: Promises vs Observables
- Angular 18 vs starší verze
- Standalone komponenty
- Pokročilý routing

2FJ.io

Průběh kurzu a přestávky

- Kurz je od 09:00 - 16:00
- Přestávka 10:30 - 10:50
- Přestávka na oběd 12:00 - 13:00
- Přestávka 14:00 - 14:20

(Popřípadě dle potřeby - dle domluvy)

2FJ.io

Materiály z kurzu

- pptx. prezentaci na konci kurzu rozešleme
- Kód ze cvičení na Githubu

2FJ.io

Začínáme!

2FJ.io

Je Angular framework nebo knihovna?

2FJ.io

Rozdíl mezi knihovnou a frameworkem

- Technický rozdíl “inversion of control” Pokud používáte knihovnu, udáváte způsob psaní aplikace vy s použitím funkcí knihovny
- Pokud používáte framework, způsob psaní aplikace je pevně daný frameworkem

2FJ.io

Angular a současnost

- AngularJS autor Miško Hevery, následně vývoj podpořen společností Google
- AngularJS první verze vydána 20. října 2010. Psal se v čistém Javascriptu
- AngularJS poslední verze 1.8.3 poté následně ukončen support v roce 2022
- Angular ve verzi 2.0.0 označen jako nástupce AngularJS jedná se o úplně přepracovaný framework psaný v Typescriptu
- Aktuální verze Angularu - 18

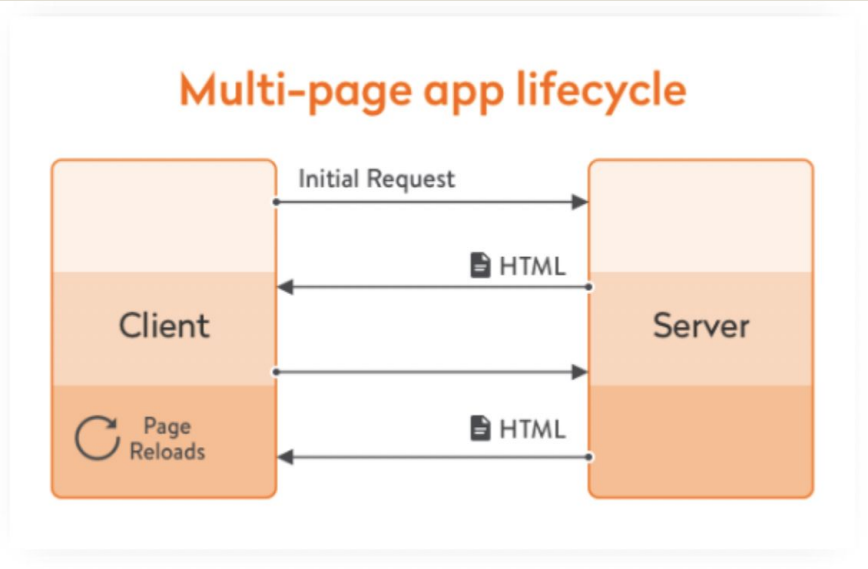
2FJ.io

Nová dokumentace

- https://angular.dev/
- (angular.io je přesměrován...)

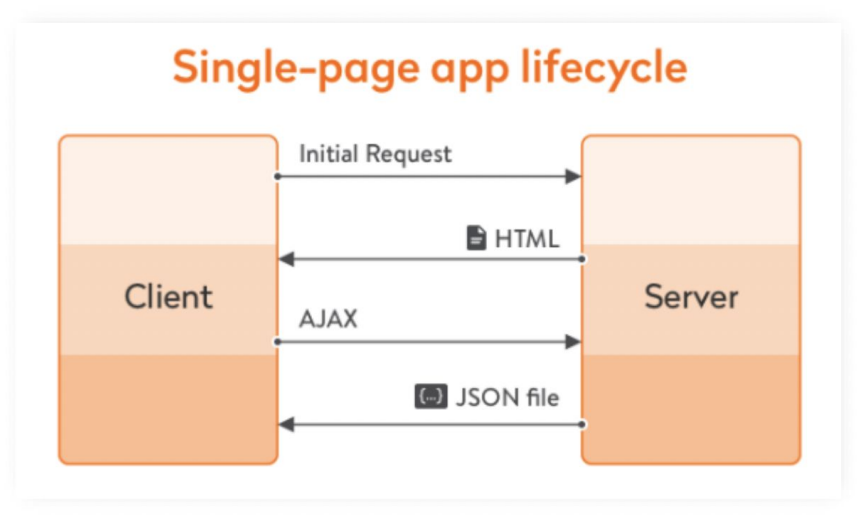
2FJ.io

Tradiční aplikace



2FJ.io

SPA - Single page aplikace



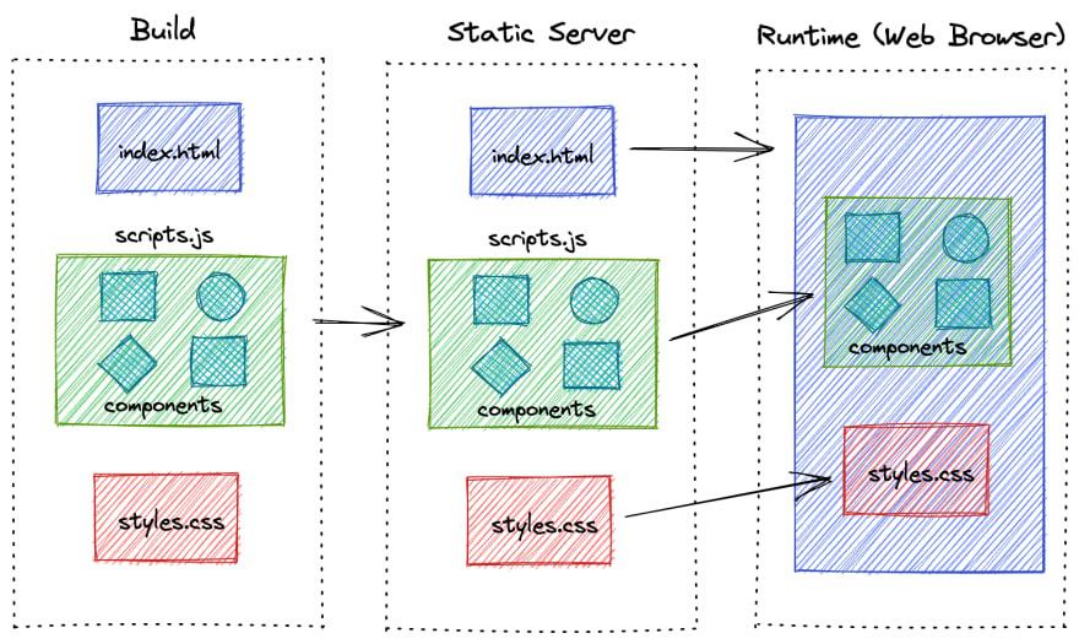
2FJ.io

Rendering v Angularu

- **CSR**
  - Výhody: Interaktivní a flexibilní uživatelské rozhraní, možnost práce offline nebo s cachingem dat.
  - Nevýhody: Pomalejší prvotní načtení a horší SEO, protože obsah stránky není okamžitě dostupný pro vyhledávače.
- **SSR**
  - Výhody: Rychlejší načtení a lepší SEO, protože prohlížeč obdrží kompletní HTML ihned.
  - Nevýhody: Vyšší náročnost na nastavení serveru a složitější synchronizace mezi serverem a klientem.
- **SSG**
  - Výhody: Nejrychlejší pro statický obsah
  - Nevýhody: nelze použít pro dynamický obsah, jednotlivé routes jsou na serveru jako HTML soubory.

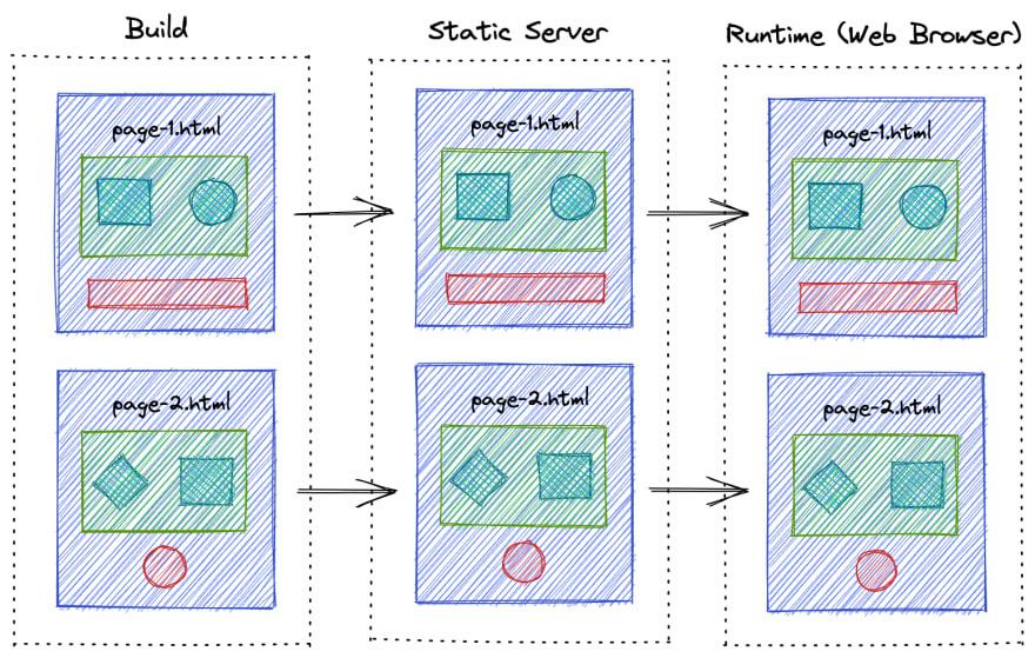
2FJ.io

CSR


Build
index.html
scripts.js
components
styles.css
Static Server
index.html
scripts.js
components
styles.css
Runtime (Web Browser)
components
styles.css

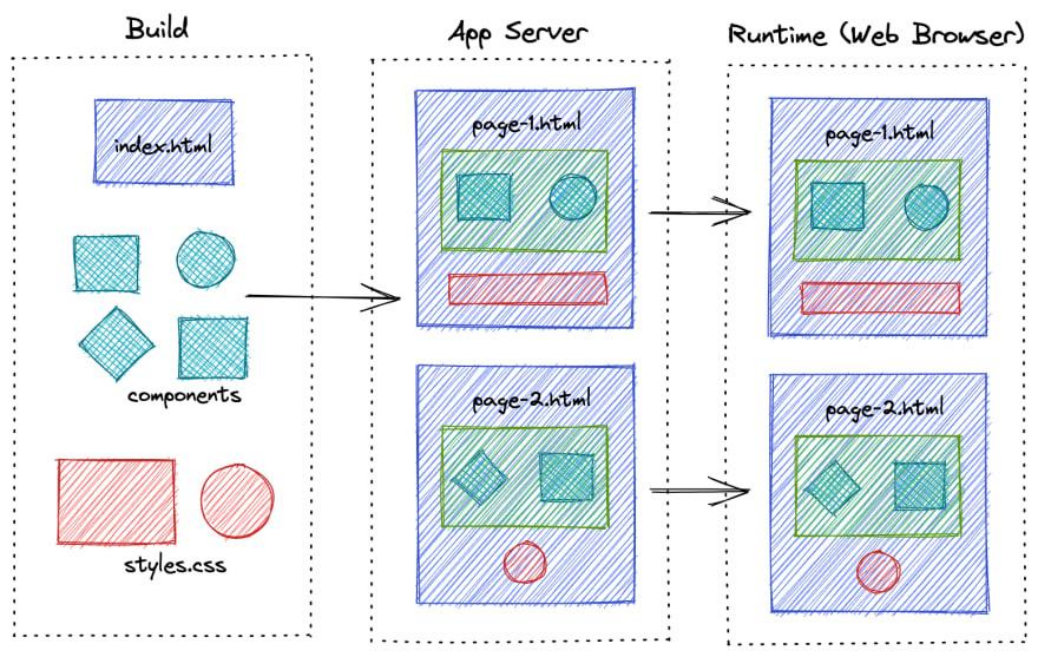
2FJ.io

SSG


Build
Static Server
Runtime (Web Browser)
page-1.html
page-2.html
page-1.html
page-2.html
page-1.html
page-2.html

2FJ.io

SSR


Build
App Server
Runtime (Web Browser)
index.html
components
styles.css
page-1.html
page-2.html
page-1.html
page-2.html

2FJ.io

Věděli jste že Angular umí...

- **Widget přístup:**
  - Využití Angular Elements pro implementaci specifických částí stránky
  - Modularita a opakované použití komponent v různých částech aplikace

- **Klasický přístup:**
  - Angular jako kompletní frontendový framework pro řízení celé klientské části aplikace
  - Komunikace s backendem prostřednictvím HTTP služeb nebo WebSocket pro efektivní výměnu dat v reálném čase

2FJ.io

Angular elements

- Angular Elements je funkcionalita v Angularu, která umožňuje vytvořit opakovaně použitelné komponenty zabalené jako vlastní HTML prvky (tzv. web components). Tyto komponenty lze pak použít v libovolné webové aplikaci, ať už je postavena pomocí Angularu, Reactu, Vue nebo jen čistého HTML a JavaScriptu.

2FJ.io

Rekapitulace JS

2FJ.io

Block bindings & hoisting

```
function getValue(condition) {

    if (condition) {
        var value = "blue";

        // other code

        return value;
    } else {

        // value exists here with a value of undefined

        return null;
    }

    // value exists here with a value of undefined
}
```

```
function getValue(condition) {

    var value;

    if (condition) {
        value = "blue";

        // other code

        return value;
    } else {

        return null;
    }
}
```

2FJ.io

Let vs Var

```
function getValue(condition) {

    if (condition) {
        let value = "blue";

        // other code

        return value;
    } else {

        // value doesn't exist here

        return null;
    }

    // value doesn't exist here
}
```

2FJ.io

Const vs Let

```
if(condition) {
    const maxItems = 5;

    //more code
}

//maxItems not accessible

console.log(maxItems)
```

2FJ.io

Template literals

```
let name = "Nicholas",
let message = `Hello, my name is ${name}.`;
console.log(message);
```

2FJ.io

Arrow funkce

```
# es5.syntax.js
function timesTwo(params) {
        return params * 2
}

timesTwo(4); // return 8
```

```
# es6.syntax.js
const timesTwo = (params) => rn params * 2
timesTwo(4);  // return 8

// function with more lines
const addValues = (x, y) => {
        return x + y
}

// function without params
( ) => 42

_  => 42
```

2FJ.io

Object destructing

```
const course = {
        name: “NodeJS“,
        school: “Gopas“,
        students: 4,
        emoji: “:-)“
};

const { name, school, students, emoji } = course;
```

ekvivalentní jako

```
const name = course.name;
const school = course.school;
const students = course.students;
const emoji = course.emoji;
```

2FJ.io

Spread operator

```
const arr1 = ['one', 'two'];
const arr2 = [...arr1, 'three', 'four', 'five'];

console.log(arr2);
//  Output:
//  ["one", "two", "three", "four", "five"]
```

```
const obj1 = { x : 1, y : 2 };
const obj2 = { z : 3 };

// add members obj1 and obj2  to obj3
const obj3 = {...obj1, ...obj2};

console.log(obj3); // {x: 1, y: 2, z: 3}
```

2FJ.io

Stavební bloky Angularu

- NgModuly
- Komponenty
- Direktivy
- Pipes
- Servisy

2FJ.io

Angular Module přístup vs. Standalone přístup

- V Angular 14 došlo k nejvýznamnějším změnám v celém frameworku díky „standalone API“.
- Předchozí verze používaly přístup založený na NgModule.
- V Angular 15 je standalone API stabilní pro použití v produkci.

  (Standalone přístup lze použít pro: Directives, Pipes, Components)

2FJ.io

Module přístup

- Hlavním důvodem pro zavedení NgModules bylo pragmatické rozhodnutí.

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

import { AppComponent } from './app.component';
[…]

@NgModule({
  imports: [BrowserModule, OtherModule],
  declarations: [AppComponent, OtherComponent, OtherDirective],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Compilation Context

NgModules provide the Compilation Context

2FJ.io

Standalone přístup

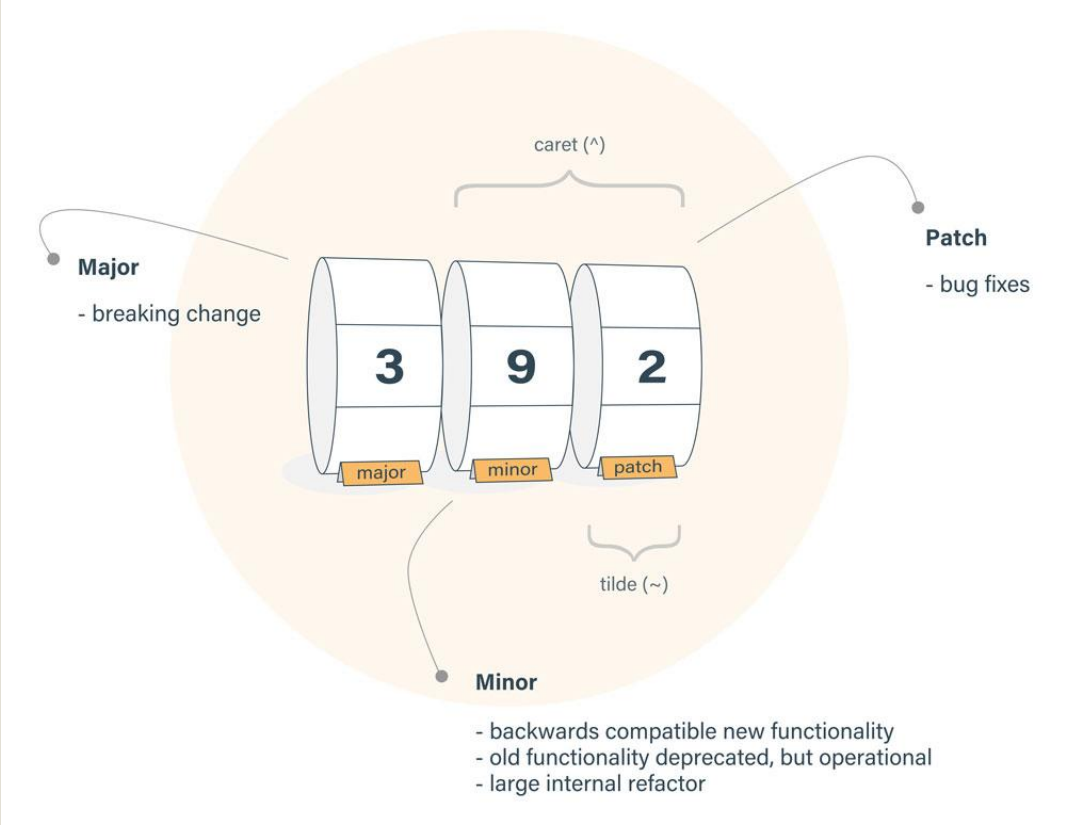
```
@Component({
  standalone: true,
  selector: 'app-root',
  imports: [
    RouterOutlet,
    NavbarComponent,
    SidebarComponent,
  ],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
    [...]
}
```

- Zjednodušení vývoje
- Zlepšení organizace kódu
- Usnadnění lazy loadingu
- Zvýšení flexibility
- Jednoduchá a modulární architektura, protože mít další modulární systém vedle toho, který poskytuje EcmaScript, nepůsobilo správně.

2FJ.io

Pojďme vyzkoušet Standalone komponenty!

2FJ.io


caret (^)
Major
- breaking change
Patch
- bug fixes
3
9
2
major
minor
patch
tilde (~)
Minor
- backwards compatible new functionality
- old functionality deprecated, but operational
- large internal refactor

2FJ.io

Standalone komponenty

Proč to používat?

- Zjednodušení vývoje malých i velkých webových aplikací.
- Lepší možnosti tree-shakingu, což vede k menší velikosti aplikace a lepšímu výkonu.
- Upozornění: Budete muset použít tuto funkcionalitu v novějších verzích Angularu. Module přístup může být deprecated.

2FJ.io

boostrapApplication

- bootstrapApplication je nová API zavedená v Angularu 14, která zjednodušuje proces bootstrappingu pro aplikace využívající Standalone Components. Umožňuje vývojářům inicializovat Angular aplikaci bez potřeby kořenového NgModule, čímž se proces spuštění stává přehlednějším a snižuje se množství opakovaného kódu.

2FJ.io

eventCoalescing

- Např. když je kliknuto na tlačítko, event bubbling způsobí, že se zavolají oba obslužné prvky události, což ve výchozím nastavení spustí dvě detekce změn.

```
<div (click)="parentHandler()">
  <button (click)="childHandler()"></button>
</div>
```

2FJ.io

typescript Zkopírovat kód

```typescript
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent)
  .catch(err => console.error(err));
```

typescript Zkopírovat kód

```typescript
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

2FJ.io

Párové x nepárové tagy

- <my-component></my-component>

Nyní

- <my-component />

2FJ.io

Pojďme vyzkoušet new control flow!

2FJ.io

Nový syntax vestavěných direktivy @if, @for, @switch

- Nová syntaxe kontrolního toku je optimalizována pro lepší výkon.

- Integrací kontrolního toku přímo do Angular frameworku nová syntaxe snižuje potřebu složitých konstrukcí v šablonách, což vede k čistšímu a snadněji udržitelnému kódu.

2FJ.io

trackBy

- Když použijete trackBy, Angular dokáže identifikovat jednotlivé položky podle jedinečného identifikátoru (například id). Díky tomu znovu vykreslí pouze ty položky, které se skutečně změnily, místo toho, aby vykresloval celý seznam od začátku.

2FJ.io

Pojďme vyzkoušet routing!

2FJ.io

provideRouter

- provideRouter() se používá pro standalone aplikace nebo moduly, poskytující funkčnější a modernější způsob konfigurace routování.
- Starý způsob: RouterModule.forRoot() se používalo k nastavení hlavních rout na úrovni kořene aplikace.
- Starý způsob: RouterModule.forFeature() se používalo k definování rout pro feature moduly, čímž se rozšiřovala konfigurace routování aplikace.

2FJ.io

Starý přístup: Routing Module

typescript

Zkopírovat kód

```typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { RouterModule, Routes } from '@angular/router';

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

const routes: Routes = [
  {
    path: 'feature',
    loadChildren: () => import('./feature/feature.module').then(m => m.FeatureModu
  },
  {
    path: '',
    redirectTo: '/home',
    pathMatch: 'full'
  },
  {
    path: '**',
    redirectTo: '/home'
  }
];

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot(routes)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2FJ.io

Načítání route a komponent

- Eager loading - načtení před spuštěním aplikace
- Lazy loading - loading takzvaně on demand
- Preloading - načtení po spuštění aplikace

2FJ.io

preloadingStrategy

- withPreloading
  - Umožňuje nastavit strategii přednačítání, kterou chcete použít. Strategie se konfiguruje poskytnutím odkazu na třídu, která implementuje PreloadingStrategy.

2FJ.io

preloadingStrategy

Proč to použít?

- Upřednostnění kritických cest: Strategie přednačítání umožňují upřednostnit načítání kritických modulů, které budou pravděpodobně přístupné krátce po počátečním načtení, což zlepšuje celkový výkon a uživatelský zážitek.

2FJ.io

Jak konfigurovat standalone komponenty?

2FJ.io

K čemu jsou statické metody v ngModulech? Např. forRoot a forChild?

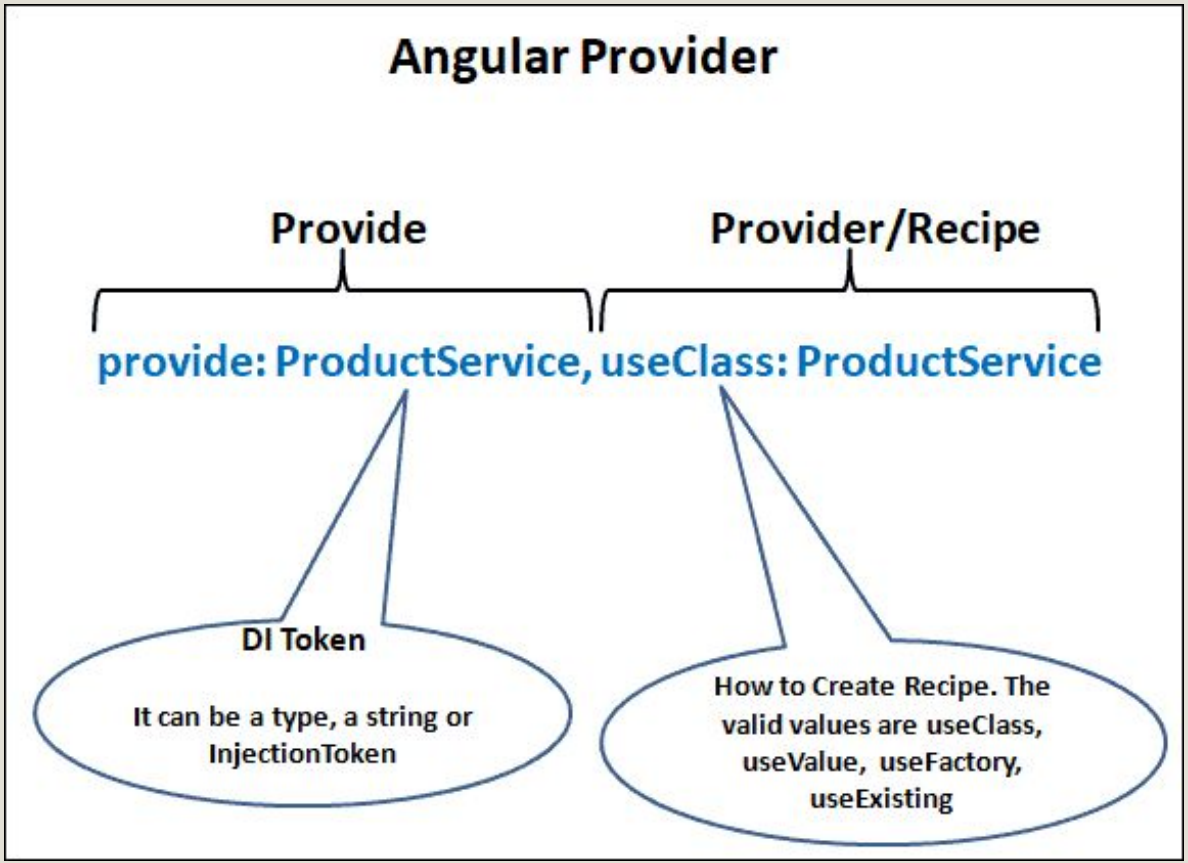
2FJ.io

Konfigurovatelný ngModule

Proč se to používalo?

- Opakované použití – Modul můžete použít v různých částech aplikace s odlišným nastavením.
- Oddělení konfigurace od implementace – Logika se soustředí do jednoho místa a konfigurace je jasně oddělená, což usnadňuje údržbu a čitelnost.

2FJ.io


Angular Provider
Provide
Provider/Recipe
provide: ProductService, useClass: ProductService
DI Token
It can be a type, a string or InjectionToken
How to Create Recipe. The valid values are useClass, useValue, useFactory, useExisting

2FJ.io

Injection token

Co to je?

- Jedinečný identifikátor pro DI: Slouží k bezpečnému injektování hodnot a konfigurací.
- InjectionToken umožňuje injektování hodnot, které nemají žádnou runtime reprezentaci (např. nemají vlastní třídu nebo instanci, kterou by Angular mohl použít)

2FJ.io

Pojďme se podívat na používání services a HTTP client

2FJ.io

HttpClient

- Integrovaná třída služeb dostupná v balíčku @angular/common/http
- Využívá implementace RxJS Observables
- Poskytuje operace GET, POST, PUT, DELETE

2FJ.io

Promise

- Promise je objekt, který představuje hodnotu, která ještě nemusí být dostupná v budoucnu.
- Příklad: HTTP request z klienta na server.

2FJ.io

Observable

- Wrapper nad datovým zdrojem, na který jsme schopni poslouchat pomocí Observer
- Příklad: Datový tok telefonního hovoru

2FJ.io

Rozdíly mezi Promises a Observables

- Promise vrátí pouze jednu hodnotu nebo chybu a je navržena pro jednorázové asynchronní operace.
- Observable může emitovat více hodnot v průběhu času, což ho činí vhodným pro sledování více událostí nebo proudů dat.

2FJ.io

Dependency injection

- Vkládání závislostí (anglicky Dependency injection (DI)) je v objektově orientovaném programování technika pro vkládání závislostí mezi jednotlivými komponentami programu tak, aby jedna komponenta mohla používat druhou, aniž by na ni měla v době sestavování programu referenci.

2FJ.io

typescript

Zkopírovat kód

```typescript
class UserService {
  sayHi() {
    console.log('hi');
  }
}

class Component {
  public userService: UserService;

  constructor() {
    this.userService = new UserService(); // Manually creating the instance
  }
}

const component = new Component();
component.userService.sayHi();
```

2FJ.io

typescript

Zkopírovat kód

```typescript
class UserService {
  sayHi() {
    console.log('hi');
  }
}

class Component {
  constructor(public userService: UserService) {}
}

class Injector {
  private container = new Map();

  constructor(private providers: any[] = []) {
    this.providers.forEach((Service) => this.container.set(Service, new Service()))
  }

  get(service: any) {
    const serviceInstance = this.container.get(service);

    if (!serviceInstance) {
      throw new Error('No provider found');
    }

    return serviceInstance;
  }
}

const injector = new Injector([UserService]);
const component = new Component(injector.get(UserService));
component.userService.sayHi();
```

2FJ.io

inject

- V Angularu 14 bylo zavedeno inject. Umožňuje vynechat explicitní typování — nechajte TypeScript, aby to udělal za vás.
- Rozšiřování tříd je jednodušší, aniž by bylo nutné předávat každý argument do konstruktoru základní třídy.
- Není potřeba deklarovat všechny závislosti v konstruktoru.

2FJ.io

typescript

Zkopírovat kód

```typescript
import { inject } from '@angular/core';

class BaseComponent {
  constructor(public configService: ConfigService) {}
}

class ExtendedComponent extends BaseComponent {
  private logger = inject(LoggerService); // No need to modify the constructor

  someMethod() {
    this.logger.log('Using logger from inject in extended class');
  }
}
```

2FJ.io

typescript

Zkopírovat kód

```typescript
import { Injector } from '@angular/core';

// This is a simplified example; the real implementation is much more complex.
function inject<T>(token: any): T {
  const injector = Injector.getCurrent(); // Get the current injector
  if (!injector) {
    throw new Error('No injector found!');
  }
  return injector.get(token);
}
```

2FJ.io

inject

Proč používat?

- Více se hodí jako funkcionální programování
- Méně boilerplate kódu

2FJ.io

Injektory

- Angular 18 má dva typy injector hierarchií:
  - Environment hierarchy
  - Element hierarchy

- Dříve: Module přístup Angular
  - Module hierarchy
  - Element hierarchy

2FJ.io

```
const routes: Routes = [
   { path: ‘user’, component: UserComponent, providers: [ UserService ] }
]
```

Environment hierarchy

- NullInjector
- PlatformInjector
- RootInjector
- EnvironmentInjector

```
@Injectable({providedIn: 'root'})
export class UserService {
  name = 'John'
}
```

```
bootstrapApplication(AppComponent, { providers: [UserService] });
```

2FJ.io

NullInjector

2FJ.io

PlatformInjector

typescript

Zkopírovat kód

```typescript
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app/app.module';

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule).then(ref => {
  const platformInjector = ref.injector;
  // platformInjector can now be used to get platform-level services
});
```

2FJ.io

RootInjector

typescript

Zkopírovat kód

```typescript
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class RootService {
  constructor() {}
}
```

2FJ.io

EnvironmentInjector

- EnvironmentInjector: Nabízí pokročilou flexibilitu pro dynamické vytváření komponent a izolované testovací prostředí.
-

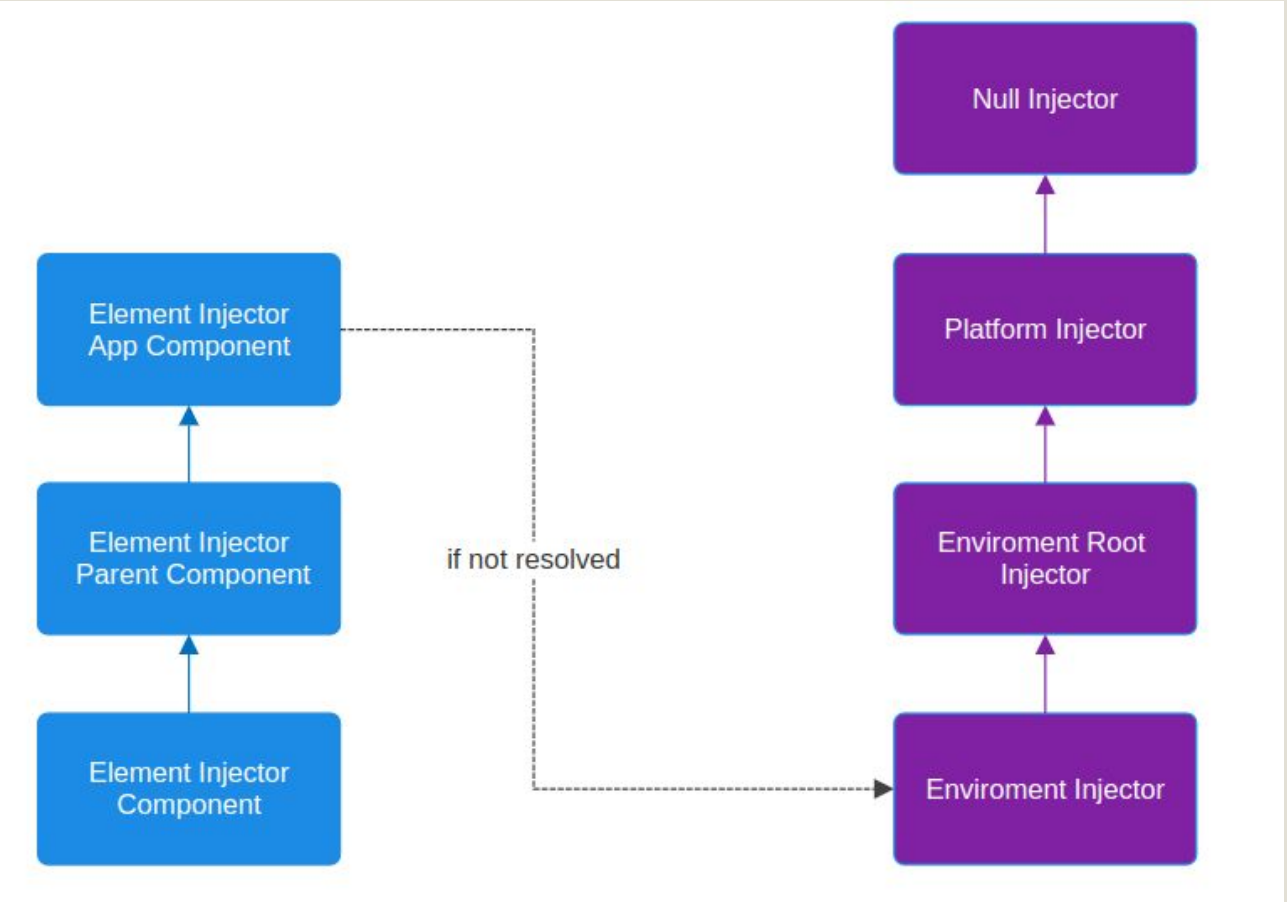
2FJ.io

Element injector

- Element Injector — Registruje závislosti definované v providers uvnitř dekorátorů @Component nebo @Directive. Tyto závislosti jsou dostupné pro komponentu a její podřízené komponenty.

```
1  @Component({
2    ...
3    providers: [UserService]
4  })
5  export class UserComponent {}
```

2FJ.io


Null Injector
Element Injector
App Component
Platform Injector
Element Injector
Parent Component
if not resolved
Enviroment Root
Injector
Element Injector
Component
Enviroment Injector

2FJ.io

Co takhle volat servisy rovnou v routingu?

2FJ.io

ResolveFn

- Lze jej použít s routerem k načtení dat během navigace.
- Route resolvery založené na třídách jsou zastaralé ve prospěch funkčních resolverů.

2FJ.io

ResolveFn

Proč je používat?

- Přednačtením dat se můžete vyhnout zobrazení prázdných stavů nebo načítacích indikátorů v rámci komponenty, což vede k plynulejšímu a bezproblémovějšímu uživatelskému zážitku.

2FJ.io

Osnova dnešního kurzu

- Rekapitulace základních stavebních prvků
- Úvod do signálů
- Signály, Efekty, Computed
- Signály vs RxJS

2FJ.io

Předmluva

- V únoru 2023 tým Angularu představil Signály jako odpověď na požadavek komunity.
- Od té doby se svět začal měnit – a Angular také…

2FJ.io

Reaktivita obecně

- V moderních webových aplikacích je jedním z hlavních úkolů frameworku synchronizace změn v datovém modelu aplikace s uživatelským rozhraním.
- Tento mechanismus se nazývá „reaktivita“.

2FJ.io

Co je to change detection v Angular?

2FJ.io

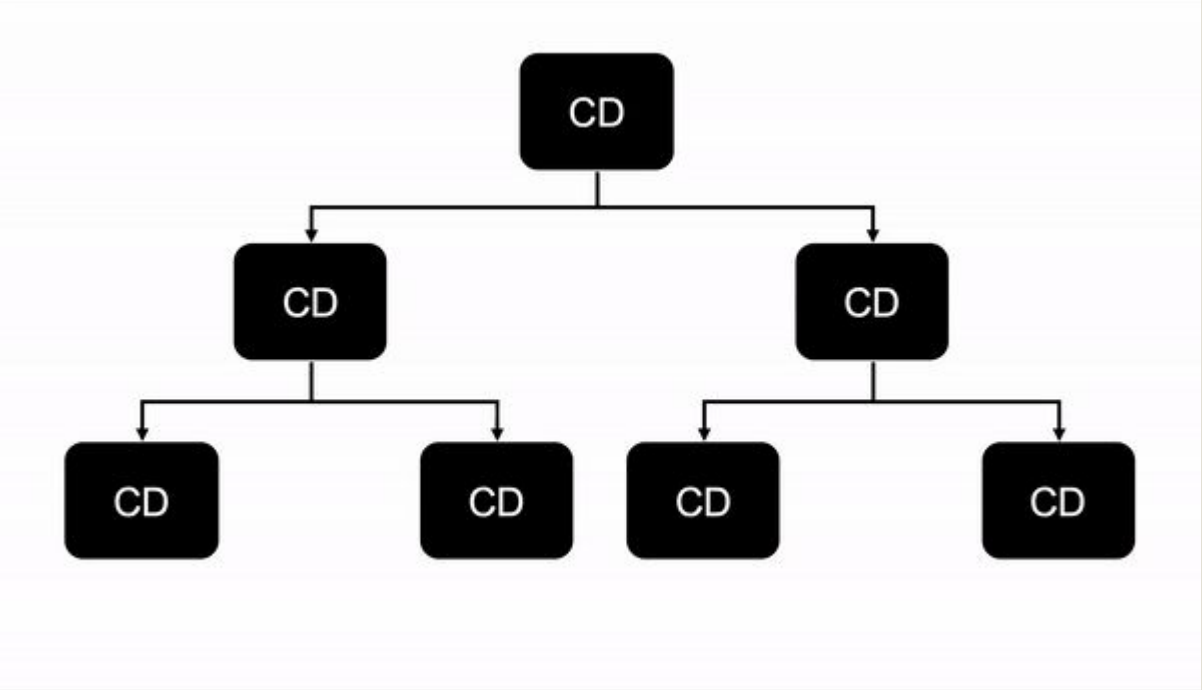
Change detection

- Detekce změn v Angularu je mechanismus, který zajišťuje, že uživatelské rozhraní (UI) je vždy synchronizováno s daty aplikace. Zahrnuje zajištění, že se data v aplikaci změní, a aktualizaci UI, které tyto změny odráží.

2FJ.io

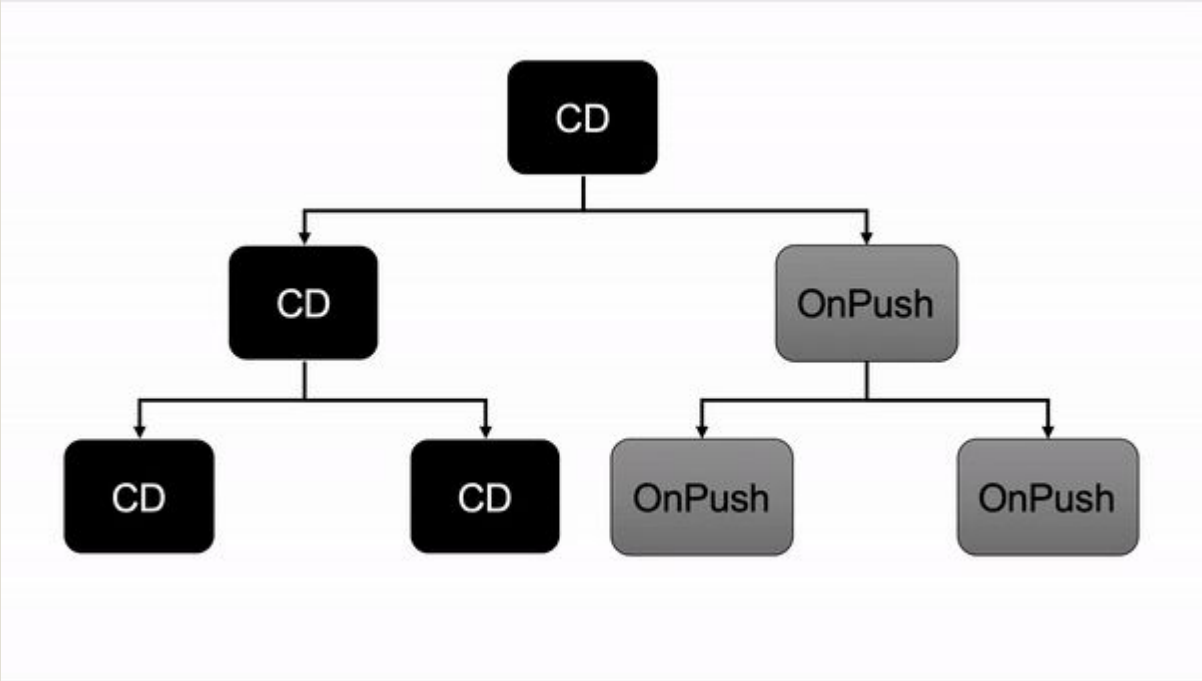
Reaktivita v Angular

- Zone.js
- Change detection



2FJ.io

Change detection OnPush



2FJ.io

Co je to Zone.js

- Zone.js se stará o zachycení a sledování asynchronních operací, jako jsou asynchronní funkce, asynchronní požadavky na server nebo události uživatelského rozhraní. Když je zahájena asynchronní operace, vytvoří se nová zóna, která sleduje všechny události v rámci této operace. Když dojde k události, Zone.js upraví stav aplikace a zajistí, aby všechny ostatní operace v této zóně tyto změny sledovaly.

2FJ.io

Zone.js monkey patching

```
15
16  // Save the original console.log function
17  const originalConsoleLog = console.log;
18
19  // Override console.log with a custom function
20  console.log = function(...args) {
21  // Call the original console.log function with the arguments
22  originalConsoleLog(...args);
23
24  // Log a warning message
25  console.warn('This method was monkey patched to trigger change detection in Angular');
26
27  // Get the root component of the Angular app
28  const appRootComponent = ng.getComponent(document.getElementsByTagName('app-root')[0]);
29
30  // Trigger change detection on the root component
31  appRootComponent.cdr.detectChanges();
32  };
```

2FJ.io

Angular reaktivita krok za krokem

- Inicializace aplikace
  - Když Angular aplikace startuje, Zone.js se inicializuje a začíná monitorovat asynchronní operace.
- Zpracování událostí
  - Když nastane událost (např. uživatel klikne na tlačítko), Zone.js tuto událost zachytí.
  - Zone.js sleduje všechny probíhající asynchronní úkoly.
- Dokončení úkolu
  - Jakmile je asynchronní úkol (např. HTTP požadavek) dokončen, Zone.js detekuje jeho konec.
  - Zone.js poté spustí detekci změn v Angularu.
- Provedení detekce změn
  - Mechanismus detekce změn v Angularu zkontroluje strom komponent na změny.
  - Aktualizuje zobrazení s nejnovějším stavem aplikačního modelu.

2FJ.io

Angular bez Zone.js

```
platformBrowserDynamic().bootstrapModule(AppModule, {
  ngZone: 'noop'
}).catch(err => console.error(err));
```

2FJ.io

Co je špatného na change detection v Angularu?

2FJ.io

Co je na detekci změn v Angularu špatného?

- Výchozí strategie Angularu je spouštět detekci změn na celém stromu komponent.
- Angular nemá informace o tom, které části stavu aplikace se skutečně změnily.

2FJ.io

Proč onPush není řešení?

- Detekce změn vždy začíná od kořenové komponenty. Komponenty s nastavením OnPush brání kontrolování svých podřízených komponent. To vytváří chaos v architektuře aplikace.

2FJ.io

Co je špatného na Zone.js?

- Zone.js neposkytuje informace o změnách konkrétní komponenty.
- Pouze oznamuje, že došlo k události.

2FJ.io

Co jsou to Signals?

- Signál je wrapper kolem hodnoty, který informuje zainteresované consumer, když se tato hodnota změní.
- Signály mohou obsahovat jakoukoli hodnotu, od primitiv až po složité datové struktury.

2FJ.io

Jak bude vypadat nový Angular bez Zone.js, který používá Signály?

- V moderním Angularu jsou všechna klíčová data zapouzdřena a používána jako signály.
- Signály se rychle staly základní a nejdůležitější součástí Angularu.

2FJ.io

Definování Signálů

- Angular signály začínají počáteční hodnotou.
- Při vyvolání vrátí aktuální hodnotu signálu.

```
quantity_ = signal<number>(1)
selectedCar_ = signal<Car>(null)
userMsg_ = signal({ txt: '', type: '' })
cars_ = signal<Car[]>([])


<h5> Quantity: {{ quantity_() }} </h5>
<user-msg [msg]="userMsg_()"></user-msg>
```

2FJ.io

Nastavení hodnoty signálu

- Funkce signálu vrací WritableSignal<T>, která umožňuje upravit hodnotu.

```
// Replace the signal value
const movies_ = signal<Movie[]>([])
movies_.set([{ name: 'Fight club' }])

// Derive a new value
const someCount_ = signal<Number>(0)
someCount_.update(n => n + 1)

// Perform internal mutation of arrays and other objects
movies_.mutate(list => {
    list.push({name: 'Aba Ganuv'})
})
```

2FJ.io

Metoda set() signálu

- Účel: Přímo přiřadí signálu novou hodnotu.
- Použití: Tuto metodu použijte, když chcete nahradit aktuální hodnotu signálu zcela novou hodnotou.

2FJ.io

Metoda update() signálu

- Účel: Aktualizuje hodnotu signálu na základě jeho aktuální hodnoty.
- Použití: Tuto metodu použijte, když potřebujete vypočítat novou hodnotu z aktuální hodnoty. Jako argument přijímá funkci, která obdrží aktuální hodnotu a vrátí novou hodnotu.

2FJ.io

Computed()

- Computed() signály automaticky sledují závislosti a reaktivně aktualizují své hodnoty.
- Computed() nelze upravit ručně.

```
const counter_ = signal(0)

// Automatically updates when `counter` changes:
const isEven_ = computed(() => counter_() % 2 === 0)
```

2FJ.io

Effect()

- Effect() spouští funkci s vedlejším efektem Kdykoli některá ze závislostí vytvoří novou hodnotu, efekt se spustí.

```
const counter_ = signal(0)
effect(() => console.log('The counter is:', counter_()))
// The counter is: 0

counter_.set(1)
// The counter is: 1
```

2FJ.io

Effect() - desktruktor

- Efekty jsou automaticky zničeny, když je zničen jejich kontext.
- Efekty lze zničit ručně zavoláním metody destroy().

2FJ.io

Shrnutí základů

- Signál: Uchovává hodnotu a informuje konzumenta, když se hodnota změní.
- Computed signál: Představuje hodnotu odvozenou od jiných signálů.
- Effect: Spustí funkci, kdykoli se změní signály, na kterých závisí.

2FJ.io

Jdeme si hrát se signály!

2FJ.io

Děkuji za pozornost