Advanced Routing

Лектор: Петър Маламов

Lazy-loading

Lazy-loading

- Какво е lazy-loading ?
- Как да използваме lazy-loading ?

Какво е lazy-loading ?

Техника за оптимизация на приложенията, при която част от модулите и ресурсите се зареждат само когато са необходими, а не предварително при стартирането на приложението.

Това води до по-бързо първоначално зареждане на приложението, тъй като първоначално свалените пакети са доста по-малки.

Как да използваме lazy-loading ?

Lazy-loading се осъществява с помощта на **dynamic imports**.

Израз, с който можем да заредим ECMAScript модул асинхронно и динамично.

```
import('./path-to-my-module.js');
```

Прични за lazy-loading

- Намалява първоначалния размер на свалените пакети
- Отлага свалянето на даден пакет, докато не е нужен
- Improves performance

Ползи

- По-бързо първоначално време за зареждане
- По-добро управление на ресурсите
- Подобрява SEO

Негативи

- Забавяне при първо зареждане на lazy-loaded модул.
- Може да навреди на UX.
- Усложнява debugging

Lazy-loading в Angular

- Lazy-loading в Angular
- Lazy-loading modules
- Lazy-loading components
- Упражнение

Lazy-loading в Angular

По подразбиране всички компоненти в Angular се зареждат eagerly. Това означава, че щом приложението се зареди, всички модули и компоненти също се зареждат заедно с него.

За да не се налага да зареждаме всичко при началното посещение на приложението, Angular ни предоставя начин да отложим зареждането на определени компоненти или модули.

Lazy-loading в Angular

Най-често използваният подход за имплементиране на lazy-loading е чрез Angular Router. Той ни позволява да зареждаме компоненти или модули в зависимост от това на коя страница се намира потребителят.

Има два начина, по които можем да се възползваме от Angular Router:

- Lazy-loading modules
- Lazy-loading components

Lazy-loading modules

За да направим един модул lazy-loaded трябва:

1. Да използваме функцията **loadChildren**, когато го дефинираме като маршрут.
2. Да създадем маршрути за модула.

TS app.routes.ts

```ts
import { Routes } from '@angular/router';
import { DashboardComponent } from './dashboard/dashboard.component';
import { AboutComponent } from './about/about.component';

export const routes: Routes = [
  {
    path: 'dashboard',
    component: DashboardComponent,
  },
  {
    path: 'about',
    component: AboutComponent,
  }
];
```

TS app.routes.ts

```ts
import { Routes } from '@angular/router';
import { AboutComponent } from './about/about.component';

export const routes: Routes = [
  {
    path: 'dashboard',
    loadChildren: () =>
      import('./dashboard/dashboard.module').then((m) => m.DashboardModule),
  },
  {
    path: 'about',
    component: AboutComponent,
  },
];
```

TS dashboard.module.ts

```ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DashboardComponent } from './dashboard.component';
import { MatTabsModule } from '@angular/material/tabs';

@NgModule({
  declarations: [DashboardComponent],
  imports: [CommonModule, MatTabsModule],
})
export class DashboardModule {}
```

TS dashboard.module.ts

```ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DashboardComponent } from './dashboard.component';
import { MatTabsModule } from '@angular/material/tabs';
import { RouterModule, Routes } from '@angular/router';

const routes: Routes = [
  {
    path: '',
    component: DashboardComponent,
  },
];

@NgModule({
  declarations: [DashboardComponent],
  imports: [CommonModule, RouterModule.forChild(routes), MatTabsModule],
})
export class DashboardModule {}
```

Lazy-loading components

Единственото нещо, което трябва да променим, за да направим един компонент lazy-loaded, е да използваме функцията loadComponent при дефинирането му като маршрут.

TS app.routes.ts

```ts
import { Routes } from '@angular/router';
import { DashboardComponent } from './dashboard/dashboard.component';
import { AboutComponent } from './about/about.component';

export const routes: Routes = [
  {
    path: 'dashboard',
    component: DashboardComponent,
  },
  {
    path: 'about',
    component: AboutComponent,
  }
];
```

TS app.routes.ts

```ts
import { Routes } from '@angular/router';
import { DashboardComponent } from './dashboard/dashboard.component';

export const routes: Routes = [
  {
    path: 'dashboard',
    component: DashboardComponent,
  },
  {
    path: 'about',
    loadComponent: () =>
      import('./about/about.component').then((m) => m.AboutComponent),
  },
];
```

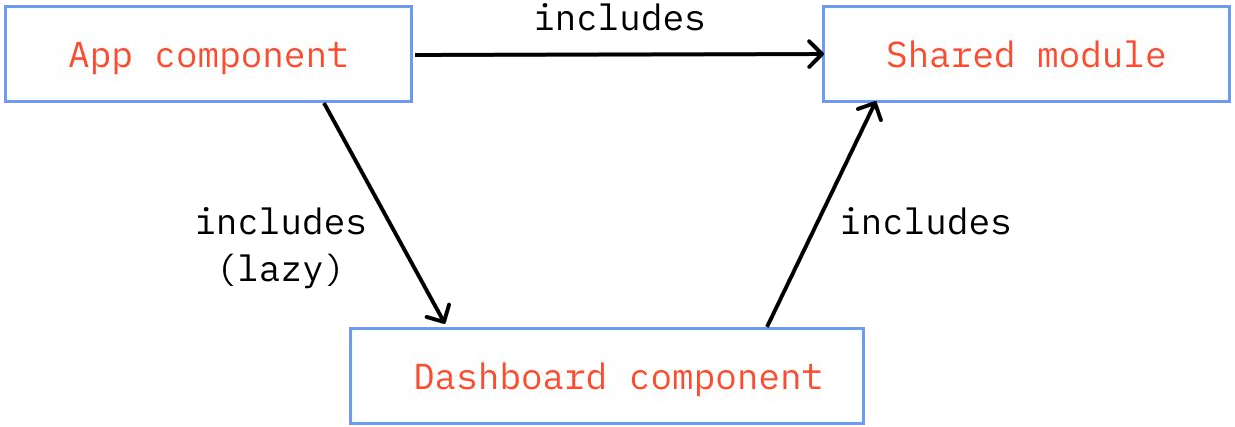
Упражнение

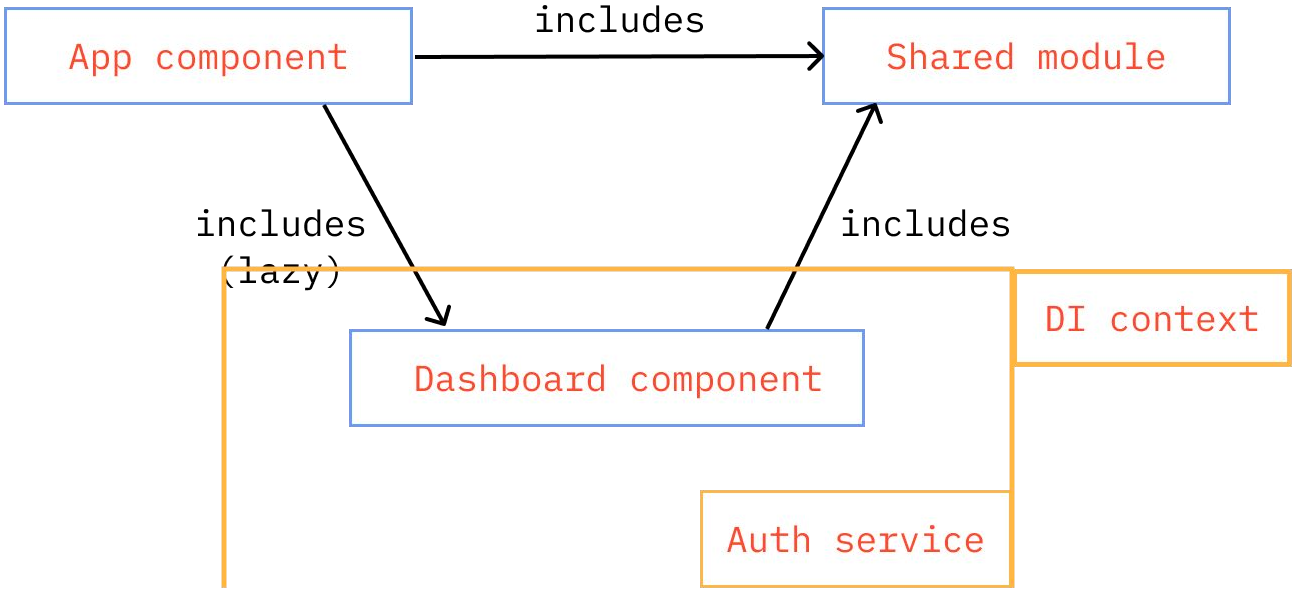
github.com/Diadraw-interns/angular_course -> 04-advanced-routing

Направете компонентите DashboardComponent и AboutComponent да се зареждат динамично.

```
export const routes: Routes = [
  {
    path: 'sing-in',
    component: SigninComponent,
  },
  {
    path: 'dashboard',
    component: DashboardComponent,
  },
  {
    path: 'about',
    component: AboutComponent,
  },
];
```

Проблеми при lazy-loading





forRoot() pattern

Начин, по който можем да разделим един модул на две части, с което да сме сигурни, че зависимостите, които предоставя, са singleton.

TS auth.model.ts

```ts
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AuthService } from './auth.service';
import { SigninComponent } from './signin/signin.component';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [SigninComponent],
  imports: [CommonModule, FormsModule],
  providers: [AuthService],
})
export class AuthModule {
}
```

TS auth.model.ts

```ts
import { ModuleWithProviders, NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AuthService } from './auth.service';
import { SigninComponent } from './signin/signin.component';
import { FormsModule } from '@angular/forms';

You, 1 second ago | 1 author (You)
@NgModule({
  declarations: [SigninComponent],
  imports: [CommonModule, FormsModule],
  providers: [],
})
export class AuthModule {
  static forRoot(): ModuleWithProviders<AuthModule> {
    return {
      ngModule: AuthModule,
      providers: [AuthService],
    };
  }
}
```

TS app.config.ts

```ts
import { ApplicationConfig } from '@angular/core';
import { provideRouter } from '@angular/router';

import { routes } from './app.routes';
import { provideAnimationsAsync } from '@angular/platform-browser/animations/async';

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes), provideAnimationsAsync()],
};
```

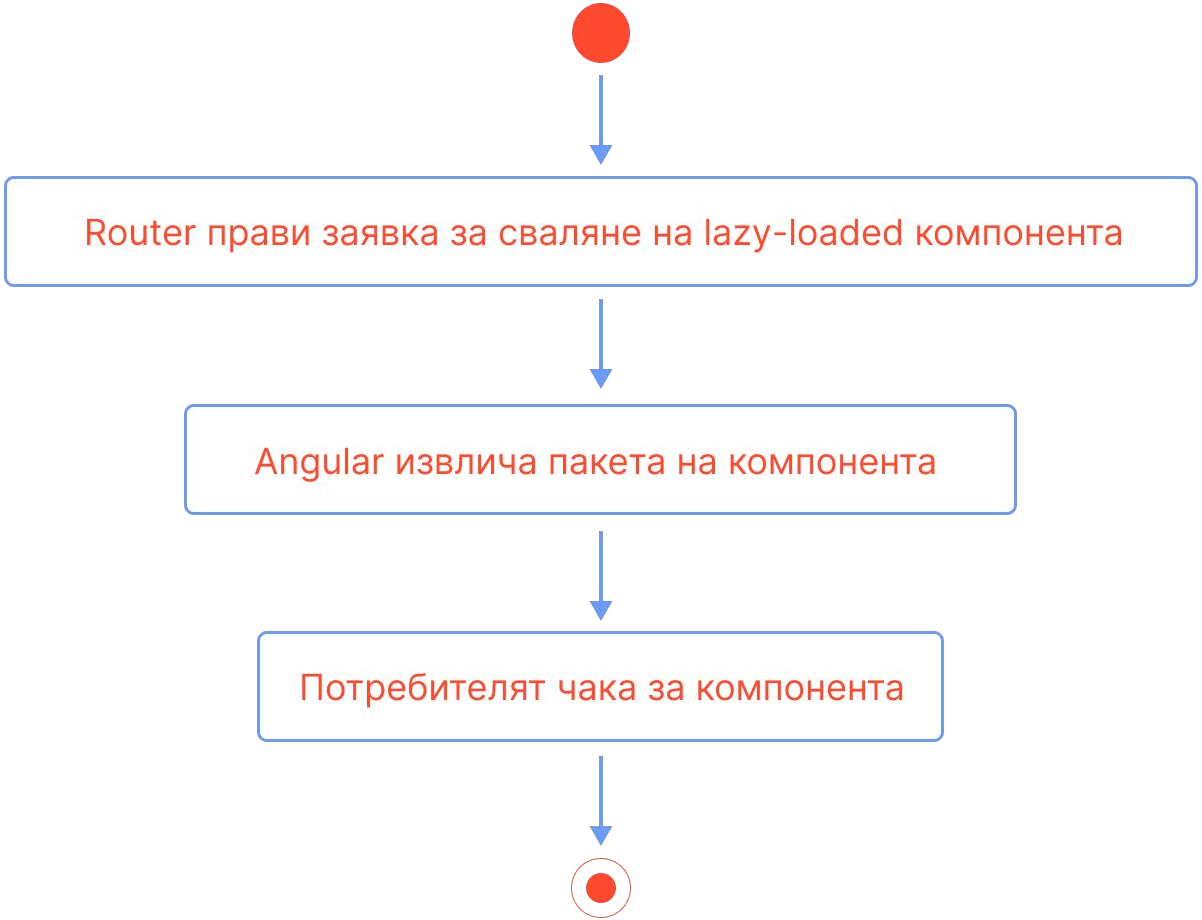
TS app.config.ts

```ts
mport { ApplicationConfig, importProvidersFrom } from '@angular/core';
mport { provideRouter } from '@angular/router';

mport { routes } from './app.routes';
mport { provideAnimationsAsync } from '@angular/platform-browser/animations/async';
mport { AuthModule } from './auth/auth.module';

xport const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(routes),
    provideAnimationsAsync(),
    importProvidersFrom(AuthModule.forRoot()),
  ],
;
```

Алтернативи

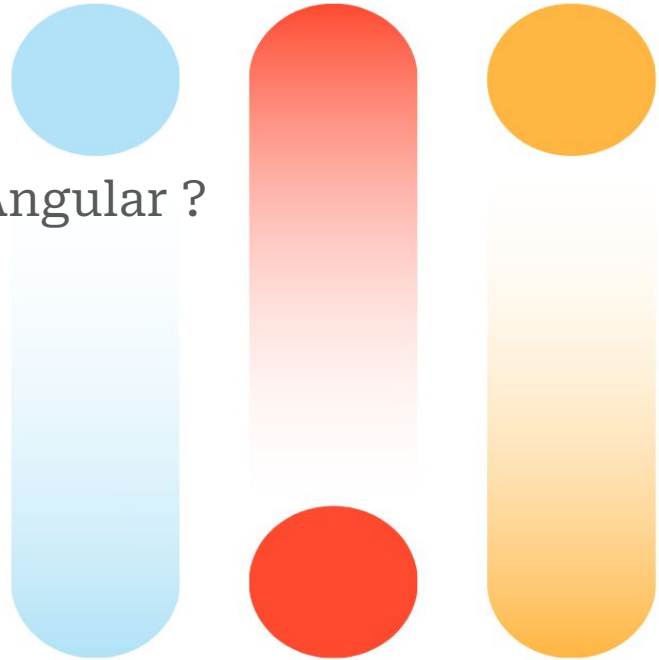
- Деклариране на зависимости с providedIn = root.
- Деклариране на зависимости в ApplicationConfig.
- Създаване на отделен модул само за зависимости.

Какво се случва когато навигираме към нова страница ?



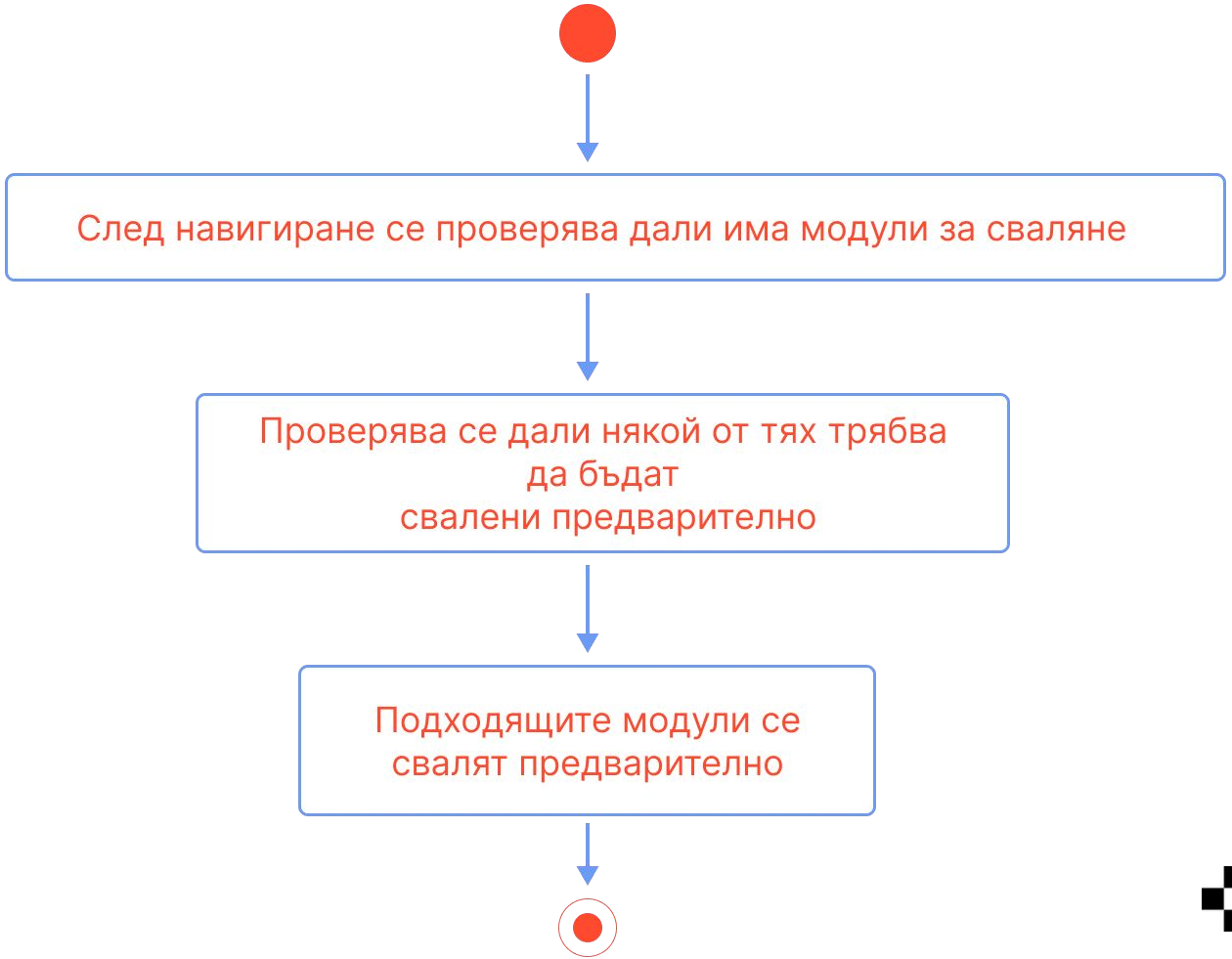
Preloading

- Какво е Preloading ?
- Как да имплементираме Preloading в Angular ?
- Preloading strategies
- Упражнение



Какво е Preloading ?

Техника, която позволява на приложението да зарежда lazy-loaded модули/компоненти на заден фон, вместо да се чака да бъдат заредени, когато потребителят навигира към съответния маршрут.



Preloading в Angular

Преди да можем да използваме **Preloading** в Angular, трябва да определим стратегия, по която ще се зареждат предварително модулите и компонентите.

TS app.config.ts

```ts
import { ApplicationConfig } from '@angular/core';
import {
  PreloadAllModules,
  provideRouter,
  withPreloading,
} from '@angular/router';

import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes, withPreloading(PreloadAllModules))],
};
```

TS app-routing.module.ts

```ts
@NgModule({
  imports: [
    RouterModule.forRoot(routes, {
      preloadingStrategy: PreloadAllModules,
    }),
  ],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Preloading strategies

Стратегия, с която можем да определим кога и как да заредим модули или компоненти предварително.

Preloading strategies

Preload None	Preload All	Custom Preload Startegy
По подразбиране модулите/ компонентите, които са lazy-loaded няма да бъдят заредени предварително	Всички модули ще бъдат предварително заредени. Полезно, но твърде агресивна стратегия!	Зареждане на модули/компоненти предварително базирано на персонализирана логика.

Preload all

TS app.config.ts

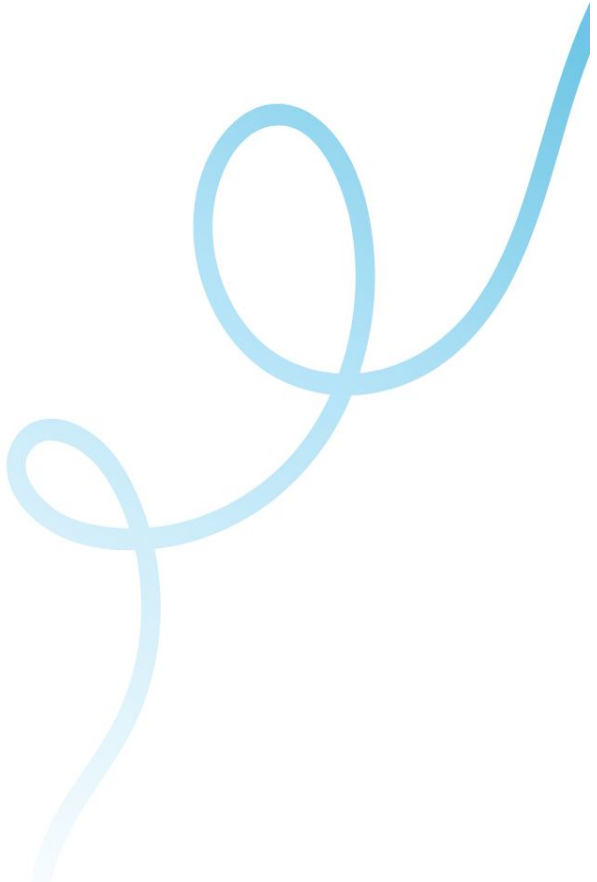
```ts
import { ApplicationConfig } from '@angular/core';
import {
  PreloadAllModules,
  provideRouter,
  withPreloading,
} from '@angular/router';

import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes, withPreloading(PreloadAllModules))],
};
```

Custom Preload Strategy

1. Трябва да имплементираме интерфейса PreloadingStrategy.
2. Трябва да създадем персонализирана логика как да зареждаме модулите/компонентите.
3. Трябва да я приложим към маршрутите.

Идеи за Custom Preload Strategy

- Opt in or out
- Базирано на скортоста на интернета.
- Спрямо действията на потребителя.



Deferrable Views

- Какво е Deferrable view ?
- Как да използваме Deferrable view?
- Упражнение

Какво е Deferrable view ?

Начин, по който можем да lazy-load-нем компоненти, без да е нужно те да са в маршрут.

Кои компоненти са defer-loadable ?

- Standalone components
- Компоненти, които не са реферирани никъде другаде освен в defer scope на темплейта.

Как да използваме Deferrable view?

TS component.ts

```
@defer{
    <app-invoices />
} @placeholder{
    <div>No content loaded yet</div>
}@loading{
    <div>Loading...</div>
}@error{
    <div>There was an error</div>
}
```

@defer

- По подразбиране се задейства, когато браузърът е **idle.**
- Може да променяме начина, по който се задейства.

TS component.ts

```
@defer{
    <app-team></app-team>
}
```

@placeholder

- Опционално. (добра практика е обаче винаги да имаме)
- Eagerly loaded.
- Показва се преди defer блока да се задейства.
- Може да приема опционален параметър **minimum**
  - Определя минималното време за което **placeholder** трябва да е видим.

TS component.ts

```
@defer{
<app-team></app-team>
} @placeholder (minimum 500ms) {
<p>Placeholder content</p>
}
```

@loading

- Опционално.
- Показва се след като defer е задействан и свалянето на компонента започне.
- Може да приема 2 опционални параметъра:
  - after - определя времето, след което да се покаже съдържанието в **loading**.
  - minimum - определя минималното време за което **loading** трябва да е видим

TS component.ts

```
@defer{
    <app-team></app-team>
} @loading (after 100ms; minimum 1s) {
    <div>Loading...</div>
}
```

@error

- Опционално.
- Показва се ако зареждането на съдържанието в defer върне грешка.

TS component.ts

```
@defer{
    <app-team></app-team>
} @error {
    <p>Failed to load team</p>
}
```

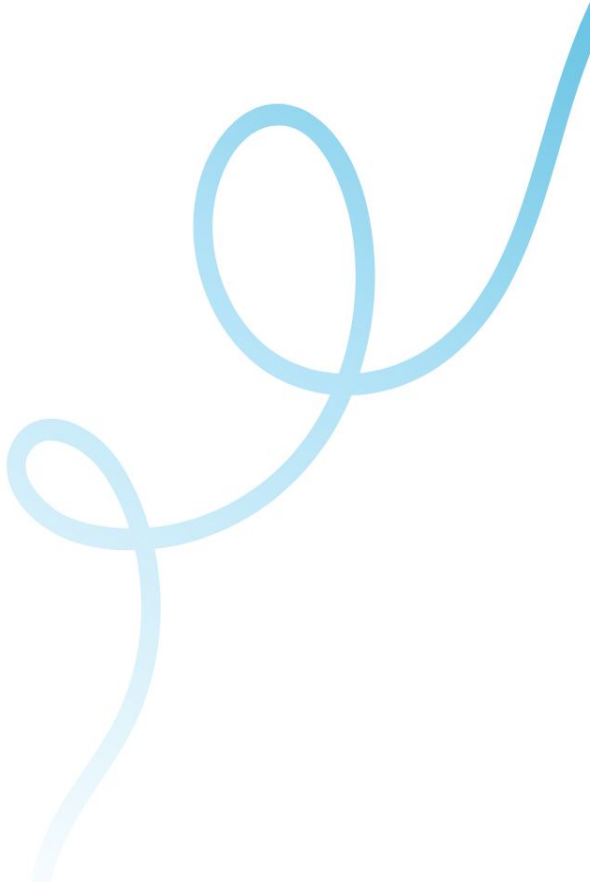
Начини за задействане на Deferrable view

Начини за задействане

- on - използва се с предварително зададени начини за задействане.
- when - определя начина за задействане, чрез израз, който връща булева

On

- idle
- viewport
- interaction
- hover
- immediate
- timer



On idle

Когато използваме **idle**, зареждането на съдържанието започва когато браузърът достигне idle състояние.

Браузъра използва функцията **requestIdleCallback**. Нейната работа е да добавя функции в опашка, която браузърът ще използва щом има свободно време.

TS component.ts

```
@defer (on idle) {
    <app-team></app-team>
}
```

On viewport

Когато използваме **viewport**, зареждането на съдържанието започва, когато то влезе във видимия прозорец на потребителя. Това се определя от IntersectionObserver API

Ако използваме viewport, без да му подадем елемент, за който да следи, Angular ще използва съдържанието, зададено в **@placeholder**, по подразбиране.

TS component.ts

```
@defer (on viewport) {
    <app-team></app-team>
} @placeholder {
    <div>Placeholder</div>
}
```

TS component.ts

```
<div #element>Hello!</div>

@defer (on viewport(element)) {
    <app-team></app-team>
} @placeholder {
    <div>Placeholder</div>
}
```

On interaction

Когато използваме **interaction**, зареждането на съдържанието започва, когато потребителят взаимодейства чрез **click** или **keydown**.

Ако не определим елемент, за който да следим, Angular ще използва съдържанието в **@placeholder**

TS component.ts

```
@defer (on interaction) {
    <app-team></app-team>
} @placeholder {
    <div>Placeholder</div>
}
```

TS component.ts

```
<button type="button" #element>button!</button>

@defer (on interaction(element)) {
    <app-team></app-team>
} @placeholder {
    <div>Placeholder</div>
}
```

On hover

Когато използваме **hover**, зареждането на съдържанието започва, когато мишката на потребителя се намира над него. (**mouseenter** and **focusin**)

Ако не определим елемент, за който да следим, Angular ще използва съдържанието в **@placeholder**

TS component.ts

```
@defer (on hover) {
    <app-team></app-team>
} @placeholder {
    <div>Calendar placeholder</div>
}
```

TS component.ts

```
<div #element>Load on hover</div>

@defer (on hover(element)) {
    <app-team></app-team>
} @placeholder {
    <div>Calendar placeholder</div>
}
```

On immediate

Когато използваме **immediate**, зареждането на съдържанието започва веднага, щом рендерирането на всички компоненти приключи.

Това е действието, което се използва по подразбиране.

TS component.ts

```
@defer (on immediate) {
    <app-team></app-team>
} @placeholder {
    <div>Calendar placeholder</div>
}
```

On timer

Когато използваме **timer**, зареждането на съдържанието започва веднага, щом приключи периодът на чакане.

TS component.ts

```
@defer (on timer(500ms)) {
    <app-team></app-team>
}
```

When

Позволява ни сами да определим кога ще бъде започнато свалянето на съдържанието.

TS component.ts

```
@defer (when true) {
    <app-team></app-team>
}
```

Използване заедно

Angular ни позволява да използваме едновременно няколко начина за задействане на свалянето.

Ако едно от условията върне положителен резултат, свалянето започва.

TS component.ts

```
@defer (on viewport; on timer(5s)) {
    <app-team></app-team>
} @placeholder {
    <div>Placeholder</div>
}
```

Prefetching

Освен да зареждаме съдържание на база на някакви действия, **@defer** ни позволява също така да свалим това съдържание предварително.

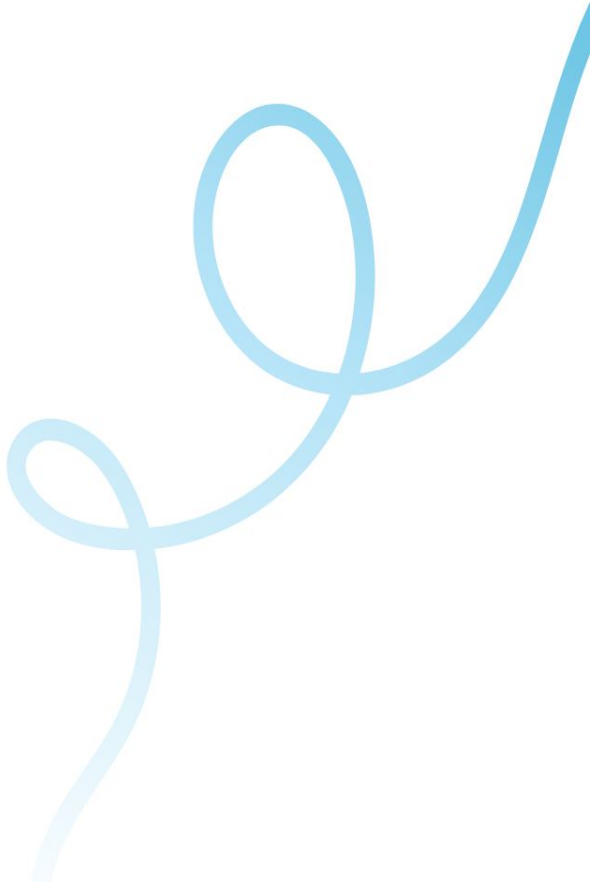
Prefetch ни позволява да използваме и два начина за задействане на **@defer**:

- on
- when

TS component.ts

```
@defer (on interaction; prefetch on idle) {
    <app-team></app-team>
} @placeholder {
    <div>Placeholder</div>
}
```

Упражнение



Съвети

- Lazy-loading е най-ефективен когато имаме качествен caching.
- Ако нещо не е от съществено значение и може да бъде lazy-loaded, значи то трябва да бъде.
- Preload/prefetch съдържание, което е много вероятно да бъде използвано.
- Следете размера на приложението и пакетите, които използвате.

Благодаря за вниманието!