ANGULAR

UNIVERSAL

in produzione

Michele Stieven

Consulente e Sviluppatore Front-End

www.accademia.dev

Accademia DEV
WEB DEVELOPER ACADEMY

Topic del talk

- SSR (server-side rendering)
- Pre-rendering
- Problematiche di Universal
- Possibili workaround

La vostra app:

```
<app-root></app-root>
```

Benefici di Universal

- SEO (ottimizzazione per motori di ricerca e social share)
- FCP (first contentful paint)
- Migliori performance iniziali

Default

Lifecycle

L’utente visita il sito web

Il server restituisce file statici (HTML e vari bundle JS)

L’utente vede una pagina completamente vuota

Angular parte e l’applicazione prende vita

Server-side Rendering

Lifecycle

L’utente visita il sito web

Il server avvia Angular e l’app prende vita su server

Il client riceve l’HTML dell’app popolata

Il client riavvia Angular dietro le quinte e rimpiazza l’HTML

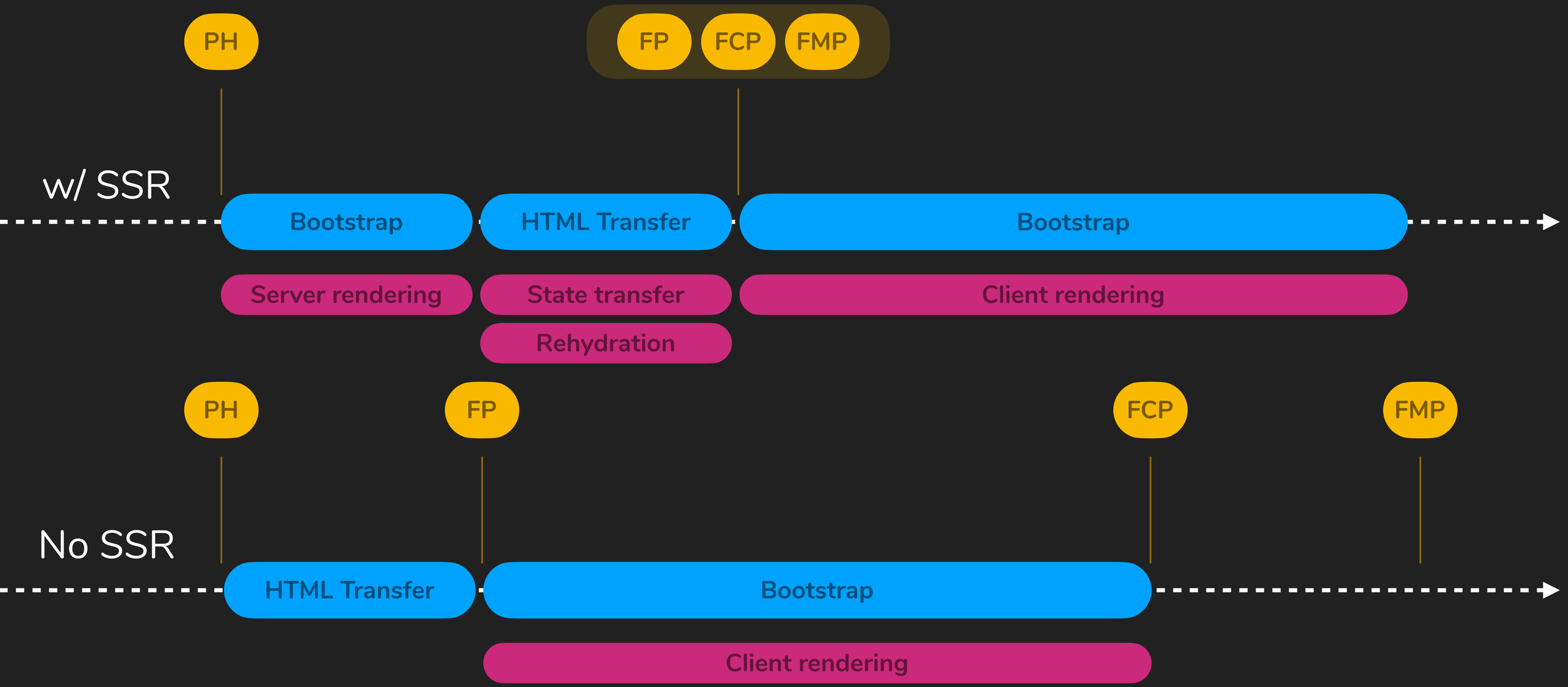
Front-End Lifecycle

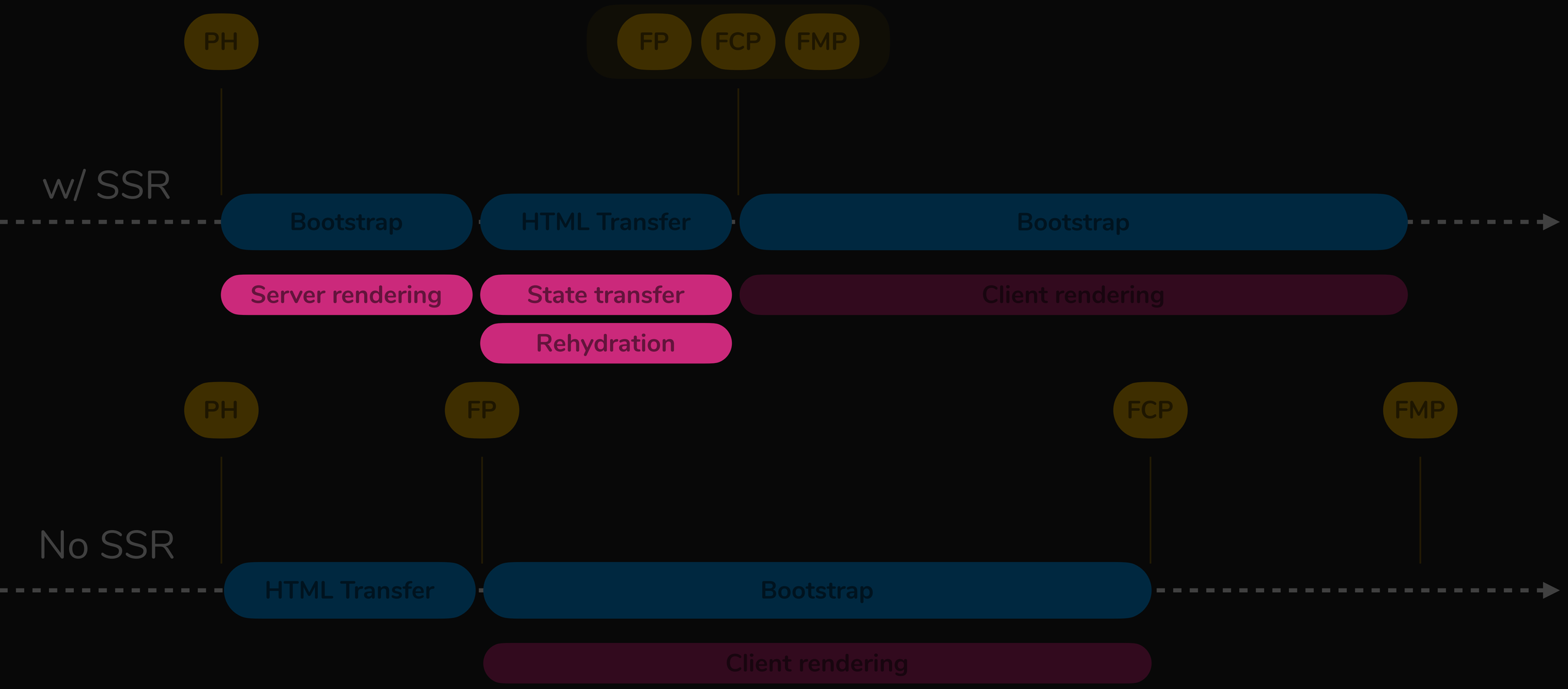
Page Hit

First Paint

First Contentful Paint

First Meaningful Paint


PH
FP
FCP
FMP
w/ SSR
Bootstrap
HTML Transfer
Bootstrap
Server rendering
State transfer
Client rendering
Rehydration
PH
FP
FCP
FMP
No SSR
HTML Transfer
Bootstrap
Client rendering


PH
FP
FCP
FMP
w/ SSR
Bootstrap
HTML Transfer
Bootstrap
Server rendering
State transfer
Client rendering
Rehydration
PH
FP
FCP
FMP
No SSR
HTML Transfer
Bootstrap
Client rendering

Setup

- ng add @nguniversal/express-engine
- ng update @nguniversal/express-engine

No Universal

Already Universal

Changes

- main.server.ts
- app.server.module.ts
- server.ts
- tsconfig.server.json

Gotchas

- Non modificare il DOM direttamente
- Attenzione a librerie che manipolano il DOM
- Attenzione a Memory Leak che potrebbero inceppare Node
- Attenzione a chiamate HTTP lunghe
- Attenzione a trasferire uno stato troppo grande
- La pagina iniziale non offre interazioni (a parte routerLink)

Non modificare il DOM direttamente

Usare invece *Renderer2* e *ElementRef*

```
constructor(el: ElementRef, renderer: Renderer2) {
    renderer.setStyle(el.nativeElement, ‘color’, ‘red’);
}
```

window non esiste (su server)

quindi niente *document*, *navigator*, *localStorage*...

```
import { PLATFORM_ID } from ‘@angular/core’;
import { isPlatformBrowser, isPlatformServer } from ‘@angular/common’;

constructor(@Inject(PLATFORM_ID) private platformId: string) { }

ngOnInit() {
    if (isPlatformBrowser(this.platformId)) {
        localStorage.set(...);
    }
}
```

L’app passa di qui **2 volte**

```
<div *isBrowser>
    <my-graph></my-graph>
</div>

<div *isServer>
    ...Loading...
</div>
```

```
@Directive({ selector: [isBrowser] })
export class IsBrowserDirective implements OnInit {

  constructor(
    @Inject(PLATFORM_ID) private platformId: string,
    private templateRef: TemplateRef<any>,
    private viewContainer: ViewContainerRef
  ) {}

  ngOnInit() {
    if (isPlatformBrowser(this.platformId)) {
      this.viewContainer.createEmbeddedView(this.templateRef);
    } else {
      this.viewContainer.clear();
    }
  }
```

Gotchas

- Non modificare il DOM direttamente
- Attenzione a librerie che manipolano il DOM
- Attenzione a Memory Leak che potrebbero inceppare Node
- Attenzione a chiamate HTTP lunghe
- Attenzione a trasferire uno stato troppo grande
- La pagina iniziale non offre interazioni (a parte routerLink)

DOMINO

Server-side DOM implementation

```
const domino = require('domino');
const fs = require('fs');
const path = require('path');
const template = fs.readFileSync(path.join(__dirname, '..', 'browser', 'index.html')).toString();
const window = domino.createWindow(template);
global['window'] = window;
global['document'] = window.document;
```

Gotchas

- Non modificare il DOM direttamente
- Attenzione a librerie che manipolano il DOM
- Attenzione a Memory Leak che potrebbero inceppare Node
- Attenzione a chiamate HTTP lunghe
- Attenzione a trasferire uno stato troppo grande
- La pagina iniziale non offre interazioni (a parte routerLink)

Trasferimento di Stato

Da server a client

```
import { makeStateKey, TransferState } from '@angular/platform-browser';

constructor(private transferState: TransferState) {}

// On server...
const COUNTER = makeStateKey<number>('counter');
this.transferState.set(COUNTER, 10);

// On client...
const counter = this.transferState.get(COUNTER, null);
```

Chiamate HTTP ripetute

Utilizzare *TransferHttpCacheModule*
e *ServerTransferStateModule*

BROWSER

```
@NgModule({
  declarations: [ AppComponent ],
  imports: [
    BrowserModule.withServerTransition({ appId: 'serverApp' }),
    TransferHttpCacheModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

SERVER

```
@NgModule({
  imports: [
    AppModule,
    ServerModule,
    ServerTransferStateModule
  ],
  bootstrap: [AppComponent],
})
export class AppServerModule {}
```

Gotchas

- Non modificare il DOM direttamente
- Attenzione a librerie che manipolano il DOM
- Attenzione a Memory Leak che potrebbero inceppare Node
- Attenzione a chiamate HTTP lunghe
- Attenzione a trasferire uno stato troppo grande
- La pagina iniziale non offre interazioni (a parte routerLink)

Rehydration

- Quando l’app passa sul client, l’HTML viene *riscritto e sostituito*
- Può causare un leggero flicker iniziale…
- …soprattutto se lo stato da leggere è molto grande!
- L’app inizialmente non supporta interazioni (a parte *routerLink*)
- *Preboot* può agevolare la transizione fra server e client

Preboot

Da server a client

```
@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule.withServerTransition({ appId: ‘serverApp’ }),
    PrebootModule.withConfig({ appRoot: 'app-root' })
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Authentication

- **Session Cookies - Fattibile**
  - Il server Node deve passare l'oggetto Request ad Angular...
  - ...che inoltrerà i Cookie all'API (es. interceptor)
- **Stateless Authentication - Non fattibile**
  - Il server Node non ha accesso a localStorage / sessionStorage / etc...
  - ...quindi non può recuperare l'Access Token

Soluzioni

- Usare i Cookie
- Usare una Guard per evitare di renderizzare su server una pagina protetta (verrà renderizzata su client)
- Renderizzare su server solo le pagine pubbliche (tramite Express)

AuthGuard - Universal

```
@Injectable()
export class AuthGuard implements CanActivate {

    constructor(@Inject(PLATFORM_ID) private platformId: string) {}

    canActivate() {
        if (isPlatformServer(this.platformId)) return false;
        ...
    }
}
```

```
// (BEFORE) Non-Universal Routes
server.get(CLIENT_ROUTES, (req, res) => {
    res.sendFile(join(distFolder, 'index.html'));
});

// (AFTER) Universal Routes
server.get('*', (req, res) => {
  res.render(indexHtml, { req, providers: [...] });
});
```

Pre-rendering

- Angular viene avviato in fase di sviluppo (come **ng serve**)
- Vengono rintracciate le rotte statiche (o dinamiche scelte dal team)
- Vengono salvati i file HTML per quelle rotte
- Abbiamo dei file statici già pronti da consegnare all’utente
- Angular si avvia su client in un secondo momento
- **npm run prerender**

Pre-rendering Gotchas

- Le rotte dinamiche vanno specificate manualmente
  - **angular.json**
  - - - **routes** ‘/detail/1’ - - **routes** ‘detail/2’
  - - - **routesFile** routes.txt
- Ad ogni cambiamento di contenuti bisogna rifare la build
- Anche qui, piccolo flicker iniziale quando Angular viene avviato

	Classic	SSR	Prerender
SEO	✗	✓	✓
Initial Performance	✗	✓	✓✓
App con autenticazione	✓	✓✗	✓✗
Contenuti dinamici	✓	✓	✓✗

Michele Stieven

www.accademia.dev

Accademia DEV
WEB DEVELOPER ACADEMY

Michele Stieven

github.com/UserGalileo/talks

www.accademia.dev

Accademia DEV
WEB DEVELOPER ACADEMY