INFINUM

Server–side rendering in Angular

FILIP VOSKA

01

GETTING STARTED

PROJECT SETUP

- Follow the official guide: **https://angular.io/guide/universal**
- SSR can easily be added to any existing angular-cli project:

PROJECT SETUP

- Follow the official guide: **https://angular.io/guide/universal**
- SSR can easily be added to any existing angular-cli project:

```
ng add @nguniversal/express-engine --clientProject my-app
```

```
src/
  index.html                  * app web page
  main.ts                     * bootstrapper for client app
  main.server.ts              * bootstrapper for server app
  tsconfig.app.json           * TypeScript client configuration
  tsconfig.server.json        * TypeScript server configuration
  tsconfig.spec.json          * TypeScript spec configuration
  style.css                   * styles for the app
  app/ ...                    * application code
    app.server.module.ts      * server-side application module
server.ts                     * express web server
tsconfig.json                 * TypeScript client configuration
package.json                  * npm configuration
webpack.server.config.js      * webpack server configuration
```

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

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

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

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

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

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

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

```json
{
  "extends": "./tsconfig.app.json",
  "compilerOptions": {
    "outDir": "../out-tsc/app-server",
    "baseUrl": "."
  },
  "angularCompilerOptions": {
    "entryModule": "app/app.server.module#AppServerModule"
  }
}
```

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

```json
{
  "extends": "./tsconfig.app.json",
  "compilerOptions": {
    "outDir": "../out-tsc/app-server",
    "baseUrl": "."
  },
  "angularCompilerOptions": {
    "entryModule": "app/app.server.module#AppServerModule"
  }
}
```

```
const app = express();

const {
  AppServerModuleNgFactory,
  LAZY_MODULE_MAP
} = require('./dist/server/main');

app.engine('html', ngExpressEngine({
  bootstrap: AppServerModuleNgFactory,
  providers: [
    provideModuleMap(LAZY_MODULE_MAP)
  ]
}));

app.set('view engine', 'html');
app.set('views', DIST_FOLDER);

app.get('*', (req, res) => {
  res.render('index', { req });
});
```

```
const app = express();

const {
  AppServerModuleNgFactory,
  LAZY_MODULE_MAP
} = require('./dist/server/main');

app.engine('html', ngExpressEngine({
  bootstrap: AppServerModuleNgFactory,
  providers: [
    provideModuleMap(LAZY_MODULE_MAP)
  ]
}));

app.set('view engine', 'html');
app.set('views', DIST_FOLDER);

```

```
  AppServerModuleNgFactory,
  LAZY_MODULE_MAP
} = require('./dist/server/main');

app.engine('html', ngExpressEngine({
  bootstrap: AppServerModuleNgFactory,
  providers: [
    provideModuleMap(LAZY_MODULE_MAP)
  ]
}));

app.set('view engine', 'html');
app.set('views', DIST_FOLDER);

app.get('*', (req, res) => {
  res.render('index', { req });
});
```

```
app.engine('html', ngExpressEngine({
  bootstrap: AppServerModuleNgFactory,
  providers: [
    provideModuleMap(LAZY_MODULE_MAP)
  ]
}));

app.set('view engine', 'html');
app.set('views', DIST_FOLDER);

app.get('*', (req, res) => {
  res.render('index', { req });
});
```

```
app.get('*', (req, res) => {
  res.render('index', { req });
});
```

```
app.get('*', (req, res) => {
  res.render('index', { req });
});
```

```
app.get('*', (req: Request, res: Response) => {
  res.render('index', { req, res }, (error: Error, html: string) => {
    if (error) {
      console.error(Date.now(), error);
      res.status(500).send(error);
    } else if (!res.headersSent && !res.finished) {
      res.send(html);
    }
  });
});
```

```
app.get('*', (req, res) => {
  res.render('index', { req });
});
```

```
app.get('*', (req: Request, res: Response) => {
  res.render('index', { req, res }, (error: Error, html: string) => {
    if (error) {
      console.error(Date.now(), error);
      res.status(500).send(error);
    } else if (!res.headersSent && !res.finished) {
      res.send(html);
    }
  });
});
```

EXPRESS ISN'T THE ONLY WAY

- @nguniversal/aspnetcore-engine
- @nguniversal/hapi-engine
- @nguniversal/socket-engine

02

SERVER-SIDE RENDERING CHALLENGES

DETECT WHERE THE CODE IS RUNNING 🧭

DETECT WHERE THE CODE IS RUNNING 🧭

```
1  import { isPlatformBrowser, isPlatformServer } from '@angular/common';
2  import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
3
4  @Injectable()
5  export class PlatformService {
6    constructor(@Inject(PLATFORM_ID) private platformId: object) { }
7
8    public get isBrowser(): boolean {
9      return isPlatformBrowser(this.platformId);
10   }
11
12   public get isServer(): boolean {
13     return isPlatformServer(this.platformId);
14   }
15 }
```

DETECT WHERE THE CODE IS RUNNING 🧭

```
1  import { isPlatformBrowser, isPlatformServer } from '@angular/common';
2  import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
3
4  @Injectable()
5  export class PlatformService {
6    constructor(@Inject(PLATFORM_ID) private platformId: object) { }
7
8    public get isBrowser(): boolean {
9      return isPlatformBrowser(this.platformId);
10   }
11
12   public get isServer(): boolean {
13     return isPlatformServer(this.platformId);
14   }
15 }
```

DETECT WHERE THE CODE IS RUNNING 🧭

```
1  import { isPlatformBrowser, isPlatformServer } from '@angular/common';
2  import { Inject, Injectable, PLATFORM_ID } from '@angular/core';
3
4  @Injectable()
5  export class PlatformService {
6    constructor(@Inject(PLATFORM_ID) private platformId: object) { }
7
8    public get isBrowser(): boolean {
9      return isPlatformBrowser(this.platformId);
10   }
11
12   public get isServer(): boolean {
13     return isPlatformServer(this.platformId);
14   }
15 }
```

DETECT WHERE THE CODE IS RUNNING 🧭

```
import { isPlatformBrowser, isPlatformServer } from '@angular/common';
import { Inject, Injectable, PLATFORM_ID } from '@angular/core';

@Injectable()
export class PlatformService {
  constructor(@Inject(PLATFORM_ID) private platformId: object) { }

  public get isBrowser(): boolean {
    return isPlatformBrowser(this.platformId);
  }

  public get isServer(): boolean {
    return isPlatformServer(this.platformId);
  }
}
```

SETTING CORRECT RESPONSE STATUS CODE 🆗

SETTING CORRECT RESPONSE STATUS CODE 🆗

SETTING CORRECT RESPONSE STATUS CODE 🆗

SETTING CORRECT RESPONSE STATUS CODE 🆗

SETTING CORRECT RESPONSE STATUS CODE 🆗

```ts
import { RESPONSE } from '@nguniversal/express-engine/tokens';
import { Response } from 'express';
...

@Injectable()
export class ResponseStatusService {
  constructor(
    private platform: PlatformService,
    @Optional() @Inject(RESPONSE) private response: Response,
  ) { }

  public setResponseStatus(status: HttpStatusCode): void {
    if (this.platform.isServer) {
      this.response.status(status);
    }
  }
}
```

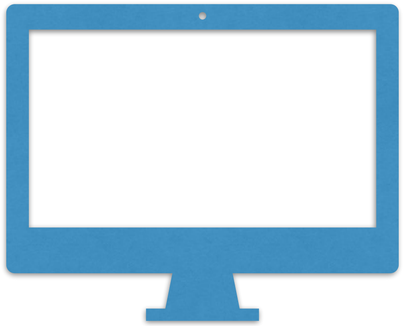
SETTING CORRECT RESPONSE STATUS CODE 🆗

```
import { RESPONSE } from '@nguniversal/express-engine/tokens';
import { Response } from 'express';
...

@Injectable()
export class ResponseStatusService {
  constructor(
    private platform: PlatformService,
    @Optional() @Inject(RESPONSE) private response: Response,
  ) { }

  public setResponseStatus(status: HttpStatusCode): void {
    if (this.platform.isServer) {
      this.response.status(status);
    }
  }
}
```

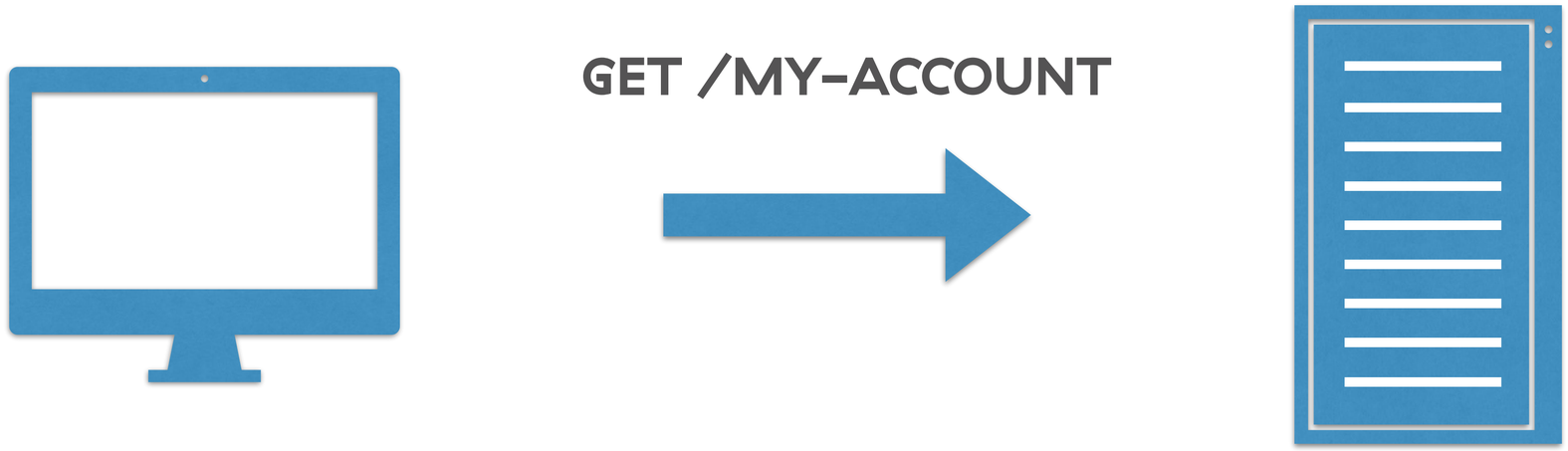
SETTING CORRECT RESPONSE STATUS CODE 🆗

```
import { RESPONSE } from '@nguniversal/express-engine/tokens';
import { Response } from 'express';
...

@Injectable()
export class ResponseStatusService {
  constructor(
    private platform: PlatformService,
    @Optional() @Inject(RESPONSE) private response: Response,
  ) { }

  public setResponseStatus(status: HttpStatusCode): void {
    if (this.platform.isServer) {
      this.response.status(status);
    }
  }
}
```

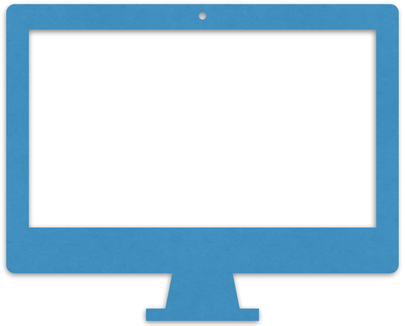
AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫



AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫



AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫

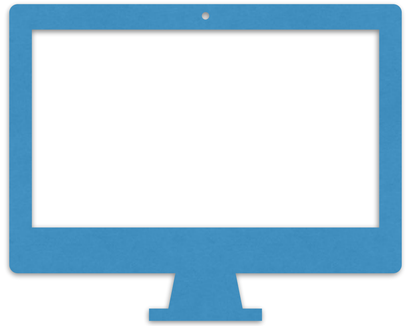


GET /MY-ACCOUNT

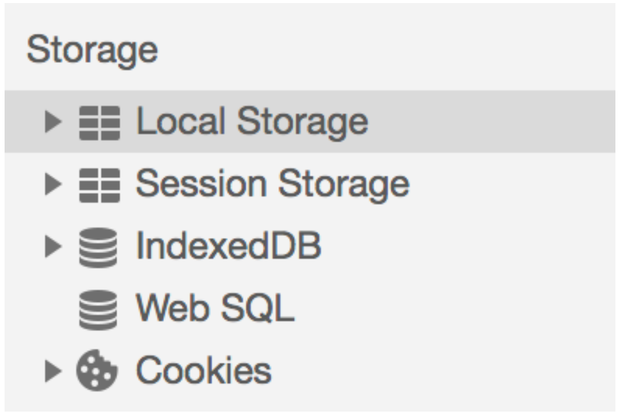
WHO ARE YOU?

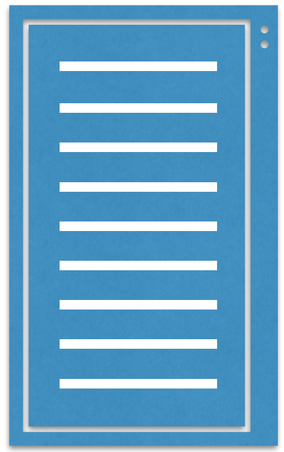
(WHO, WHO? WHO, WHO?)

AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫

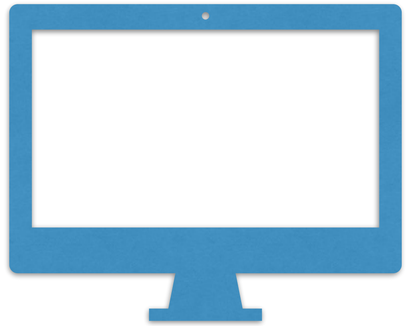


GET /MY-ACCOUNT

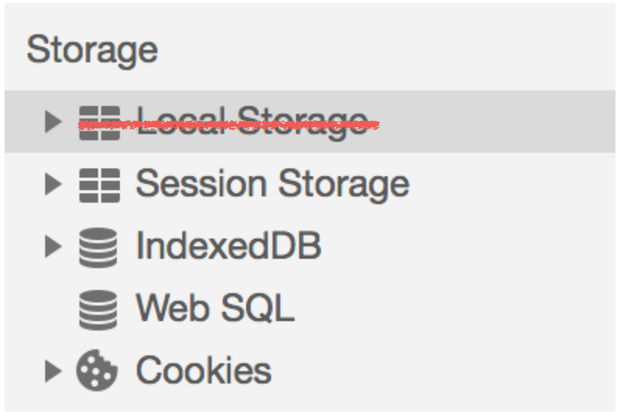


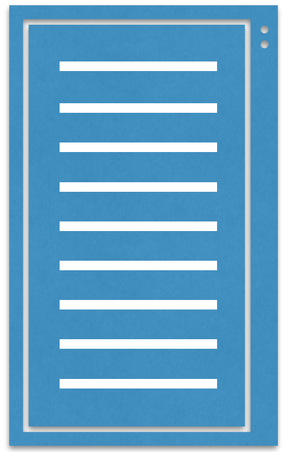


AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫

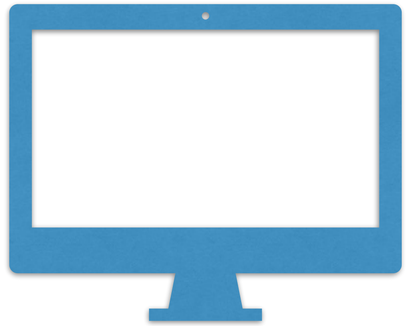


GET /MY-ACCOUNT

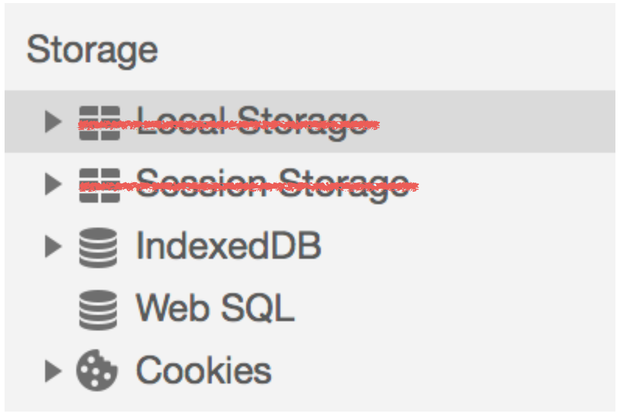


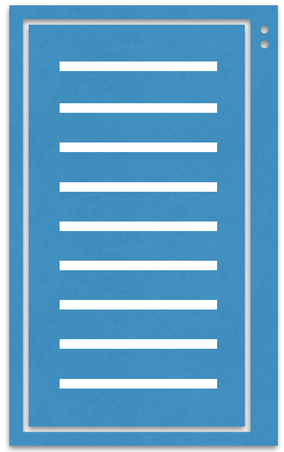


AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫

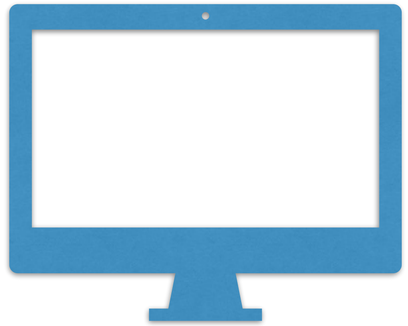


GET /MY-ACCOUNT

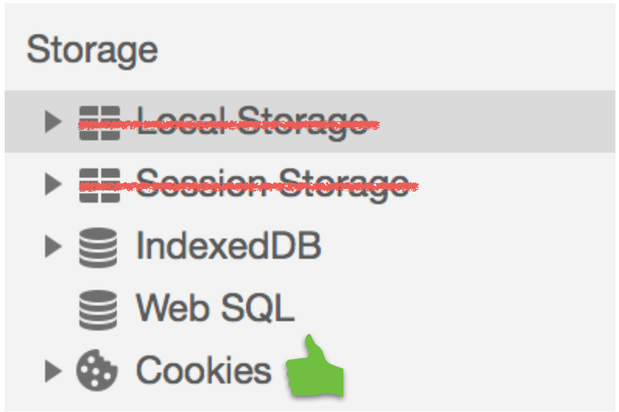


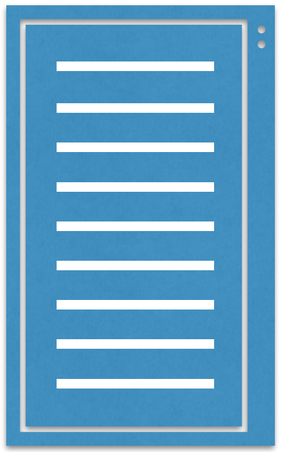


AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫

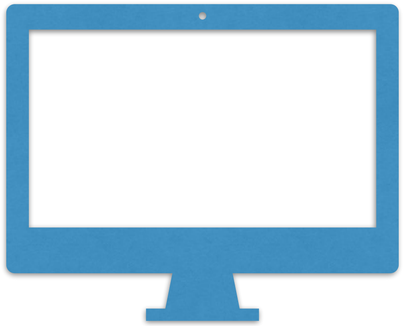


GET /MY-ACCOUNT





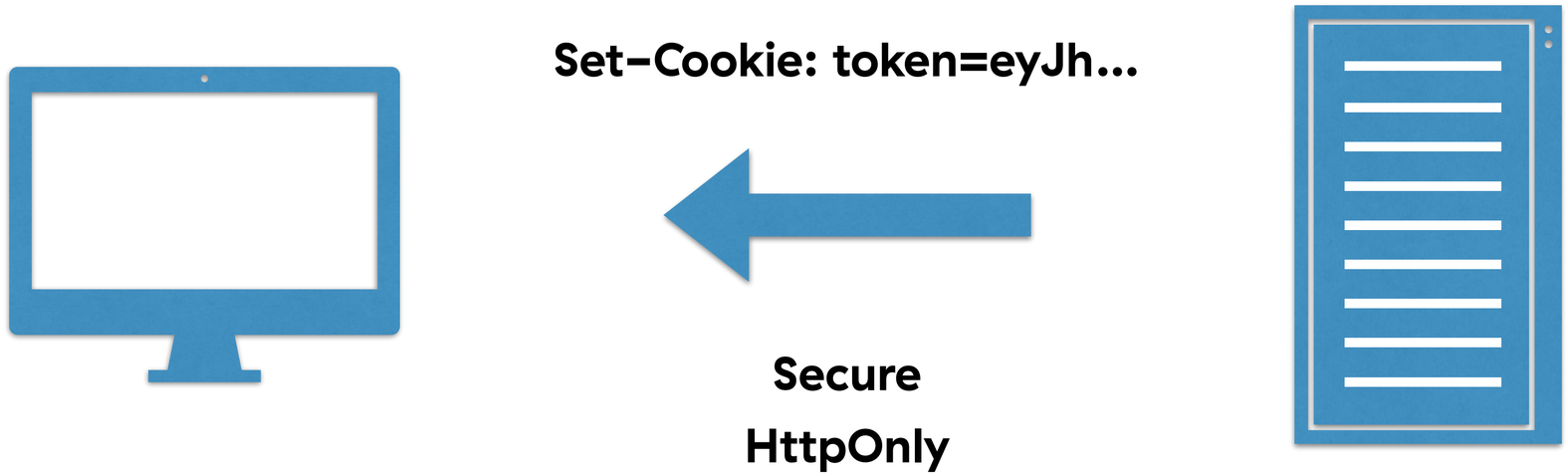
AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫



AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫

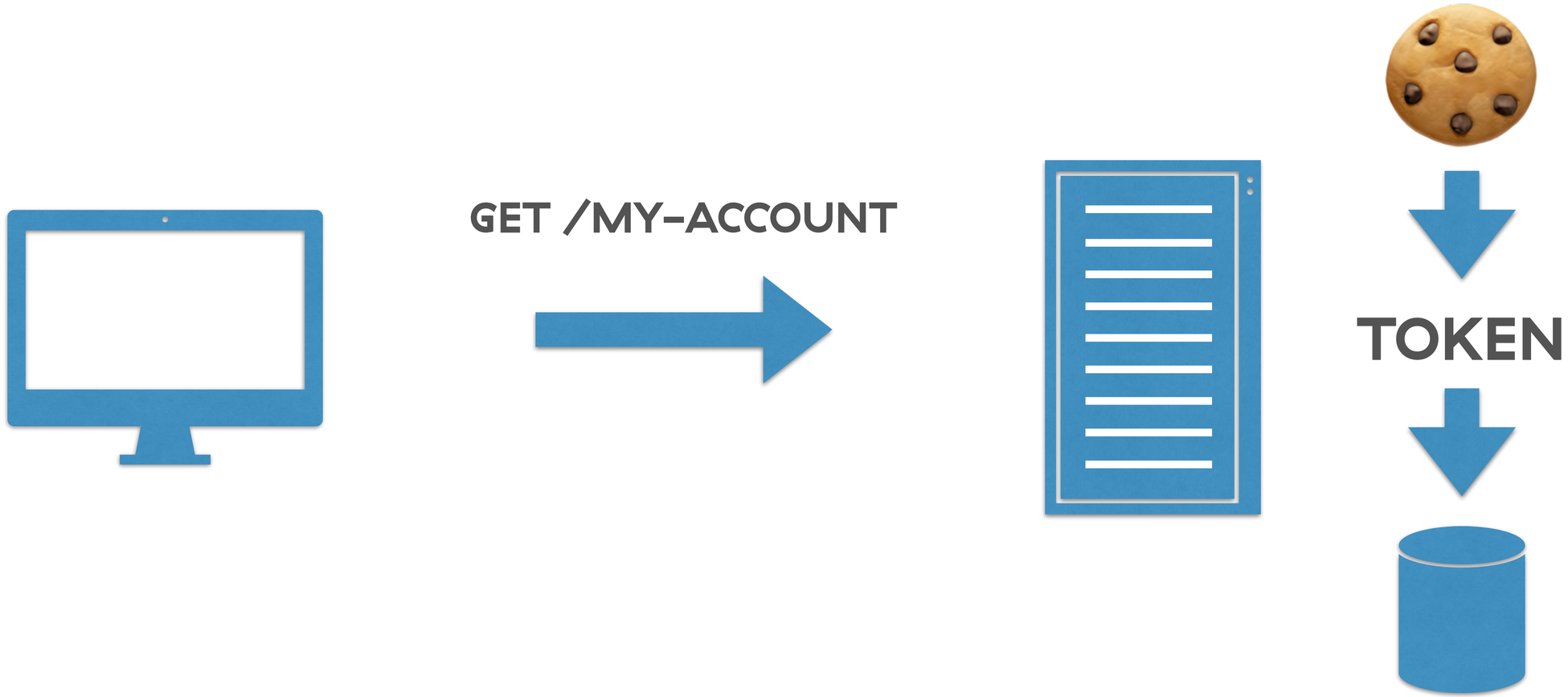
AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫

AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫



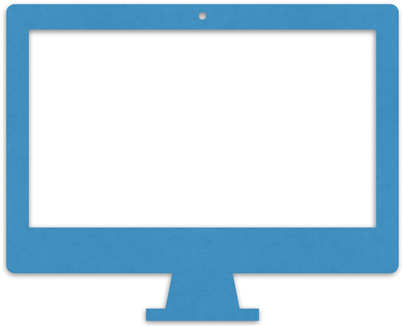
AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫

AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫



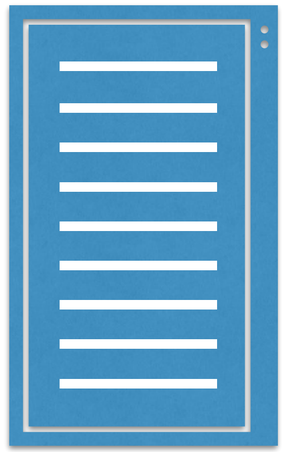
AUTHENTICATION 🔑 & HANDLING SECURE ROUTES 🚫

REDIRECTS ↩

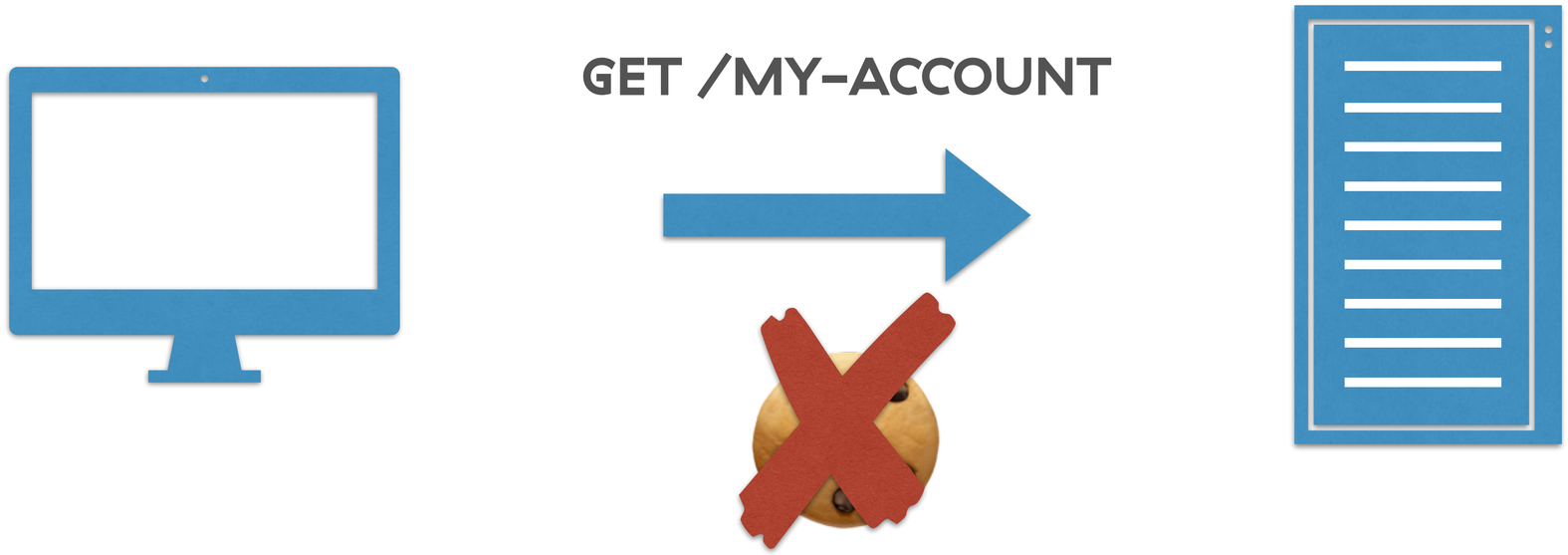


REDIRECTS ↩

GET /MY-ACCOUNT

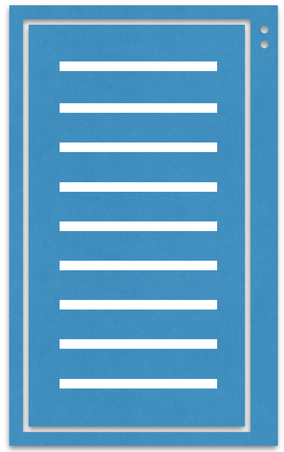


REDIRECTS ↩

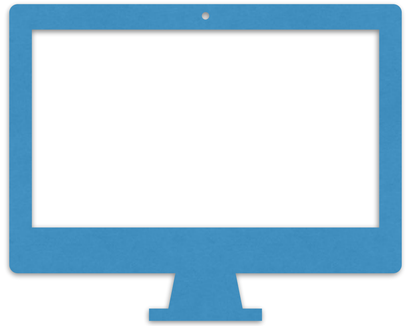


REDIRECTS ↩

GET /MY-ACCOUNT



REDIRECTS ↩

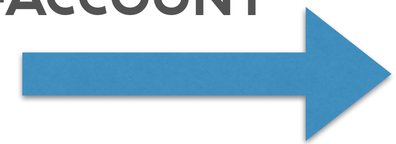


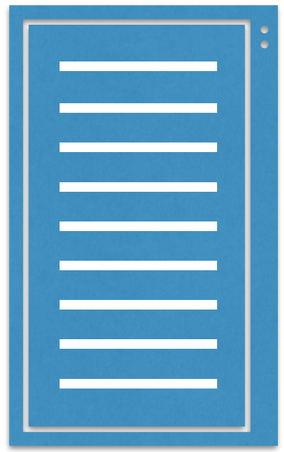
302 REDIRECT

LOCATION /ACCESS-DENIED
?REDIRECTURL
=/MY-ACCOUNT

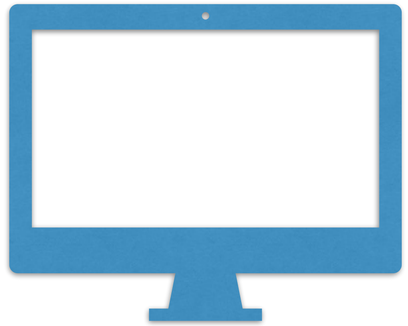
REDIRECTS ↩

GET /ACCESS-DENIED
?REDIRECTURL
=/MY-ACCOUNT



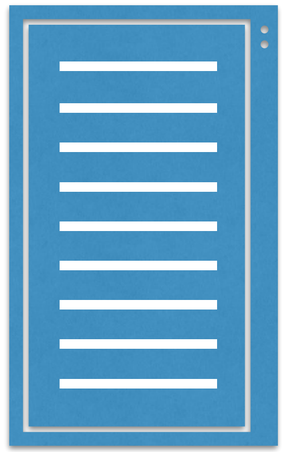


REDIRECTS ↩



RENDERED
ACCESS-DENIED
PAGE

401/403



REDIRECTS ↩ & HANDLING SECURE ROUTES 🚫

```ts
import { REQUEST, RESPONSE } from '@nguniversal/express-engine/tokens';
import { Request, Response } from 'express';

@Injectable()
export class AuthenticationGuard implements CanActivate {
  constructor(
    private router: Router,
    @Optional() @Inject(REQUEST) private request: Request,
    @Optional() @Inject(RESPONSE) private response: Response,
  ) { }

  public canActivate(
    _route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean | UrlTree> {
    const authorized = ...;

    if (authorized) { return true; }

    if (this.platform.isBrowser) {
      const redirectTree = this.router.createUrlTree(['/access-denied'], {
        queryParams: {
          redirectUrl,
        },
      });

      return of(redirectTree);
    } else {
      const redirectHost = this.request.headers['host'];
      this.response.redirect(`https://${redirectHost}/access-denied?redirectUrl=${redirectUrl}`);
      this.response.end();
      this.response.finished = true;

      return of(false);
    }
  }
}
```

```
import { REQUEST, RESPONSE } from '@nguniversal/express-engine/tokens';
import { Request, Response } from 'express';

@Injectable()
export class AuthenticationGuard implements CanActivate {
  constructor(
    private router: Router,
    @Optional() @Inject(REQUEST) private request: Request,
    @Optional() @Inject(RESPONSE) private response: Response,
  ) { }

  public canActivate(
    _route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean | UrlTree> {
    const authorized =  ... ;

    if (authorized) { return true; }

    if (this.platform.isBrowser) {
      const redirectTree = this.router.createUrlTree(['/access-denied'], {
        queryParams: {
```

```
import { REQUEST, RESPONSE } from '@nguniversal/express-engine/tokens';
import { Request, Response } from 'express';

@Injectable()
export class AuthenticationGuard implements CanActivate {
  constructor(
    private router: Router,
    @Optional() @Inject(REQUEST) private request: Request,
    @Optional() @Inject(RESPONSE) private response: Response,
  ) { }

  public canActivate(
    _route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean | UrlTree> {
    const authorized =  ... ;

    if (authorized) { return true; }

    if (this.platform.isBrowser) {
      const redirectTree = this.router.createUrlTree(['/access-denied'], {
        queryParams: {
```

```
@Injectable()
export class AuthenticationGuard implements CanActivate {
  constructor(
    private router: Router,
    @Optional() @Inject(REQUEST) private request: Request,
    @Optional() @Inject(RESPONSE) private response: Response,
  ) { }

  public canActivate(
    _route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean | UrlTree> {
    const authorized =  ... ;

    if (authorized) { return true; }

    if (this.platform.isBrowser) {
      const redirectTree = this.router.createUrlTree(['/access-denied'], {
        queryParams: {
          redirectUrl,
        },
      });

      return of(redirectTree);
    } else {
```

```
11
12  public canActivate(
13    _route: ActivatedRouteSnapshot,
14    state: RouterStateSnapshot
15  ): Observable<boolean | UrlTree> {
16    const authorized =  ... ;
17
18    if (authorized) { return true; }
19
20    if (this.platform.isBrowser) {
21      const redirectTree = this.router.createUrlTree(['/access-denied'], {
22        queryParams: {
23          redirectUrl,
24        },
25      });
26
27      return of(redirectTree);
28    } else {
29      const redirectHost = this.request.headers['host'];
30      this.response.redirect(`https://${redirectHost}/access-denied?redirectUrl=${redirectUrl}`);
31      this.response.end();
32      this.response.finished = true;
33
34      return of(false);
35    }
36  }
```

```
    if (authorized) { return true; }

    if (this.platform.isBrowser) {
      const redirectTree = this.router.createUrlTree(['/access-denied'], {
        queryParams: {
          redirectUrl,
        },
      });

      return of(redirectTree);
    } else {
      const redirectHost = this.request.headers['host'];
      this.response.redirect(`https://${redirectHost}/access-denied?redirectUrl=${redirectUrl}`);
      this.response.end();
      this.response.finished = true;

      return of(false);
    }
  }
}
```

STATE TRANSFER (REHYDRATION) 🌊

```
1 import { makeStateKey, StateKey, TransferState } from '@angular/platform-browser';
2 ...
3
4 @Injectable()
5 export class StateService implements IStateService {
6   constructor(private transferState: TransferState) { }
7
8   public makeKey<T>(id: string): StateKey<T> {
9     return makeStateKey(id);
10  }
11
12  public get<T>(stateKey: StateKey<T>, defaultValue: T = null): T {
13    return this.transferState.get<T>(stateKey, defaultValue);
14  }
15
16  public set<T>(stateKey: StateKey<T>, value: T): void {
17    this.transferState.set<T>(stateKey, value);
18  }
19
20  public remove<T>(stateKey: StateKey<T>): void {
21    this.transferState.remove<T>(stateKey);
22  }
23 }
24
```

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

@Injectable()
export class StateService implements IStateService {
  constructor(private transferState: TransferState) { }

  public makeKey<T>(id: string): StateKey<T> {
    return makeStateKey(id);
  }

  public get<T>(stateKey: StateKey<T>, defaultValue: T = null): T {
    return this.transferState.get<T>(stateKey, defaultValue);
  }

  public set<T>(stateKey: StateKey<T>, value: T): void {
```

```
1  import { makeStateKey, StateKey, TransferState } from '@angular/platform-browser';
2  ...
3
4  @Injectable()
5  export class StateService implements IStateService {
6    constructor(private transferState: TransferState) { }
7
8    public makeKey<T>(id: string): StateKey<T> {
9      return makeStateKey(id);
10   }
11
12   public get<T>(stateKey: StateKey<T>, defaultValue: T = null): T {
13     return this.transferState.get<T>(stateKey, defaultValue);
14   }
15
16   public set<T>(stateKey: StateKey<T>, value: T): void {
```

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

@Injectable()
export class StateService implements IStateService {
  constructor(private transferState: TransferState) { }

  public makeKey<T>(id: string): StateKey<T> {
    return makeStateKey(id);
  }

  public get<T>(stateKey: StateKey<T>, defaultValue: T = null): T {
    return this.transferState.get<T>(stateKey, defaultValue);
  }

  public set<T>(stateKey: StateKey<T>, value: T): void {
    this.transferState.set<T>(stateKey, value);
  }

  public remove<T>(stateKey: StateKey<T>): void {
```

```
@Injectable()
export class StateService implements IStateService {
  constructor(private transferState: TransferState) { }

  public makeKey<T>(id: string): StateKey<T> {
    return makeStateKey(id);
  }

  public get<T>(stateKey: StateKey<T>, defaultValue: T = null): T {
    return this.transferState.get<T>(stateKey, defaultValue);
  }

  public set<T>(stateKey: StateKey<T>, value: T): void {
    this.transferState.set<T>(stateKey, value);
  }

  public remove<T>(stateKey: StateKey<T>): void {
    this.transferState.remove<T>(stateKey);
  }
}

```

```
const homepageKey = state.makeKey('homepageData');
state.set(homepageKey, {
  title: 'Welcome',
  text: 'Angular Universal Demo',
  time: Date.now()
})
```

```
const homepageKey = state.makeKey('homepageData');
state.set(homepageKey, {
  title: 'Welcome',
  text: 'Angular Universal Demo',
  time: Date.now()
})
```

```
<body>
  <!-- ... rendered page ... -->

  <script id="universal-demo-state" type="application/json">
    { &q;homepageData&q;:{&q;title&q;:&q;Welcome&q;,&q;text&q;:&q;Angular
          Universal Demo & q;,
        & q;time & q;: 1531076035630
      }
    }
  </script>
</body>
```

```
const homepageKey = state.makeKey('homepageData');
state.set(homepageKey, {
  title: 'Welcome',
  text: 'Angular Universal Demo',
  time: Date.now()
})
```

```
<body>
  <!-- ... rendered page ... -->

  <script id="universal-demo-state" type="application/json">
    { &q;homepageData&q;:{&q;title&q;:&q;Welcome&q;,&q;text&q;:&q;Angular
          Universal Demo & q;,
        & q;time & q;: 1531076035630
      }
    }
  </script>
</body>
```

AVOIDING DUPLICATE REQUESTS ♊

```
@Injectable()
export class PageFetchingService {
  constructor(
    private platform: PlatformService,
    private state: StateService,
    private http: HttpClient,
    private responseStatus: ResponseStatusService,
  ) { }

  public getPage(slug: string): Observable<Page> {
    const pageStateKey = this.stateService.makeKey<IPage>(slug);
    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);

    if (this.platform.isBrowser && pageDataFromSSR) {
      this.stateService.remove<IPage>(pageStateKey);

      return of(new Page(pageDataFromSSR));
    }

    return this.http.get(path).pipe(
      catchError((error: HttpErrorResponse) => {
        this.responseStatusService.setResponseStatus(error.status);
      })
      tap((pageData: IPage) => {
        if (this.platform.isServer) {
          this.stateService.set<IPage>(pageStateKey, pageData);
        }
      }),
      map((pageData: IPage) => {
        return new Page(pageData);
      }),
    );
  }
```

```
@Injectable()
export class PageFetchingService {
  constructor(
    private platform: PlatformService,
    private state: StateService,
    private http: HttpClient,
    private responseStatus: ResponseStatusService,
  ) { }

  public getPage(slug: string): Observable<Page> {
    const pageStateKey = this.stateService.makeKey<IPage>(slug);
    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);

    if (this.platform.isBrowser && pageDataFromSSR) {
```

```
@Injectable()
export class PageFetchingService {
  constructor(
    private platform: PlatformService,
    private state: StateService,
    private http: HttpClient,
    private responseStatus: ResponseStatusService,
  ) { }

  public getPage(slug: string): Observable<Page> {
    const pageStateKey = this.stateService.makeKey<IPage>(slug);
    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);

    if (this.platform.isBrowser && pageDataFromSSR) {
```

ON SERVER

```
4     private platform: PlatformService,
5     private state: StateService,
6     private http: HttpClient,
7     private responseStatus: ResponseStatusService,
8   ) { }
9
10  public getPage(slug: string): Observable<Page> {
11    const pageStateKey = this.stateService.makeKey<IPage>(slug);
12    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);
13
14    if (this.platform.isBrowser && pageDataFromSSR) {
15      this.stateService.remove<IPage>(pageStateKey);
16
17      return of(new Page(pageDataFromSSR));
18    }
19
20    return this.http.get(path).pipe(
21      catchError((error: HttpErrorResponse) => {
22        this.responseStatusService.setResponseStatus(error.status);
23      })
```

ON SERVER

```
    private platform: PlatformService,
    private state: StateService,
    private http: HttpClient,
    private responseStatus: ResponseStatusService,
  ) { }

  public getPage(slug: string): Observable<Page> {
    const pageStateKey = this.stateService.makeKey<IPage>(slug);
    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);

    if (this.platform.isBrowser && pageDataFromSSR) {
      this.stateService.remove<IPage>(pageStateKey);

      return of(new Page(pageDataFromSSR));
    }

    return this.http.get(path).pipe(
      catchError((error: HttpErrorResponse) => {
        this.responseStatusService.setResponseStatus(error.status);
      })
```

```
12    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);
13
14    if (this.platform.isBrowser && pageDataFromSSR) {
15      this.stateService.remove<IPage>(pageStateKey);
16
17      return of(new Page(pageDataFromSSR));
18    }
19
20    return this.http.get(path).pipe(
21      catchError((error: HttpErrorResponse) => {
22        this.responseStatusService.setResponseStatus(error.status);
23      })
24      tap((pageData: IPage) => {
25        if (this.platform.isServer) {
26          this.stateService.set<IPage>(pageStateKey, pageData);
27        }
28      }),
29      map((pageData: IPage) => {
30        return new Page(pageData);
31      }),
```

ON SERVER

ON SERVER

```
14      if (this.platform.isBrowser && pageDataFromSSR) {
15        this.stateService.remove<IPage>(pageStateKey);
16
17        return of(new Page(pageDataFromSSR));
18      }
19
20      return this.http.get(path).pipe(
21        catchError((error: HttpErrorResponse) => {
22          this.responseStatusService.setResponseStatus(error.status);
23        })
24        tap((pageData: IPage) => {
25          if (this.platform.isServer) {
26            this.stateService.set<IPage>(pageStateKey, pageData);
27          }
28        }),
29        map((pageData: IPage) => {
30          return new Page(pageData);
31        }),
32      );
```

ON SERVER

```
17        return of(new Page(pageDataFromSSR));
18      }
19
20      return this.http.get(path).pipe(
21        catchError((error: HttpErrorResponse) => {
22          this.responseStatusService.setResponseStatus(error.status);
23        })
24        tap((pageData: IPage) => {
25          if (this.platform.isServer) {
26            this.stateService.set<IPage>(pageStateKey, pageData);
27          }
28        }),
29        map((pageData: IPage) => {
30          return new Page(pageData);
31        }),
32      );
33    }
```

```
21        catchError((error: HttpErrorResponse) => {
22          this.responseStatusService.setResponseStatus(error.status);
23        })
24        tap((pageData: IPage) => {
25          if (this.platform.isServer) {
26            this.stateService.set<IPage>(pageStateKey, pageData);
27          }
28        }),
29        map((pageData: IPage) => {
30          return new Page(pageData);
31        }),
32      );
33    }
```

ON SERVER

ON CLIENT

```
4     private platform: PlatformService,
5     private state: StateService,
6     private http: HttpClient,
7     private responseStatus: ResponseStatusService,
8   ) { }
9
10  public getPage(slug: string): Observable<Page> {
11    const pageStateKey = this.stateService.makeKey<IPage>(slug);
12    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);
13
14    if (this.platform.isBrowser && pageDataFromSSR) {
15      this.stateService.remove<IPage>(pageStateKey);
16
17      return of(new Page(pageDataFromSSR));
18    }
19
20    return this.http.get(path).pipe(
21      catchError((error: HttpErrorResponse) => {
22        this.responseStatusService.setResponseStatus(error.status);
23      })
```

ON CLIENT

```
8   ) { }
9
10  public getPage(slug: string): Observable<Page> {
11    const pageStateKey = this.stateService.makeKey<IPage>(slug);
12    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);
13
14    if (this.platform.isBrowser && pageDataFromSSR) {
15      this.stateService.remove<IPage>(pageStateKey);
16
17      return of(new Page(pageDataFromSSR));
18    }
19
20    return this.http.get(path).pipe(
21      catchError((error: HttpErrorResponse) => {
22        this.responseStatusService.setResponseStatus(error.status);
23      })
24      tap((pageData: IPage) => {
25        if (this.platform.isServer) {
26          this.stateService.set<IPage>(pageStateKey, pageData);
27        }
```

ON CLIENT

```
8   ) { }
9
10  public getPage(slug: string): Observable<Page> {
11    const pageStateKey = this.stateService.makeKey<IPage>(slug);
12    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);
13
14    if (this.platform.isBrowser && pageDataFromSSR) {
15      this.stateService.remove<IPage>(pageStateKey);
16
17      return of(new Page(pageDataFromSSR));
18    }
19
20    return this.http.get(path).pipe(
21      catchError((error: HttpErrorResponse) => {
22        this.responseStatusService.setResponseStatus(error.status);
23      })
24      tap((pageData: IPage) => {
25        if (this.platform.isServer) {
26          this.stateService.set<IPage>(pageStateKey, pageData);
27        }
```

ON CLIENT

```
8   ) { }
9
10  public getPage(slug: string): Observable<Page> {
11    const pageStateKey = this.stateService.makeKey<IPage>(slug);
12    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);
13
14    if (this.platform.isBrowser && pageDataFromSSR) {
15      this.stateService.remove<IPage>(pageStateKey);
16
17      return of(new Page(pageDataFromSSR));
18    }
19
20    return this.http.get(path).pipe(
21      catchError((error: HttpErrorResponse) => {
22        this.responseStatusService.setResponseStatus(error.status);
23      })
24      tap((pageData: IPage) => {
25        if (this.platform.isServer) {
26          this.stateService.set<IPage>(pageStateKey, pageData);
27        }
```

ON CLIENT

```
8   ) { }
9
10  public getPage(slug: string): Observable<Page> {
11    const pageStateKey = this.stateService.makeKey<IPage>(slug);
12    const pageDataFromSSR = this.stateService.get<IPage>(pageStateKey);
13
14    if (this.platform.isBrowser && pageDataFromSSR) {
15      this.stateService.remove<IPage>(pageStateKey);
16
17      return of(new Page(pageDataFromSSR));
18    }
19
20    return this.http.get(pat pipe(
21      catchError((error: Htt rR se) => {
22        this.responseStatusSe ResponseStatus(error.status);
23      })
24      tap((pageData: IPage)
25        if (this.platform.i er
26          this.stateServic t<IPage>(pageStateKey, pageData);
27        }
```

USING ENVIRONMENT VARIABLES 🌍

USING ENVIRONMENT VARIABLES 🌍

```
1 import { NODE_PROCESS } from './injection-tokens';
2
3 @NgModule({
4   imports: [],
5   providers: [
6     { provide: NODE_PROCESS, useValue: process },
7   ],
8   bootstrap: [AppComponent],
9 })
10 export class AppServerModule { }
```

USING ENVIRONMENT VARIABLES 🌍

```
import { NODE_PROCESS } from './injection-tokens';

@NgModule({
  imports: [],
  providers: [
    { provide: NODE_PROCESS, useValue: process },
  ],
  bootstrap: [AppComponent],
})
export class AppServerModule { }
```

```
export enum EnvironmentVariable {
  API_URL = 'API_URL',
}
```

USING ENVIRONMENT VARIABLES 🌍

```
import { NODE_PROCESS } from './injection-tokens';

@NgModule({
  imports: [],
  providers: [
    { provide: NODE_PROCESS, useValue: process },
  ],
  bootstrap: [AppComponent],
})
export class AppServerModule { }
```

```
export enum EnvironmentVariable {
  API_URL = 'API_URL',
}
```

```
import { InjectionToken } from '@angular/core';

export const NODE_PROCESS = new InjectionToken('NODE_PROCESS');
```

USING ENVIRONMENT VARIABLES 🌍

```
@Injectable()
export class EnvironmentVariablesService {
  private transferKey: StateKey<IDictionary<string>> = this.state.makeKey('env');
  private environmentVariables: IDictionary<string> = {};

  constructor(
    private platform: PlatformService,
    private state: StateService,
    @Optional() @Inject(NODE_PROCESS) private process: INodeProcess,
  ) {
    this.transferVariables();
  }

  public getVariable(variable: EnvironmentVariable): string {
    return this.environmentVariables[variable];
  }

  private transferVariables(): void {
    if (this.platform.isServer) {
      Object.keys(EnvironmentVariable).forEach((variable: EnvironmentVariable) => {
        if (this.process.env.hasOwnProperty(variable)) {
          this.environmentVariables[variable] = this.process.env[variable];
        }
      });

      this.state.set(this.transferKey, this.environmentVariables);
    }

    if (this.platform.isBrowser) {
      this.environmentVariables = this.state.get(this.transferKey, {});
    }
  }
}

```

```
@Injectable()
export class EnvironmentVariablesService {
  private transferKey: StateKey<IDictionary<string>> = this.state.makeKey('env');
  private environmentVariables: IDictionary<string> = {};

  constructor(
    private platform: PlatformService,
    private state: StateService,
    @Optional() @Inject(NODE_PROCESS) private process: INodeProcess,
  ) {
    this.transferVariables();
  }

  public getVariable(variable: EnvironmentVariable): string {
    return this.environmentVariables[variable];
  }

  private transferVariables(): void {
    if (this.platform.isServer) {
```

```
@Injectable()
export class EnvironmentVariablesService {
  private transferKey: StateKey<IDictionary<string>> = this.state.makeKey('env');
  private environmentVariables: IDictionary<string> = {};

  constructor(
    private platform: PlatformService,
    private state: StateService,
    @Optional() @Inject(NODE_PROCESS) private process: INodeProcess,
  ) {
    this.transferVariables();
  }

  public getVariable(variable: EnvironmentVariable): string {
    return this.environmentVariables[variable];
  }

  private transferVariables(): void {
    if (this.platform.isServer) {
```

```
@Injectable()
export class EnvironmentVariablesService {
  private transferKey: StateKey<IDictionary<string>> = this.state.makeKey('env');
  private environmentVariables: IDictionary<string> = {};

  constructor(
    private platform: PlatformService,
    private state: StateService,
    @Optional() @Inject(NODE_PROCESS) private process: INodeProcess,
  ) {
    this.transferVariables();
  }

  public getVariable(variable: EnvironmentVariable): string {
    return this.environmentVariables[variable];
  }

  private transferVariables(): void {
    if (this.platform.isServer) {
```

```
  ) {
    this.transferVariables();
  }

  public getVariable(variable: EnvironmentVariable): string {
    return this.environmentVariables[variable];
  }

  private transferVariables(): void {
    if (this.platform.isServer) {
      Object.keys(EnvironmentVariable).forEach((variable: EnvironmentVariable) => {
        if (this.process.env.hasOwnProperty(variable)) {
          this.environmentVariables[variable] = this.process.env[variable];
        }
      });

      this.state.set(this.transferKey, this.environmentVariables);
    }

    if (this.platform.isBrowser) {
      this.environmentVariables = this.state.get(this.transferKey, {});
    }
  }
```

```
10  ) {
11    this.transferVariables();
12  }
13
14  public getVariable(variable: EnvironmentVariable): string {
15    return this.environmentVariables[variable];
16  }
17
18  private transferVariables(): void {
19    if (this.platform.isServer) {
20      Object.keys(EnvironmentVariable).forEach((variable: EnvironmentVariable) => {
21        if (this.process.env.hasOwnProperty(variable)) {
22          this.environmentVariables[variable] = this.process.env[variable];
23        }
24      });
25
26      this.state.set(this.transferKey, this.environmentVariables);
27    }
28
29    if (this.platform.isBrowser) {
30      this.environmentVariables = this.state.get(this.transferKey, {});
31    }
32  }
```

```
18   private transferVariables(): void {
19     if (this.platform.isServer) {
20       Object.keys(EnvironmentVariable).forEach((variable: EnvironmentVariable) => {
21         if (this.process.env.hasOwnProperty(variable)) {
22           this.environmentVariables[variable] = this.process.env[variable];
23         }
24       });
25
26       this.state.set(this.transferKey, this.environmentVariables);
27     }
28
29     if (this.platform.isBrowser) {
30       this.environmentVariables = this.state.get(this.transferKey, {});
31     }
32   }
33 }
34
```

```
3  private transferKey: StateKey<IDictionary<string>> = this.state.makeKey('env');
4  private environmentVariables: IDictionary<string> = {};
5
6  constructor(
7    private platform: PlatformService,
8    private state: StateService,
9    @Optional() @Inject(NODE_PROCESS) private process: INodeProcess,
10 ) {
11   this.transferVariables();
12 }
13
14 public getVariable(variable: EnvironmentVariable): string {
15   return this.environmentVariables[variable];
16 }
17
18 private transferVariables(): void {
19   if (this.platform.isServer) {
20     Object.keys(EnvironmentVariable).forEach((variable: EnvironmentVariable) => {
21       if (this.process.env.hasOwnProperty(variable)) {
22         this.environmentVariables[variable] = this.process.env[variable];
23       }
24     });
25
26   this.state.set(this.transferKey, this.environmentVariables);
```

RESPONSIVE DESIGN ON SERVER-SIDE 🤔

RESPONSIVE DESIGN ON SERVER-SIDE 🤔

@ANGULAR/FLEX-LAYOUT:

- Readme about SSR: https://github.com/angular/flex-layout/blob/master/guides/SSR.md
- Flex-layout SSR design document: https://docs.google.com/document/d/1fgO4ihw42dJJHGd6fugdiBe39iJot8aErhiE7CjwfmQ/edit
- **tl;dr:** boils down to rendering all elements and showing/hiding with CSS

RESPONSIVE DESIGN ON SERVER-SIDE 🤔

@ANGULAR/FLEX-LAYOUT:

- Readme about SSR: https://github.com/angular/flex-layout/blob/master/guides/SSR.md
- Flex-layout SSR design document: https://docs.google.com/document/d/1fgO4ihw42dJJHGd6fugdiBe39iJot8aErhiE7CjwfmQ/edit
- **tl;dr:** boils down to rendering all elements and showing/hiding with CSS

USER-AGENT PARSING:

- Try to match to one of your breakpoints based on UA device type

TURNING OFF JS IN THE BROWSER 😱

TURNING OFF JS IN THE BROWSER 😱

THINGS WHICH SHOULD WORK OUT-OF-THE-BOX:

- Displaying content
- Navigation
- Authorization

TURNING OFF JS IN THE BROWSER 😱

THINGS WHICH SHOULD WORK OUT-OF-THE-BOX:

- Displaying content
- Navigation
- Authorization

THINGS WHICH COULD WORK WITH SOME EXTRA EFFORT:

- Forms
  - Submission
  - Validation
- Lists with CRUD

TURNING OFF JS IN THE BROWSER 😱

THINGS WHICH SHOULD WORK OUT-OF-THE-BOX:

- Displaying content
- Navigation
- Authorization

THINGS WHICH COULD WORK WITH SOME EXTRA EFFORT:

- Forms
  - Submission
  - Validation
- Lists with CRUD

PROBLEMATIC:

- Dynamic interactions
  - Try mimicking interactions with forms
  - Make app fully-routable

INFINUM

Thank you!

FILIP.VOSKA@INFINUM.CO

Visit infinum.co or find us on social networks:

infinum.co infinumco infinumco infinum

INFINUM

Questions?

Visit infinum.co or find us on social networks:

infinum.co infinumco infinumco infinum