

# Pipes, JWT, Interceptors

## Authentication, Intercepting HTTP Requests



**Software University**

<https://softuni.bg>



**SoftUni**



**SoftUni Team**  
**Technical Trainers**

# Table of Contents

- Pipes
- JWT Authentication
- Interceptors
- Lazy Loading
- Subjects



sli.do


**#js-web**



# **Pipes**

**Usage, Chaining, Creating Custom**

# What Are Pipes ?

- 
- Pipes in Angular are used to **transform data** in the template
  - It takes integers, strings, arrays and date as input separated with **|** to be converted

```
{{ username | uppercase }}
```

- Pipes can also be **chained**

```
{{ username | lowercase | titlecase }}
```

Keep the **order** in mind

- Some pipes in Angular take **parameters**

```
{{ data.creationDate | date: 'fullDate' }}
```

```
{{ data.creationDate | date: 'MM/dd/yyyy' }}
```

- More on pipes in the documentation
  - <https://angular.io/api?query=pipe>

Import in **declarations**

```
@Pipe({
  name: 'shorten'
})
export class ShortenPipe implements PipeTransform {
  transform(value: string) {
    if (value.length > 10) {
      return `${value.substr(0, 10)}...`;
    }

    return value;
  }
}
```

```
{{ description | shorten }}
```

- Custom Pipes can also receive parameters

```
transform(value: string, limit: number) {  
  if (value.length > limit) {  
    return `${value.substr(0, limit)}...`;  
  }  
  return value;  
}  
{{ description | shorten:10 }}
```



# Async Pipe - Example

- Execute asynchronous code (promises, observables) using the **async pipe**

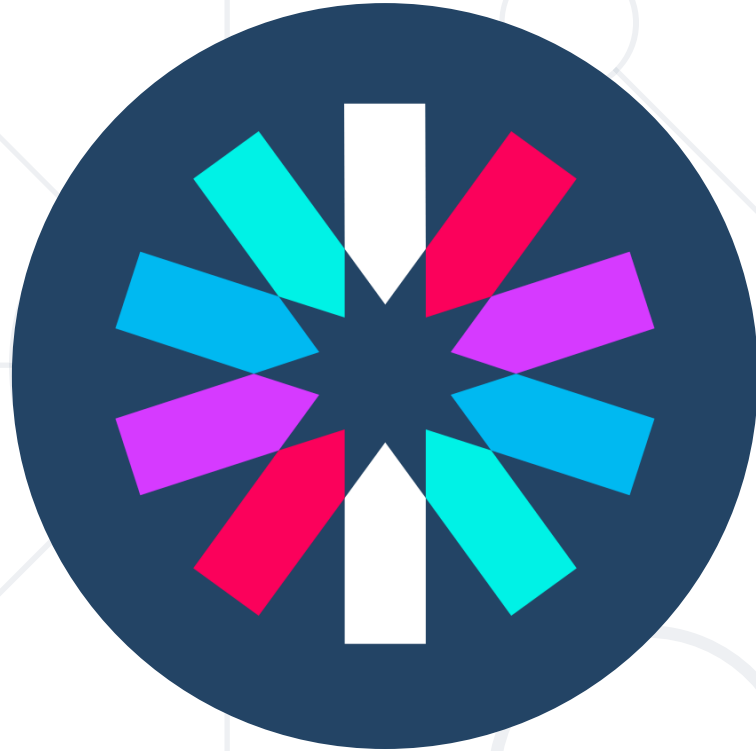
```
text = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    resolve('show some text');  
  }, 3000)  
})
```

```
<h1>{{ text | async }}</h1>
```

- Async pipe takes care of **subscribing** and **unwrapping** the data
- As well as **unsubscribing** when the component is **destroyed**

```
export class PostsComponent implements OnInit {  
  posts$ : Observable<Post[]>  
  
  ngOnInit() {  
    this.posts$ = this.postsService.getAllPosts();  
  }  
}
```

```
<div *ngFor="let post of posts$ | async">...</div>
```



**JWT**  
**JSON Web Token**

# What is JWT?

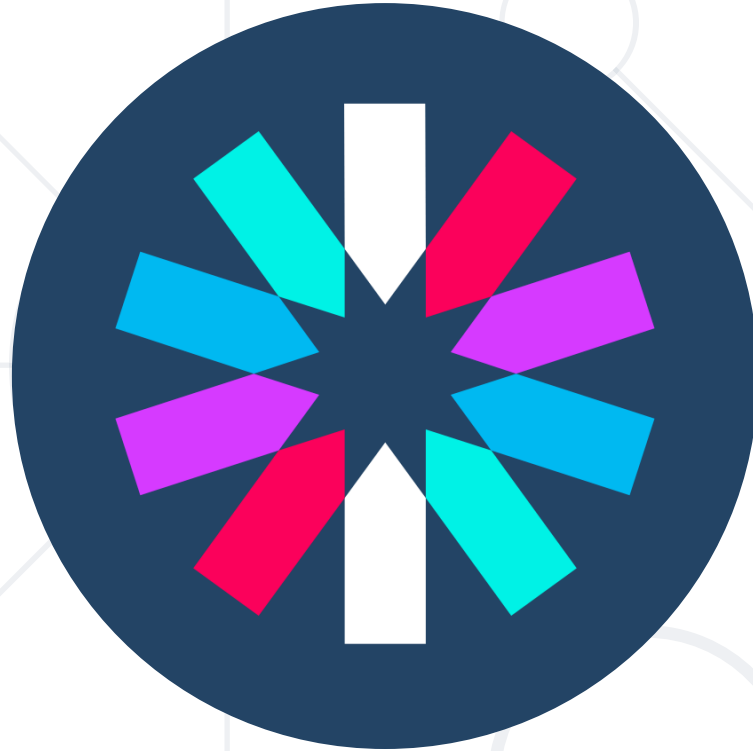
- **JSON Web Token (JWT)** is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object
- This information can be verified and trusted because it is digitally signed
- JWTs can be signed using a secret or a public/private key pair using **RSA** or **ECDSA**

# When should you use JWT?

- JSON Web Tokens are useful for
  - **Authorization** (most common scenario) - Once the user is logged in, each subsequent request will include JWT, allowing the user to access routes, services and resources that are permitted with that token
  - **Information Exchange**: JSON Web Tokens are good way of securely transmitting information between parties. Because they are signed digitally

- In its compact form, JSON Web Tokens consist of three parts separated by dots ( . )
  - **Header**
  - **Payload**
  - **Signature**

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4  
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.  
4pcPyMD09o1PSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```



# **HTTP Interceptors**

## **Attaching Tokens, Error Handling**

# Interceptors Overview

- Automatically **attach** authentication information to **requests**
- Often involves attaching **tokens**
  - JSON Web Token (JWT)
  - Other form of access tokens
- Implemented since **Angular 4** using **HttpInterceptor**





# Create Http Interceptor

- Import the following

```
import {  
  HttpResponse,  
  HttpRequest,  
  HttpHandler,  
  HttpEvent,  
  HttpInterceptor  
} from '@angular/common/http'
```

- All interceptors that we create are **injectables** and **implement** the **HttpInterceptor** interface

```
export class TokenInterceptor implements HttpInterceptor
```

- The interface gives us an **intercept** method

```
intercept(request: HttpRequest<any>, next: HttpHandler):  
Observable<HttpEvent<any>> {  
  
    request = request.clone({  
        setHeaders: {  
            Authorization: `Bearer ${this.authService.token}`,  
            Content-Type: 'application/json'  
        }  
    });  
  
    return next.handle(request);  
}
```

To make changes **clone**  
the **original** request

Passing control to **next**  
**interceptor** in the **chain**

# Provide the Interceptor

- The interceptor needs to be added to the **HTTP\_INTERCEPTORS** array (in **app.module.ts**)

```
import { HTTP_INTERCEPTORS } from '@angular/common/http'
```

- Provide it the following way

```
providers: [  
  {  
    provide: HTTP_INTERCEPTORS,  
    useClass: TokenInterceptor,  
    multi: true  
  }  
]
```

- Handle responses using the **pipe** and **tap** operators

```
import { tap } from 'rxjs/operators'
```

```
return next.handle(req)  
  .pipe(tap((event : HttpEvent<any>) => {  
    if (event instanceof HttpResponse  
      && req.url.endsWith('login')) {  
      this.saveToken(event);  
    }  
  })
```

- Handle server errors with **catchError** and **throwError** operators

```
import { catchError } from 'rxjs/operators'
import { throwError } from 'rxjs'
```

```
return next.handle(req)
  .pipe(catchError((err: HttpResponse) => {
    if (err.status === 401) {
      // Log the errors
      this.router.navigate([ '/login' ])
    }
    return throwError(err);
  }
));
```



# **Lazy Loading**

Asynchronous Loading

# What is Lazy Loading ?

- Loading **everything** in a big bundle could be **slow**
- Lazy Loading helps us to **download** the web pages **in chunks**
- In **Angular** this is done by firstly organizing the application into **separate modules**
- The module should be loaded the moment a user navigates to the **main route**



- Create a Feature Module - **Furniture Module**
  - Components - FurnitureAll, FurnitureDetails, FurnitureEdit
- Create a **separate** routing module

```
const furnitureRoutes = [ { path: '', children: [...] } ]
@NgModule({
  imports: [
    RouterModule.forChild(furnitureRouting)
  ],
  exports: [
    RouterModule
  ]
})
```



# Warning - Don't Import in Bootstrap Module

- The Feature Module **shouldn't** be imported inside the **bootstrap module** (app.module.ts)

```
import { AppFurnitureModule } from '../furniture/...'
@NgModule({
  imports: [
    AppFurnitureModule // Avoid this
  ]
})
```

Loaded at **app start**

- Instead use **loadChildren** inside the **main routing**

```
const routes: Routes = [  
  { path: 'signin', component: SigninComponent },  
  { path: 'signup', component: SignupComponent },  
  {  
    path: 'furniture',  
    loadChildren: import('./furniture/furniture.module')  
      .then(m => m.AppFurnitureModule)  
  }  
]
```

The name of the exported class

- To protect lazy loaded modules use a **canLoad guard** instead of **canActivate guard**
- AuthGuard should implement the **CanLoad interface**

```
{  
  path: 'furniture',  
  loadChildren: import('./furniture/furniture.module')  
    .then(m => m.AppFurnitureModule),  
  canLoad: [ AuthGuard ]  
}
```



# Subjects

Observer and Observable

# What is a Subject ?

- An RxJS Subject is a **special type** of Observable
- It allows values to be **multicast**ed to many **Observers**
- Subjects are like **Event Emitters**
  - They maintain a registry of many listeners
- Every Subject is an **Observable** - has **subscribe()**
- Every Subject is an **Observer** - has methods **next()**, **error()** and **complete()**



- Subject is an Observer - provide it to the **subscribe**

```
let subject = new Subject();
subject.subscribe({
  next: (v) => console.log(`observerA: ${v}`)
});
subject.subscribe({
  next: (v) => console.log(`observerB: ${v}`)
});
let observable = from([1, 2, 3]);
observable.subscribe(subject);
```



```
observerA: 1
observerB: 1
observerA: 2
observerB: 2
observerA: 3
observerB: 3
```

# Behavior Subject

- One of the variants is the **BehaviorSubject**
  - has the notion of "**the current value**"
- Stores the **latest value** emitted to its consumers
- Whenever a new Observer subscribes - it receives the current value from the BehaviorSubject



*BehaviorSubjects are useful for representing "values over time". For instance, an event **stream of birthdays** is a **Subject**, but the stream of a **person's age** would be a **BehaviorSubject***

# Behavior Subject - Example

- Behavior Subject **initialized** with a value of **0**

```
let subject = new BehaviorSubject(0);  
subject.subscribe({  
  next: (v) => console.log(`observerA: ${v}`)  
});  
subject.next(1);  
subject.next(2);  
subject.subscribe({  
  next: (v) => console.log(`observerB: ${v}`)  
});  
subject.next(3);
```



```
observerA: 0  
observerA: 1  
observerA: 2  
observerB: 2  
observerA: 3  
observerB: 3
```



# Replay Subject

- A **ReplaySubject** is like a BehaviorSubject in that it can send **old values** to **new subscribers**
- It can also record a part of the Observable execution

*A ReplaySubject records **multiple values** from the Observable execution and replays them to new subscribers.*



# Replay Subject - Example

- When creating a ReplaySubject, you can specify **how many values** to replay

```
let subject = new ReplaySubject(2);  
subject.subscribe({ // TODO: same code });  
subject.next(1);  
subject.next(2);  
subject.next(3);  
subject.subscribe({ // TODO: same code });  
subject.next(4);
```



```
observerA: 1  
observerA: 2  
observerA: 3  
observerB: 2  
observerB: 3  
observerA: 4  
observerB: 4
```

# Async Subject

- The **AsyncSubject** is a variant where only the **last value** of the Observable execution is sent to its observers
- It is sent only when the **execution completes**
- AsyncSubject can still be used to multicast just like you would with a normal Subject



# Async Subject - Example

```
let subject = new AsyncSubject();  
subject.subscribe({ // TODO: same code });  
subject.next(1);  
subject.next(2);  
subject.next(3);  
subject.subscribe({ // TODO: same code });  
subject.next(5);  
subject.complete();
```

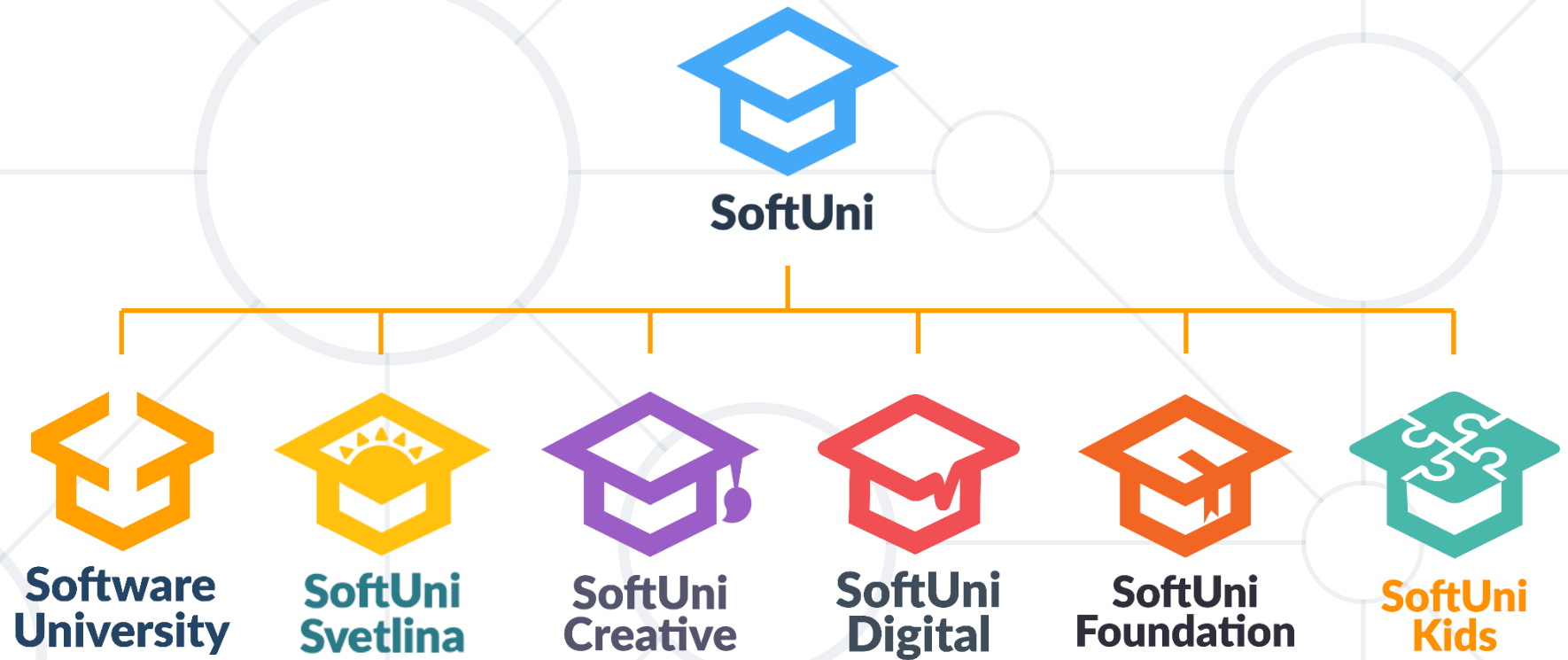


observerA: 5  
observerB: 5

- Pipes **transform** data
- Authentication with **JWT**
- Http Interceptors can modify **headers**
- Lazy loading help us to **download** the web pages in **chunks**
- Subject is a **special type** of **Observable**



# Questions?



# SoftUni Diamond Partners

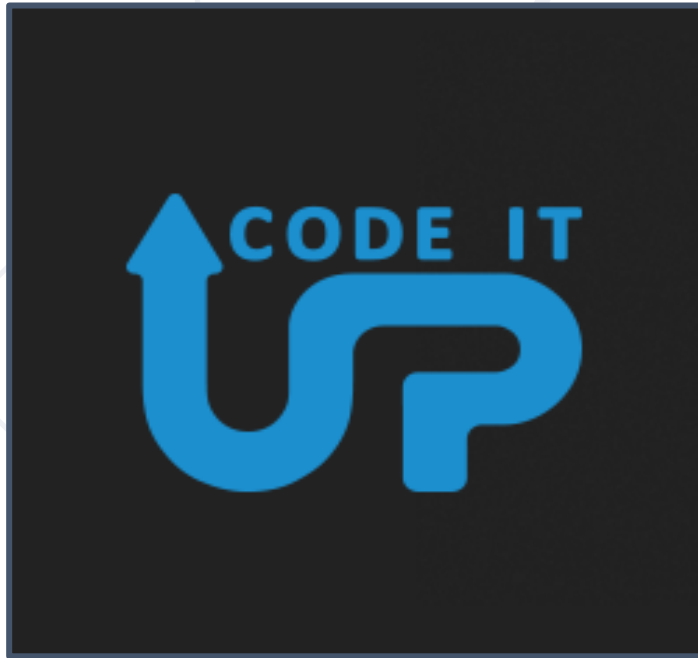


**SCHWARZ**



**SUPER  
HOSTING  
.BG**





**VIRTUAL RACING SCHOOL**





- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](http://softuni.bg), [softuni.org](http://softuni.org)

- Software University Foundation

- [softuni.foundation](http://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)

- Software University Forums

- [forum.softuni.bg](http://forum.softuni.bg)



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>

