# Backend communication

## with observables

## Make an AJAX request

The `HttpClient` class in the `HttpClientModule` is responsible for making AJAX requests.

```
import { HttpClientModule } from '@angular/common/http';

@NgModule({
    imports: [..., HttpClientModule],
    declarations: [...],
    providers: [...],
    bootstrap: [...]
})
export class AppModule { }
```

© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

# Make an AJAX request

Inject `HttpClient` in the constructor and send out a request

```
constructor(private http: HttpClient) {
    this.http
        .get<Person[]>('api/person')
        .subscribe(persons => {
            this.persons = persons;
        });
}
```

Notice the `subscribe()` here. We're dealing with so-called observables.

# Observables are used a lot

With AJAX calls:

```
let observable = this.http
    .get<Car[]>('api/car')
    .subscribe(data => console.log(data));
```

With reactive forms:

```
this.form = this.fb.group({ ... });
this.form.valueChanges.subscribe(newValue => {
    console.log('newValue:', newValue);
});
```

With routes:

```
this.route.params
    .pipe(map(params => +params['id']))
    .subscribe(id => this.carId = id);
```

And more. `EventEmitter`, web socket connections

© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

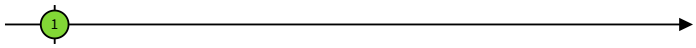
# So what are observables?

- A stream of data published by some source
  - Like an asynchronous array
  - Similar to LINQ in C# or Java 8 streams
- Listen for events in this stream by subscribing to the Observable
- Observables are lazy, no subscribers means no action is taken
- Interactive visual demo: http://rxmarbles.com
- Currently being proposed as a standard in JavaScript: http://kangax.github.io/compat-table/esnext/
- Use them now through Reactive Extensions for JavaScript (RxJS)

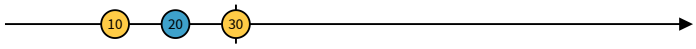
```
// static observable access methods
import { Observable, of, from, interval, empty } from 'rxjs';

// all operators (pipes)
import { map, share, count, distinct } from 'rxjs/operators';
```

# Creating observables

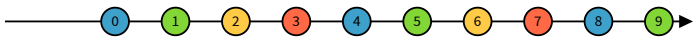
```
let data = of(1); // often used in unittests
```




```
let data = from([10, 20, 30]);
```




```
let data = interval(2000); // emit every 2 seconds
```



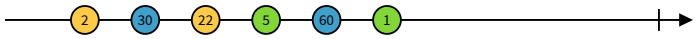

```
let o = Observable.create(() => {}); // often used in unittests
```

Thanks to RxJS Marbles

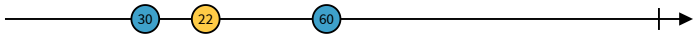
© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

# Filtering operators: `filter()`

```
let data = from([2, 30, 22, 5, 60, 1]).pipe(
    filter(x => x > 10)
);
```



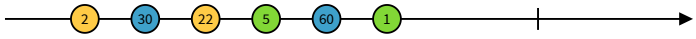

becomes



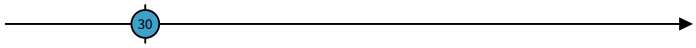

Thanks to RxJS Marbles

# Filtering operators: `find()`

```
let observableItem = from([2, 30, 22, 5, 60, 1]).pipe(
    find(x => x > 10)
);
```




becomes

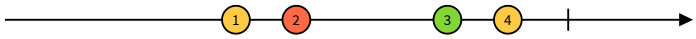



Thanks to RxJS Marbles

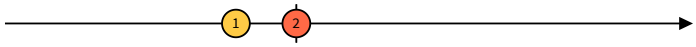
© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

# Filtering operators: `take()`

```
let observableItem = from([1, 2, 3, 4]).pipe(
    take(2)
);
```




becomes

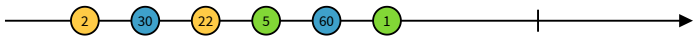



Similar: `first()`, `last()`, `takeLast(x)`

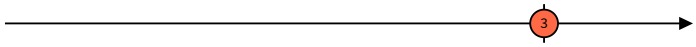
Thanks to RxJS Marbles

# Mathematical operators: `count()`

```
let countObservable = from([2, 30, 22, 5, 60, 1]).pipe(
    count(x => x > 10)
);
```




becomes



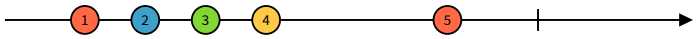

Thanks to RxJS Marbles

Similar: `max()` and `min()`

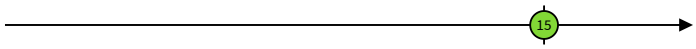
© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

# Mathematical operators: `reduce()`

```
let sumObservable = from([1, 2, 3, 4, 5]).pipe(
    reduce((prev, curr) => prev + curr)
);
```



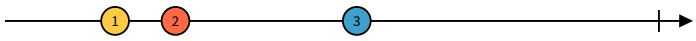

becomes



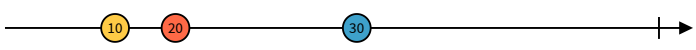

Thanks to RxJS Marbles

# Transformation operators: `map()`

```
let data = from([1, 2, 3]).pipe(
    map(x => x * 10)
);
```




becomes




Thanks to RxJS Marbles

© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

# Transformation operators

`flatMap()`

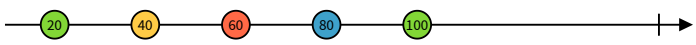
```
let employees = from([
    {
        name: 'Fred',
        languages: from(['Dutch', 'English'])
    },
    {
        name: 'Laura',
        languages: from(['French', 'English', 'Spanish'])
    }
]);
console.log('Our employees can converse with you in:');
employees.pipe(
    flatMap(x => x.languages),
    distinct()
).subscribe(lang => console.log(`- ${lang}`));
```

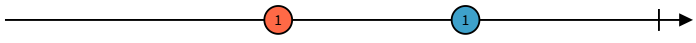
Similar: `switchMap()`

# Combination operators:

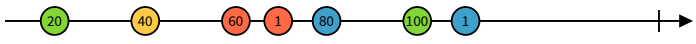
`merge()`

```
let data1 = timer(0, 1000).pipe(map(x => x * 20 + 20));
let data2 = timer(2500, 2000).pipe(map(x => 1));
data1.pipe(merge(data2)).subscribe(val => console.log('val:', val));
```




becomes




Similar: `combineLatest()`, `concat()`

© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

# Testing observables

```
it('should retrieve data during init', () => {
    let person = { id: 28, name: 'Frank' };
    spyOn(myComponent, 'getData').and.returnValue(of(person));

    myComponent.ngOnInit();

    expect(myComponent.getData).toHaveBeenCalled();
    expect(myComponent.user).toEqual(person);
});
```

# What about promises?

- In web development land thus far, promises are the popular choice for backend calls
- Working with promises is still possible:

```
this.http
    .get<Person[]>('api/people')
    .toPromise()
    .then(people => {
        this.people = people;
    });
```

© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

# Promises vs Observables

**Promises**

- Are eager. Whether someone is awaiting the result or not, a promise starts doing the asynchronous work.
- Do their job once, resolving with one result.

**Observables**

- Are lazy. An observable doesn't start until a subscriber is present.
- Can be cancelled. An Observable can return a dispose function.
- Is not limited to one result. Ideal for Web Sockets.
  - Observables can be seen as a stream of results published by some source.

# AsyncPipe

- A pipe that deals with observables/promises
- Expose the observable

```
people: Observable<Person[]>;
constructor(private http: Http) {
    this.people = this.http.get<Person[]>('api/people');
}
```

- Use in your HTML:

```
<span *ngFor="let p of people | async">{{p.name}}</span>
```

```
<p>Names: {{peopleObservable | async | greetAll}}</p>
```

© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

# Posting data

It resembles GET

```
save(person: Person): Observable<Person> {
    let headers = new HttpHeaders({
        'Auth-Token': 'my-auth-token'
    });

    return this.http
        .post<Person>('api/people', person, { headers: headers })
        .pipe(catchError(this.handleError))
        .subscribe(res => res.data);
}
```

# HTTP interceptors

Modify requests/responses by implementing interceptors.

Here's a request interceptor that adds a couple of HTTP headers to every request.

```
@Injectable()
export class AuthTokenInterceptor implements HttpInterceptor {
    constructor(private auth: AuthService) { }

    intercept(req: HttpRequest<any>, next: HttpHandler) {
        let headers = req.headers.set('Content-Type', 'application/json');
        if (this.auth.isLoggedIn) {
            headers = headers.append('Auth-Token', this.auth.getAuthToken());
        }

        const authReq = req.clone({ headers }); // requests are immutable
        return next.handle(authReq);
    }
}
```

© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

# HTTP interceptors

And here's a response interceptor that transforms the response.

```
@Injectable()
export class JsonResolverInterceptor implements HttpInterceptor {
    intercept(req: HttpRequest<any>, next: HttpHandler) {
        return next.handle(req).pipe(map(
            (ok: HttpResponse<any>) => {

                // do some sort of transformation here

                return ok.clone({
                    body: objects
                });
            }
        ));
    }
}
```

# HTTP interceptors

Provide your interceptors.

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

@NgModule({
    imports: [..., HttpClientModule, ...],
    declarations: [...],
    providers: [
        { provide: HTTP_INTERCEPTORS, useClass: AuthTokenInterceptor, multi: true },
        { provide: HTTP_INTERCEPTORS, useClass: JsonResolverInterceptor, multi: true },

        // ...
    ],
    bootstrap: [AppComponent]
})
export class AppModule { }
```

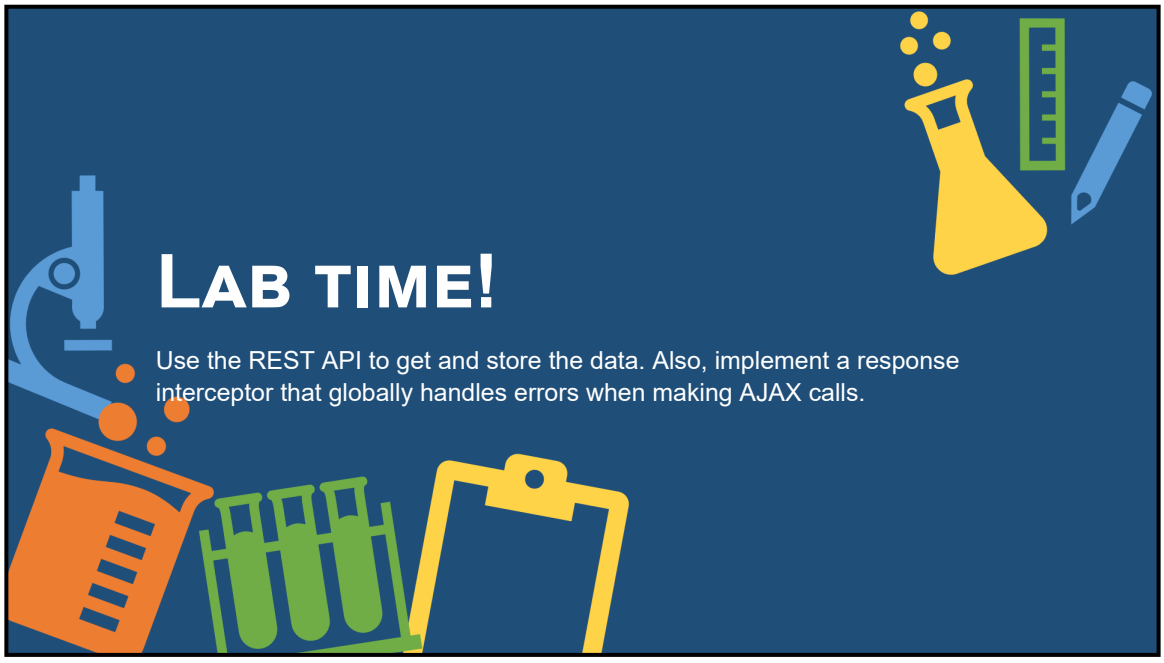
© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support

# Unittesting HTTP interceptors

Import the `HttpTestingModule` and inject the `HttpTestingController` to test for requests.

```
it('should add the authtoken if the user is logged in', () => {
    auth.isLoggedIn = true;
    auth.setAuthToken('q');
    http.get('api/bla').subscribe(x => { });
    let request = httpController.expectOne(req =>
        req.headers.has('Auth-Token') &&
        req.headers.get('Auth-Token') === 'q'
    );
    request.flush({});
});
```

```
it('should not add the authtoken if the user is not logged in', () => {
    http.get('api/bla').subscribe(x => { });
    httpController.expectOne(req => !req.headers.has('Auth-Token'));
});
```




© Info Support · All Rights Reserved · This data may not be copied or distributed without the prior approval of Info Support