Promise
RxJS
Forms

Gopalakrishnan Subraamni

gs@nodesen.se

www.nodesen.se

Promise Introduction

- Deferred Execution
- Async IO, AJAX Request, Socket Request
- Worker Thread Delegation
- setTimeout, setInterval

- Fulfil promise (Resolve)
- Fail Promise (Reject)

Promise

ES6 Promise

```
function doDelayedAsyncTask() {
    var promise = new Promise(function(resolve, reject) {
          setTimeout(function() {
                try {
                    //code here
                    //result returned after 3 seconds
                    resolve(result)
                }
                catch (err) {
                  //errors are send as reject
                   reject(err)
                }
          }, 3000); //called after 3 seconds

      return promise;
}
```

ES6 Promise

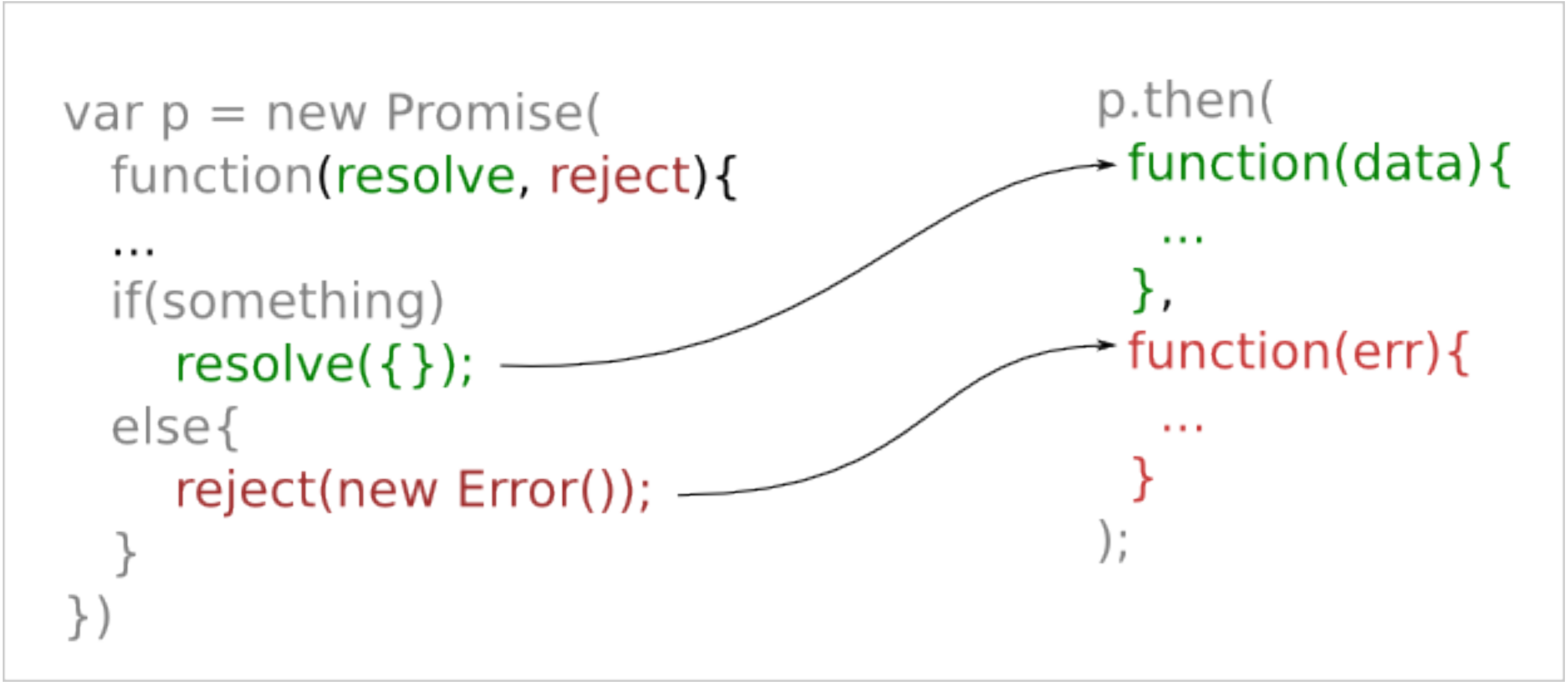
```
var promise = doDelayedAsyncTask();

promise.then(function resolveCallback(result) {
})

Or

 promise.then(null,
              function rejectCallback(error) {
              })
Or

promise.then(function resolveCallback(result) {
                         },
             function rejectCallback(error) {
             }
 )
```



```
var p = new Promise(
  function(resolve, reject){
  ...
  if(something)
     resolve({});
  else{
     reject(new Error());
  }
})
```

```
p.then(
  function(data){
    ...
  },
  function(err){
    ...
  }
);
```

Catch function

```
var promise = doDelayedAsyncTask();

promise.then(function (result) {
})

promise.catch(function (error) {
})
```

```
//OR
doDelayedAsyncTask()
.then(function (result) {
     })
 .catch(function (error) {
 })
```

Utility Functions

```
Promise.reject({error: 'everything failed'});
```

Create a promise with reject state

```
Promise.resolve({result:  true});
```

Create a promise with resolved/success state, useful to solve promise

Exceptions

```
function doDelayedAsyncTask() {
    var promise = new Promise(function(resolve, reject) {
     throw new Error("error")
    });

    return promise;
 }

//Exceptions converted into error
doDelayedAsyncTask()
.catch(function(error){
})

or

doDelayedAsyncTask()
.then(null, function(error){
})
```

Promise Chain

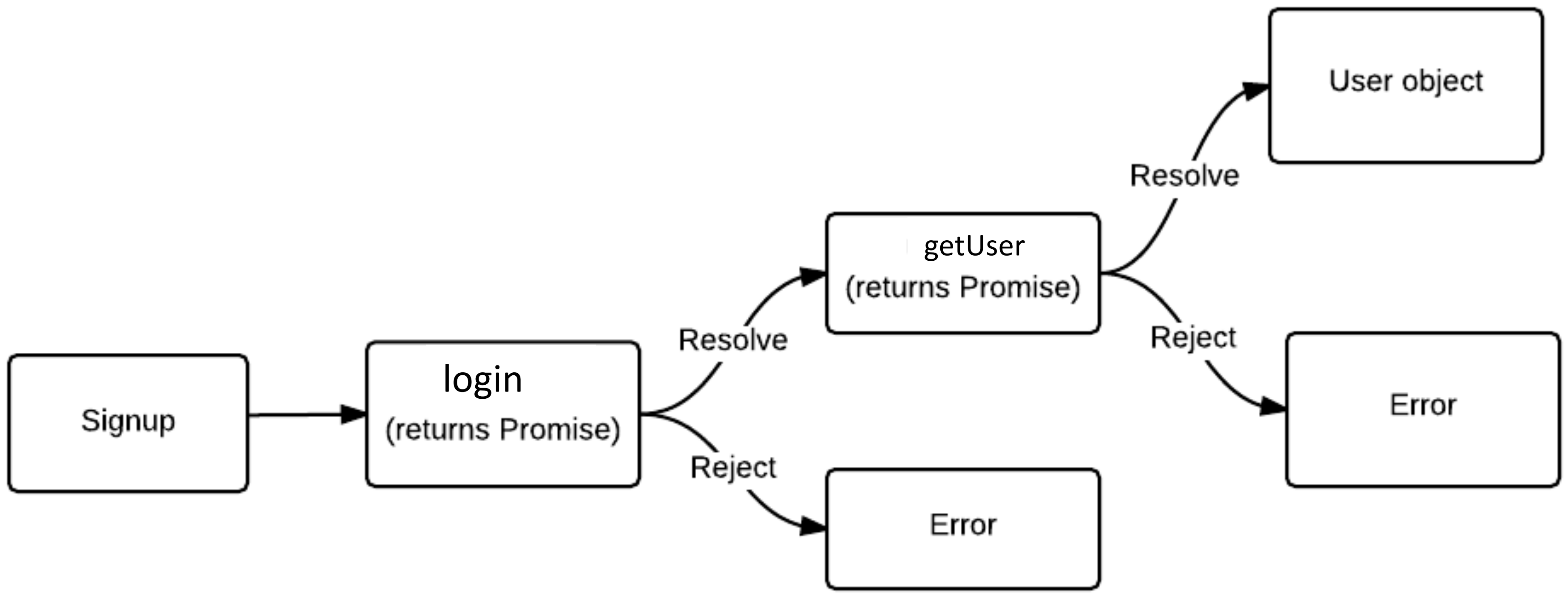
```
function login(user) {
    return new Promise(function(resolve, reject) {
        if(user.username != "" && user.password != "") {
            resolve(getUser(user));
        } else {
            reject("login failed,no user name/password")
        }
    });
}

function getUser(user) {
    return new Promise(function(resolve, reject) {
        setTimeout(function() {
            resolve({
                        name: 'Krish',
                        roles: ['Admin']
                    });
        }, 5000);
    });
};
```

```
login()
.then(function(user){

})
.catch(error) {

}
```


Signup
login
(returns Promise)
Resolve
Reject
getUser
(returns Promise)
Error
Resolve
Reject
User object
Error

Joining Promises

```
Promise.all([
    getProducts(),
    getBrands(),
    getStores()
])
.then(function (results){
    var products = results[0]; //products
    var brands = results[1]; //brands
    var stores = results[2]; //stores
})
```

Notes

Callback vs Promise

- **Callbacks are functions, promises are objects**
- **Callbacks** are just blocks of code which can be run *in response* to events such as as timers going off or messages being received from the server. Any function can be a callback, and every callback is a function.
- **Promises** are objects which store information about *whether or not* those events have happened yet, and if they have, *what their outcome is*.

Notes

Callback vs Promise

- **Callbacks are passed as arguments, promises are returned**
- **Callbacks** are defined independently of the functions they are called from – they are passed in as arguments. These functions then store the callback, and call it when the event actually happens.
- **Promises** are created *inside* of asynchronous functions (those which might not return a response until later), and then returned. When an event happens, the asynchronous function will update the promise to notify the outside world.

Notes

Callback vs Promise

- **Callbacks handle success and failure, promises don’t handle anything**
- **Callbacks** are generally called with information on whether an operation succeeded or failed, and must be able to handle both scenarios.
- **Promises** don’t handle anything by default, but success and failure handlers are attached later.

Notes

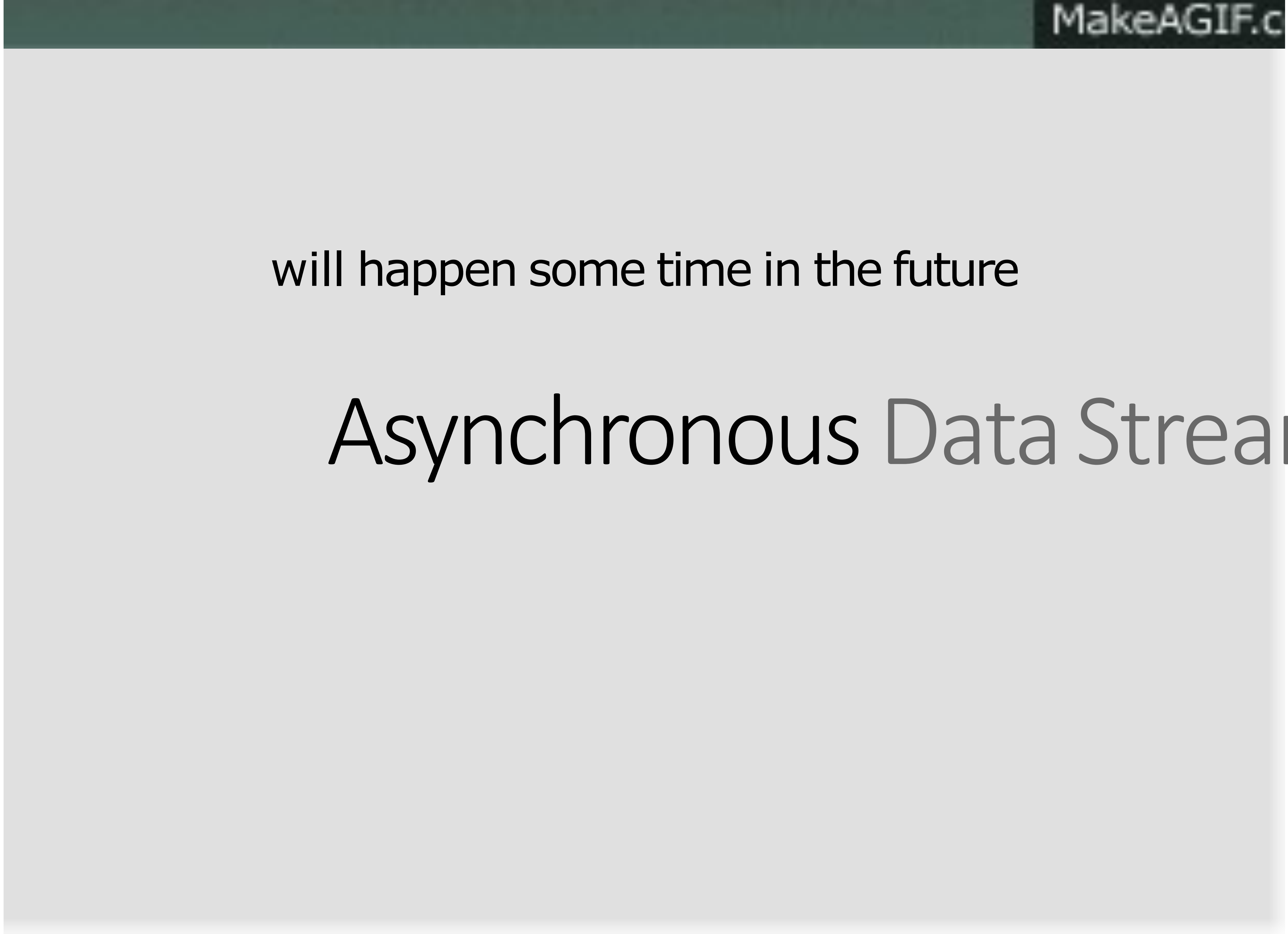
Callback vs Promise

- **Callbacks can represent multiple events, promises represent at most one**
- **Callbacks** can be called multiple times by the functions they are passed to.
- **Promises** can only represent one event – they are either successful once, or failed once.

RxJS

Reactive Extension

Source: Gerard Sans



will happen some time in the future

Asynchronous Data Streams

raw information

Asynchronous Data Streams

values made available overtime

Asynchronous Data Streams

Examples

Stream

❶ ❷ ❸

Array

[❶ , ❷ , ❸]

RxJS 5

beta2

Observable

```
//Observable constructor
let obs = new Observable(observer => {
try {
    //pushing values
    observer.next(1);
    observer.next(2);
    observer.next(3);
    //complete stream
    observer.complete()
  }
  catch(e) {
    //error handling
    observer.error(e);
  }
});

obs.subscribe( (n: number) => console.log(n));
```

Basic Stream

```
//ASCII Marble Diagram

----0----1----2----3---->   Observable.interval(1000);
----1----2----3|            Observable.fromArray([1,2,3])
----#                       Observable.of(1,2).do(x => th

---> is the timeline
0, 1, 2, 3 are emitted values
# is an error
| is the 'completed' signal
```

Observable helpers

```
//Observable creation helpers

Observable.of(1);                          // 1|
Observable.of(1,2,3).delay(100);           // ---1---2---

Observable.from(promise);

Observable.from(numbers$);
Observable.fromArray([1,2,3]);             // ---1---2---

Observable.fromEvent(inputDOMElement, 'keyup');
```

Subscribe

```
Observable.subscribe(
  /* next */       x => console.log(x),
  /* error */      x => console.log('#'),
  /* complete */   () => console.log('|')
);

Observable.subscribe({
  next:   x => console.log(x),
  error:  x => console.log('#'),
  complete:  () => console.log('|')
});
```

Unsubscribe

```
var subscriber = Observable.subscribe(
  twit => feed.push(twit),
  error => console.log(error),
  () => console.log('done')
);

subscriber.unsubscribe();
```

Subject

- import 'rxjs/Rx';
- import {Subject}   from 'rxjs/Subject';

- _source = new Subject<number>();
- _source.next(10);

- _source.subscribe((n : number) => {
- console.log(n);
- })

Operators

```
// simple operators
map(), filter(), reduce(), scan(), first(), last(), sin
elementAt(), toArray(), isEmpty(), take(), skip(), star

// merging and joining
merge(), mergeMap(flatMap), concat(), concatMap(),  swit
switchMap(), zip()

// spliting and grouping
groupBy(), window(), partition()

// buffering
buffer(), throttle(), debounce(), sample()
```

RxJS 5 usecases

- Asynchronous processing Http
- Forms: controls, validation Component events
  - EventEmitter

Why Observables?

- Flexible: sync or async
- Powerful operators
- Less code

Angular 2 Forms

Import

Defined as part of FormsModule

```
Import {FormsModule} from “@angular/forms”

@NgModule({
 …
Imports: [FormsModule]
})
```

Form Example

```
<input type="text"
       id="name"
        name="name”

       required
       minlength="4"
       maxlength="24”

       [(ngModel)]="product.name”


       #name="ngModel"
>
```

Basic HTML

HTML5 Validations

Angular 2 way binding

Angular 2 Template Variable

Errors

```
<input name=”productName" [(ngModel)]="product.name"
#name="ngModel" required minlength=“4” maxlength=“24”
>

<div *ngIf="name.errors && (name.dirty ||
name.touched)" class="alert alert-danger">
   <div [hidden]="!name.errors?.required"> Name is
required </div>
   <div [hidden]="!name.errors?.minlength"> Name must
be at least 4 characters long. </div>

 <div [hidden]="!name.errors?.maxlength"> Name cannot
be more than 24 characters long.
 </div>
</div>
```

”name” is local template variable

Field Properties

Property Name	Description
valid	Boolean (Indicate whether field is valid or not)
dirty	Boolean (indicate whether field is modified or not)
errors	Dictionary, useful to find specific error like errors.required, errors.minlength. Use with null check errors?.required

CSS Properties

State	When True	When False
visited	ng-touched	ng-untouched
Control's value has changed	ng-dirty	ng-pristine
Control's value is valid	ng-valid	ng-invalid

Form Level Error Checking

Adding ngSubmit force HTML5 validation

```
<form    #productForm="ngForm"
              (ngSubmit)="onSubmit()">
        ....
        ....

   <button type="submit"
           class="btn btn-default"
           [disabled]="!productForm.valid">
           Submit
   </button>

<form>
```

Input Types

```
<div>
       <label>Age</label>
    <input type="number" name="age”   [(ngModel)]="user.age">
 </div>
```

```
<label>
        <input type="checkbox"
                       name="isActive"
                       [(ngModel)]="user.isActive">
        Is Active
 </label>
```

Select with Value

```
<div>
       <label>Role</label>
       <select name="role" [(ngModel)]="user.role">
           <option *ngFor="let role of roles"
                  [value]="role.value">
                 {{role.display}}
          </option>
      </select>
 </div>
```

```
public roles = [
    { value: 'admin', display: 'Administrator' },
    { value: 'guest', display: 'Guest' },
    { value: 'custom', display: 'Custom' }
];
```

Select with Object

```
<div>
       <label>Role</label>
       <select name="role" [(ngModel)]="user.role">
           <option *ngFor="let role of roles"
                  [ngValue]="role">
                 {{role.display}}
          </option>
      </select>
 </div>
```

```
public roles = [
    { value: 'admin', display: 'Administrator' },
    { value: 'guest', display: 'Guest' },
    { value: 'custom', display: 'Custom' }
 ];
```

Multi Select with Values

```
<div>
       <label>Role</label>
       <select multiple name="role"
               [(ngModel)]="user.roles">
           <option *ngFor="let role of roles"
                  [value]="role.value">
                 {{role.display}}
          </option>
      </select>
 </div>
```

```
public roles = [
    { value: 'admin', display: 'Administrator' },
    { value: 'guest', display: 'Guest' },
    { value: 'custom', display: 'Custom' }
];
```

Quick Preview of Data using JSON

```
<pre> {{your_form or control_name | json }} </pre>
```