2. Get Ready

Developer

SQL Server 2017 Developer is a full-featured free edition, licensed for use as a development and test database in a non-production environment.

Download now

Microsoft **SQL Server** Management Studio 17

Desktop app

SSMS is free!

SSMS 17.x is the latest generation of *SQL Server Management Studio* and provides support for SQL Server 2017.

Download SQL Server Management Studio 17.7

Visual Studio Installer

Productos

Instalados

Visual Studio Community 2017

15.7.1

IDE gratuito y rico en contenido para estudiantes y desarrolladores individuales y de código abierto

Notas de la versión

Modificar | Iniciar | Más

Web y nube (7)

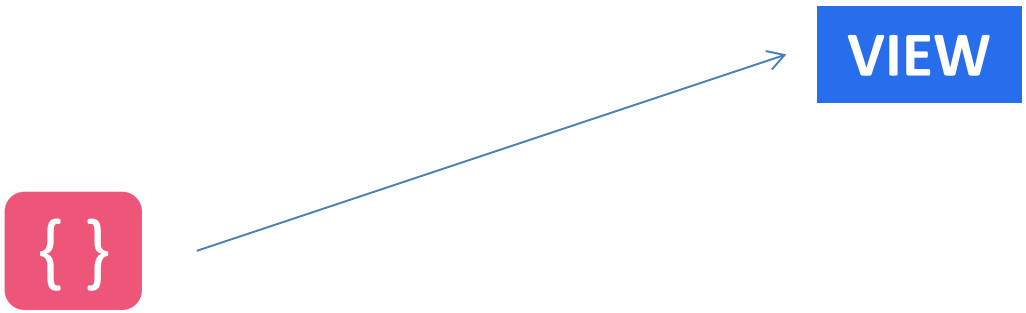
Desarrollo de ASP.NET y web
Compila aplicaciones web con ASP.NET, ASP.NET Core, HTML/JavaScript y Containers, además de ofrecer...

Desarrollo de Python
Edición, depuración, desarrollo interactivo y control de código fuente de Python.

Almacenamiento y procesamiento de datos
Conecte, desarrolle y pruebe soluciones de datos con SQL Server, Azure Data Lake o Hadoop.

Property Binding

```
<p>
  <input type="text" placeholder = 'Name:' />
  <input type="text" placeholder = "{{ placeholderName }}" />
  <input type="text" [placeholder] = "placeholderName" />
  <input type="text" bind-placeholder = "placeholderName"  />
</p>
```



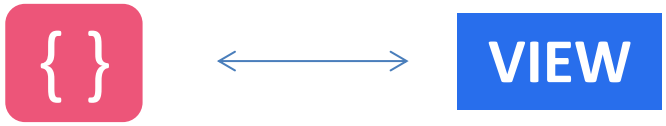

```
export class ParticipantFormComponent implements OnInit {

  placeholderName: String = "Name:" //Just for property binding
```

Two-Way Data Binding

[(ngModel)]

TS app.module.ts ✖

```
20    imports: [
21      BrowserModule,
22      AppRoutingModule,
23      FormsModule
24    ],
25    providers: [
26      ParticipantDataService
27    ],
28    bootstrap: [AppComponent]
29  })
30  export class AppModule { }
```

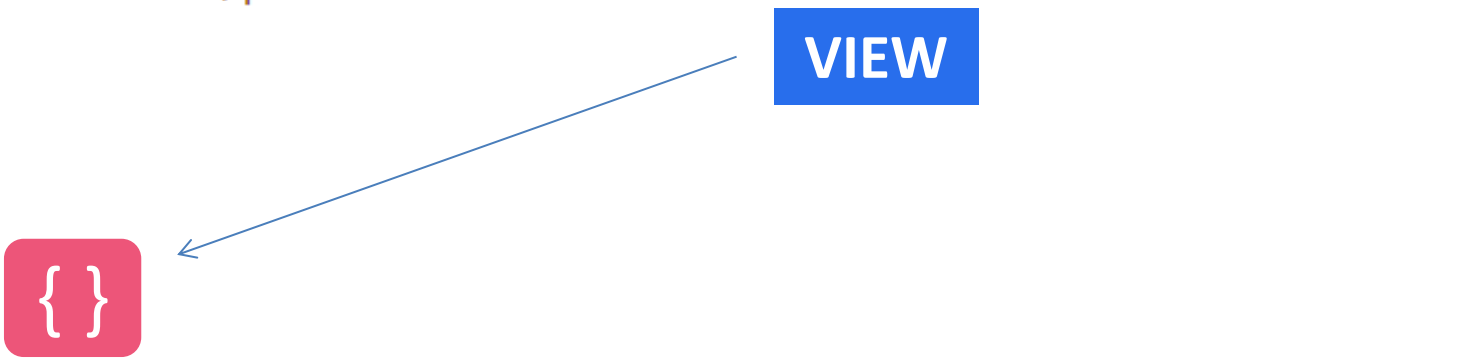



```
<p>
  <input type="text" placeholder = 'Name:' [(ngModel)]="participant.name" />
</p>
<p>
  {{ participant.name }}
</p>
```

Event Binding

(ngModel)

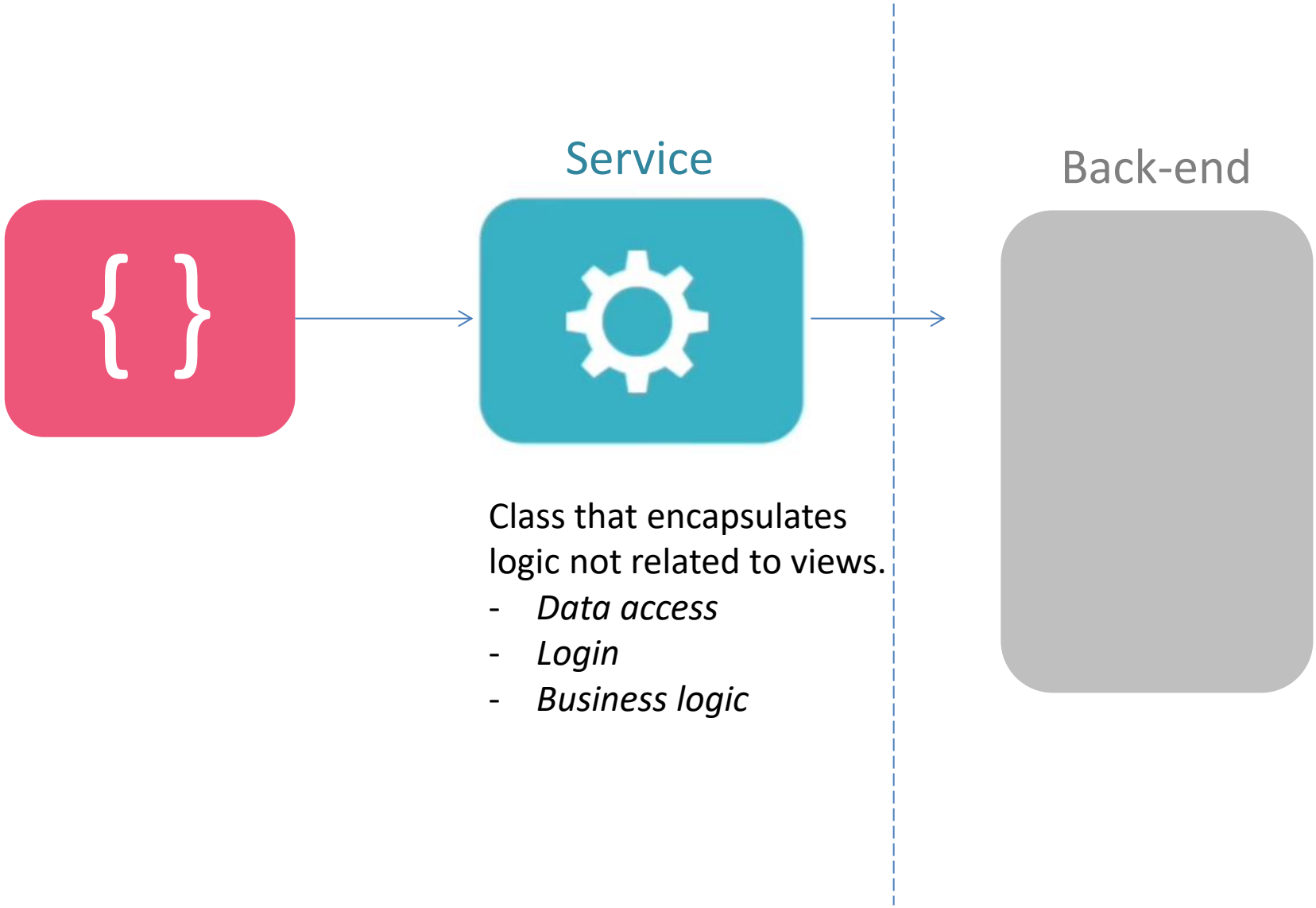
```
<p>
  <button (click)="newHandler(participant)"> New </button>
</p>
```




```
newHandler(participant1: Participant){
  //...
}
```

Service

- ✓ Components use *services*, which provide specific functionality not directly related to views.
- ✓ Service providers can be *injected* into components as *dependencies*, making your code modular, reusable, and efficient.



Class that encapsulates logic not related to views.

- *Data access*
- *Login*
- *Business logic*

1. Create the service.

```
ng generate service participant-data
```

participant-data.service.ts

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


@Injectable()
export class ParticipantDataService {

    constructor( )



}
```

2. Modify the service.

participant-data.service.ts

```
import { Injectable } from '@angular/core';
import { Participant } from './model/participant';

@Injectable()
export class ParticipantDataService {

    participants: Participant[];
    constructor( ) {…}

    getParticipants(): Participant[] {…}
    putParticipant(participante1: Participant) {…}
}
```

3. Add the service as a provider in the module.

```
@ngModule({

…
providers: [
    ParticipantDataService
],
…
```

4. Import the service.

In the component where we are going to use the service

name-list.component.ts

```
import { ParticipantDataService } from '…/participant-data.service'

participants: Participant[];


constructor() {


}
```

5. Use the service

In the component where we are going to use the service

name-list.component.ts

```
import { ParticipantDataService } from '…/participant-data.service'

participants: Participant[];
private participantDataService: ParticipantDataService;

constructor(participantDataService: ParticipantDataService ) {
        this.participantDataService = participantDataService;
        this.participants = participantDataService.getParticipants();
}
```

In the component where we are going to use the service

name-list.component.ts

```
import { ParticipantDataService } from '…/participant-data.service'

participants: Participant[];

constructor(
    private participantDataService: ParticipantDataService
) { }

ngOnInit() {
    this.participants = this.participantDataService.getParticipants();
}
```

- ✓ Create a private data
- ✓ Create a constructor parameter
- ✓ Initialize the data with the parameter

Web API

- ✓ Use **SSMS – MSSQL Server Management Studio** to create a DB and a table.
- ✓ Use **MSVS – MS Visual Studio** to create a new C# - ASP.NET Web Application.
- ✓ Use ADO.NET Entity Data Model to retrieve data from the database.
- ✓ Add the APIController.

HTTP Methods

- GET: Retrieve data from a specified resource
- POST: Submit data to be processed to a specified resource
- PUT: Update a specified resource
- DELETE: Delete a specified resource

Endpoint

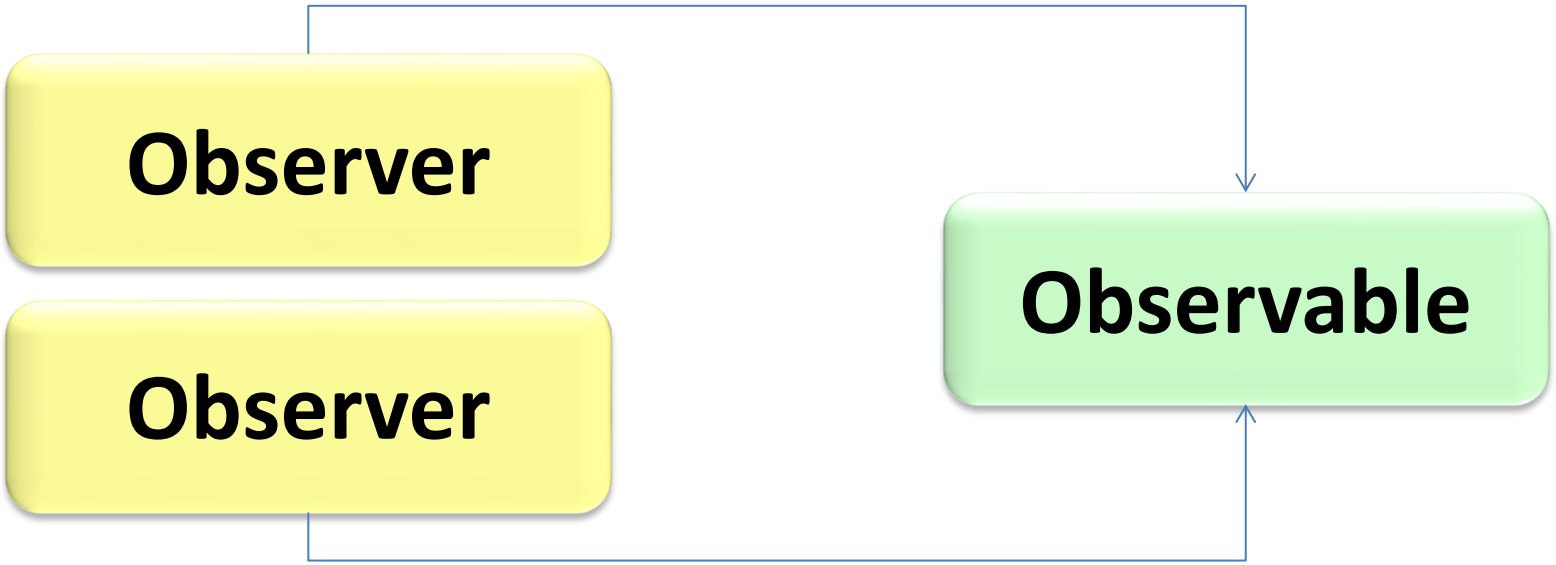
The URL where api/service can be accessed by a client application

GET http://localhost:56618/api/participants
GET http://localhost:56618/api/participants/LGF
PUT http://localhost:56618/api/participants/LGF

Observable

When version 2 of Angular came out, it introduced us to observables. The Observable isn't an Angular specific feature, but a new standard for managing async data that will be included in the ES7 release. Angular uses observables extensively in the event system and the HTTP service.

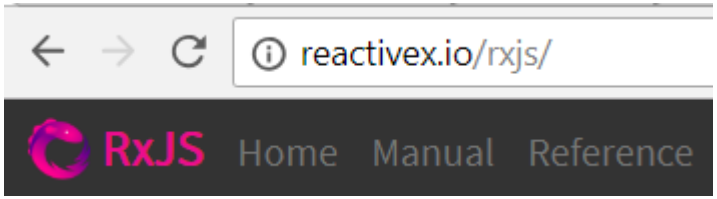
Asynchronous pattern



- *observers subscribe to an observable.*
- *observable emits an item or a sequence of items.*
- *observers react to whatever the observable emits.*

When the observer subscribes to the observable, it specifies the callback function.
The callback function is notified whenever the observable emits data items.
The callback function has code to handle the data.

The ReactiveX programming library for JavaScript



The ReactiveX library for JavaScript.

RxJS is a library for reactive programming using Observables, to make it easier to compose asynchronous or callback-based code. This project is a rewrite of Reactive-Extensions/RxJS with better performance, better modularity, better debuggable call stacks, while staying mostly backwards compatible, with some breaking changes that reduce the API surface.

```
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map'
```

_http:Http	: Observable<Response>	observableParticipants: Observable<Participant[ ]>
get(url): Observable<Response>	map( )	

```
@Injectable()
export class ParticipantDataService {

  observableParticipants: Observable<Participant[]>;

  constructor(
    private _http: Http
  ) {
    this.observableParticipants =
      this._http.get("http://localhost:56618/api/participants/")
        .map((response: Response) => <Participant[]>response.json());
  }

  getParticipants(): Observable<Participant[]> {
    return this.observableParticipants;
  }
```

observableParticipants:
Observable<Participant[ ]>

TS name-list.component.ts

```
ngOnInit() {
  this.participantDataService.getParticipants()
    .subscribe((participantsData)=>this.participants = participantsData);
}
```

HTTP - Terms and Concepts

- **Request Verbs (GET, POST, PUT & DELETE) :** These verbs describe what should be done with the resource. For the complete list of the HTTP verbs http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
- **Request Header :** contains additional information about the request. Example - what type of response is required
- **Request Body :** Contains the data to send to the server
- **Response Body :** Contains the data sent as response from the server
- **Response Status codes :** Provide the client, the status of the request. For the complete list of HTTP status codes http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

REST
Representational State Transfer

Representational State Transfer (**REST**) refers to a group of software architecture design constraints that bring about efficient, reliable, and scalable distributed systems.
A system is called RESTful when it adheres to those constraints.

A resource is really anything that can be pointed to via HTTP protocol. GET, PUT, POST are used to link resources to actions within a client-server relationship.

CRUD stands for Create, Read, Update, Delete. It's the basic operations that you expect to do on a database.

- *ngIf
- *ngFor

URL	HTTP Verb	POST Body	Result
/api/movies	GET	empty	Returns all movies
/api/movies	POST	JSON String	New movie Created
/api/movies/:id	GET	empty	Returns single movie
/api/movies/:id	PUT	JSON string	Updates an existing movie
/api/movies/:id	DELETE	empty	Deletes existing movie