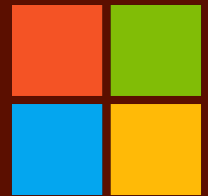


# Angular 6

MEAN Stack Developer



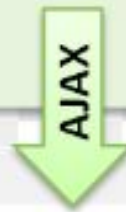
Jose Moyano Dutan.  
Expositor.

Microsoft UCuenca



ANGULARJS  
by Google

**Client**



**Server**



**Database**

---

Desarrollo del back-end usando mongodb y express js, además de la inicialización del proyecto en angular

<https://bit.ly/2MQ4R34>

# CONTENIDO

---

- ❏ Introducción a Angular 6.
- ❏ Fundamentos de TypeScript.
- ❏ Instalación y configuración
- ❏ Componentes
- ❏ Templates
- ❏ Metadatos y directivas
- ❏ Data Binding
- ❏ Servicios
- ❏ Aplicación web a través de MEAN Stack Development.

# Introducción

---

- Framework para desarrollar **aplicaciones web** modernas y escalables en el lado del cliente.
- Usa **TypeScript** para definir clases, propiedades y métodos.



# Introducción

---

- **Mejorar la productividad**, plantea arquitecturas estándar
- **Menos errores de código**, TypeScript proporciona detección temprana de errores.
- **Calidad del software**, TypeScript ayuda a comprender mejor cómo funciona internamente JavaScript.
- Angular respaldado por Google y posee una gran comunidad
- Proyectos totalmente **opensource**, publicados en GitHub



# TypeScript

---

- Lenguaje de programación **orientada a objetos**.
- TypeScript convierte su código en Javascript común.
- Es llamado **Superset** de Javascript, si el navegador está basado en Javascript, este nunca sabrá que el código original es TypeScript.



# Instalación y configuración

---

## Instalación

```
npm install -g @angular/cli
```

## Nuevo proyecto

```
ng new taller-mean
```

## Para probar localmente el nuevo proyecto

```
ng serve
```



# Modelos

---

Permite definir clases que se usarán dentro de la aplicación

Crear una clase:

```
ng generate class modelo/usuario
```

```
ng g cl modelo/mensaje
```

# Modelos

---

```
export class Usuario {  
  constructor(  
    public username: string = "",  
  ) {}  
}
```

```
export class Mensaje {  
  constructor(  
    public username: string = "",  
    public mensaje: string = "",  
  ) { }  
}
```

# Componentes

---

Un componente está compuesto de un controlador que siempre está ligado a una vista html.

Creación de un componente:

ng generate component componentes/ingreso

ng g c componentes/mensajeria

# Componentes

---

- ▲ componentes
  - ▲ ingreso
    - # ingreso.component.css
    - <> ingreso.component.html
    - TS ingreso.component.spec.ts
    - TS ingreso.component.ts
  - ▲ mensajeria
    - # mensajeria.component.css
    - <> mensajeria.component.html
    - TS mensajeria.component.spec.ts
    - TS mensajeria.component.ts

# Template

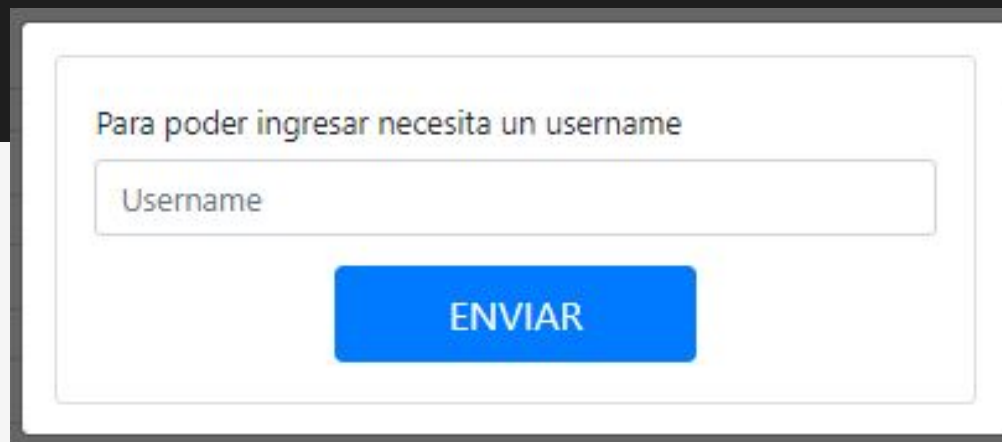
---

Permite definir la vista de un Componente.

El template es HTML decorado con otros componentes como css, scss, sass

# Template

```
<div class="card">
  <div class="card-body">
    <form (submit)="onSubmitLogin()">
      <div class="form-group">
        <label for="email">Para poder ingresar necesita un username</label>
        <input type="text" class="form-control" name="Username"
          [(ngModel)]="user.username" placeholder="Username" required>
      </div>
      <div class="col-md-6 offset-3">
        <input type="submit" value="ENVIAR" class="btn btn-primary btn-block btn-lg">
      </div>
    </form>
  </div>
</div>
```



Para poder ingresar necesita un username

# Directivas

---

## @Component

- selector: crear e instanciar el componente cuando se encuentra una etiqueta HTML con ese nombre.
- templateUrl: nombre del archivo HTML.
- styleUrls: nombre del archivo css, scss o sass.

@Input variables de ingreso, @Output variables de salida

```
@Component({  
  selector: 'app-ingreso',  
  templateUrl: './ingreso.component.html',  
  styleUrls: ['./ingreso.component.css']  
})
```

# Data Binding

---

- Interpolación: Al hacer `{{username}}`, se inserta el valor de esa propiedad en el HTML
- Property binding: Al hacer `[mensajes]="valor"`, envía información de una plantilla hija a la plantilla padre.
- Event binding: Al hacer `(submit)="onSubmitLogin(valor)"`, se indica cuando se produce un evento click
- Two-way binding: al ingresar `[(ngModel)]="mensaje"` utilizado para recibir datos desde etiquetas HTML



# Componente ingreso

---

```
<div class="card">
  <div class="card-body">
    <form (submit)="onSubmitLogin()">
      <div class="form-group">
        <label for="email">{{msgInicio}}</label>
        <input type="text" class="form-control" name="Username"
          [(ngModel)]="user.username" placeholder="Username" required>
      </div>
      <div class="col-md-6 offset-3">
        <input type="submit" value="ENVIAR" class="btn btn-primary btn-block btn-lg">
      </div>
    </form>
  </div>
</div>
```

# Componente ingreso

---

```
import { Component, Output, EventEmitter } from '@angular/core';
import { Usuario } from "../../modelo/usuario";

@Component({
  selector: 'app-ingreso',
  templateUrl: './ingreso.component.html',
  styleUrls: ['./ingreso.component.css']
})
export class IngresoComponent {
  msgInicio="Bienvenido"
  @Output() loginEvent = new EventEmitter();
  user = new Usuario;
  constructor() { }

  onSubmitLogin(){
    this.loginEvent.emit(this.user);
  }
}
```

# Metadatos

---

Directivas estructurales:

- \*ngIf: Condicional que se inserta en el DOM
- \*ngFor: repite su elemento en el DOM una vez por cada item que hay en el iterador.

```
*ngIf=!valida(m)
```

```
*ngFor="let m of mensajes"
```



# Componente Mensajería

---

```
<div class="col-md-12">
  <div class="row">
    <div class="col-md-10">
      <input type="text w-100" class="form-control" [(ngModel)]="msg.mensaje">
    </div>
    <div class="col-md-2">
      <button class="btn btn-primary w-100" (click)=newMessage()>Enviar</button>
    </div>
  </div>
</div>
```

```
.newTam{
  height: 600px;
  overflow-y: auto;
  overflow-x: hidden;
}
```



```
import { Component, Input, Output, EventEmitter } from '@angular/core';
import { Mensaje } from "../../modelo/mensaje";
import { Usuario } from "../../modelo/usuario";
```

```
@Component({
  selector: 'app-mensajeria',
  templateUrl: './mensajeria.component.html',
  styleUrls: ['./mensajeria.component.css']
})
```

```
export class MensajeriaComponent{
  msg = new Mensaje()
  @Input() mensajes:Mensaje[]
  @Input() user: Usuario
  @Output() createMessageEvent = new EventEmitter();
  constructor() { }
```

```
  newMessage() {
    if (this.msg.mensaje!=""){
      var msg = this.msg.mensaje
      this.createMessageEvent.emit(new Mensaje(this.user.username, msg));
      this.msg.mensaje="";
    }
  }
```

```
  valida(msg:Mensaje){
    if (msg.username == this.user.username) return true;
    else return false;
  }
```

```
}
```

# Componente Padre

---

Importamos las clases

```
import { Usuario } from "../modelo/usuario";  
import { Mensaje } from "../modelo/mensaje";
```

Agregamos los atributos

```
mensajes: Mensaje[] = [  
  new Mensaje("luis", "hola"),  
  new Mensaje("pedro", "como estas?")  
];  
user: Usuario = new Usuario;  
display = 'block';
```

# Componente Padre

---

Importamos las clases

```
import { Component, OnInit } from '@angular/core';  
import { Usuario } from "../modelo/usuario";  
import { Mensaje } from "../modelo/mensaje";
```

Agregamos los atributos

```
export class AppComponent implements OnInit {  
  mensajes: Mensaje[] = [  
    new Mensaje("luis", "hola"),  
    new Mensaje("pedro", "como estas?")  
  ];  
  user: Usuario = new Usuario;  
  display = 'block';  
  constructor(  
  ) { }
```



# Componente Padre

---

Las funciones

```
ngOnInit() {  
  this.getMessage();  
}  
getMessage() {  
  return this.mensajes;  
}  
createMessage(msg: Mensaje) {  
  this.mensajes.push(msg)  
}  
validate(user:Usuario){  
  if(user.username!=""){  
    this.user.username=user.username;  
    this.closeModalDialog()  
  }else{  
    this.openModalDialog()  
  }  
}  
openModalDialog() {  
  this.display = 'block';  
}  
closeModalDialog() {  
  this.display = 'none';  
}
```

# Componente Padre

---

```
<div class="container">
  <div class="card">
    <div class="card-body">
      <h3>Grupo de chat</h3>
      <app-mensajeria [mensajes]="mensajes" [user]="user" (createMessageEvent)="createMessage($event)">
      </app-mensajeria>
    </div>
  </div>
</div>

<div class="modal" tabindex="-1" role="dialog" [ngStyle]="{'display':display}">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-body">
        <app-ingreso (loginEvent)="validate($event)"></app-ingreso>
      </div>
    </div>
  </div>
</div>

<div class="backdrop" [ngStyle]="{'display':display}"></div>
```

# Componente Padre

---

```
.backdrop {  
  background-color: rgba(0,0,0,0.6);  
  position: fixed;  
  top: 0;  
  left: 0;  
  width: 100%;  
  height: 100vh;  
  z-index: 1000;  
}
```

# Servicios

---

Realizar comunicaciones con aplicaciones de terceros

Pueden ser:

- Servicio de logging
- Servicio de datos
- Bus de mensajes
- Cálculo de Impuestos
- Configuración de la app

# Servicios

---

Generar un nuevo servicio

ng g s servicio/rest-api

Vamos a app.module.ts e importamos el servicio

```
import { RestApiService } from './servicio/rest-api.service'
```

# Servicios

---

Agregamos en providers

```
@NgModule({
  declarations: [
    AppComponent,
    MensajeriaComponent,
    IngresoComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule
  ],
  providers: [
    RestApiService
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

# Aplicación web a través de MEAN Stack

---

Unir el back-end con el front-end

Importamos el servicio

```
import { RestApiService } from "../servicio/rest-api.service";
```

Instanciamos el servicio

```
constructor(  
  | private _service: RestApiService  
) { }
```

# Aplicación web a través de MEAN Stack

---

Agregamos el consumo del servicio

```
getMessages() {
  this._service.getMessages()
    .then(mensajes => this.mensajes = mensajes)
    .catch(err => console.log(err));
}

createMessage(msg: Mensaje) {
  this._service.createMsg(msg)
    .then(status => {this.getMessages()})
    .catch(err => console.log(err));
}

validate(user:Usuario){
  if(user.username!=""){
    this.user.username=user.username;
    this._service.createUser(user)
    this.closeModalDialog()
  }else{
    this.openModalDialog()
  }
}
```



# Aplicación web a través de MEAN Stack

---

Para generar la página que va a ser usando ejecutamos:

```
ng build
```

Se genera la carpeta dist, donde está toda la página web

Volvemos donde esta el archivo server.js y ejecutamos:

```
nodemon serve.js
```

# ENLACES DE INTERÉS

---

- Documentación de angular 6

<https://angular.io/guide/quickstart>

- Tarea:

Investigar socket.io





Comunidad  
Microsoft UCuenca

# MEAN Stack Developer

Mongo DB + Express JS + Angular 6 + Node JS



# MEDIOS DE CONTACTO

---



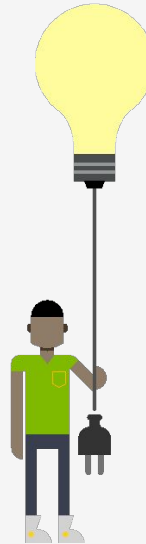
MicrosoftUCuenca



MicrosoftUC



MicrosoftUCuenca



MicrosoftUCuenca

[comunidad.microsoft@ucuenca.edu.ec](mailto:comunidad.microsoft@ucuenca.edu.ec)



Comunidad  
Microsoft UCuenca

