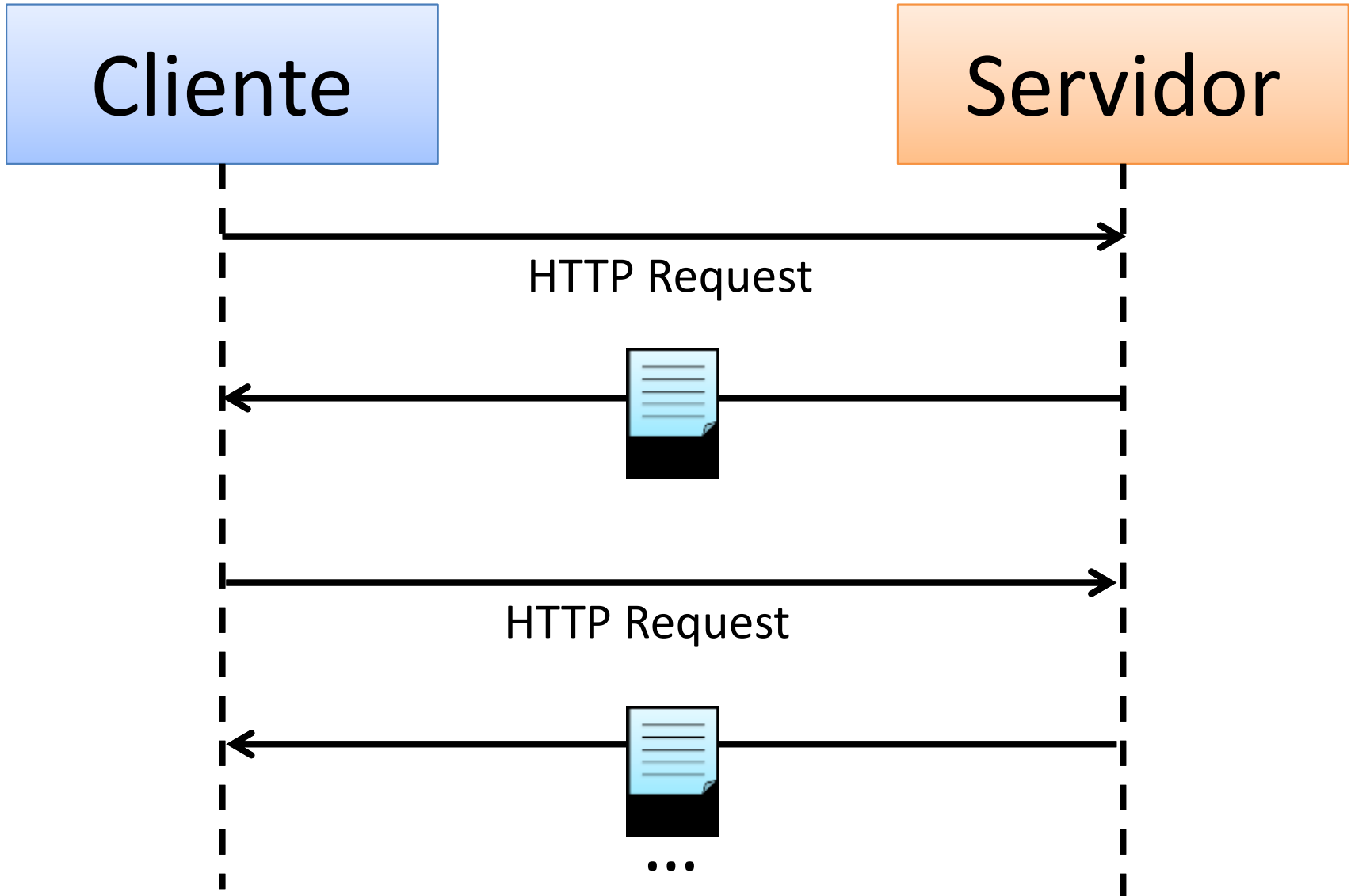


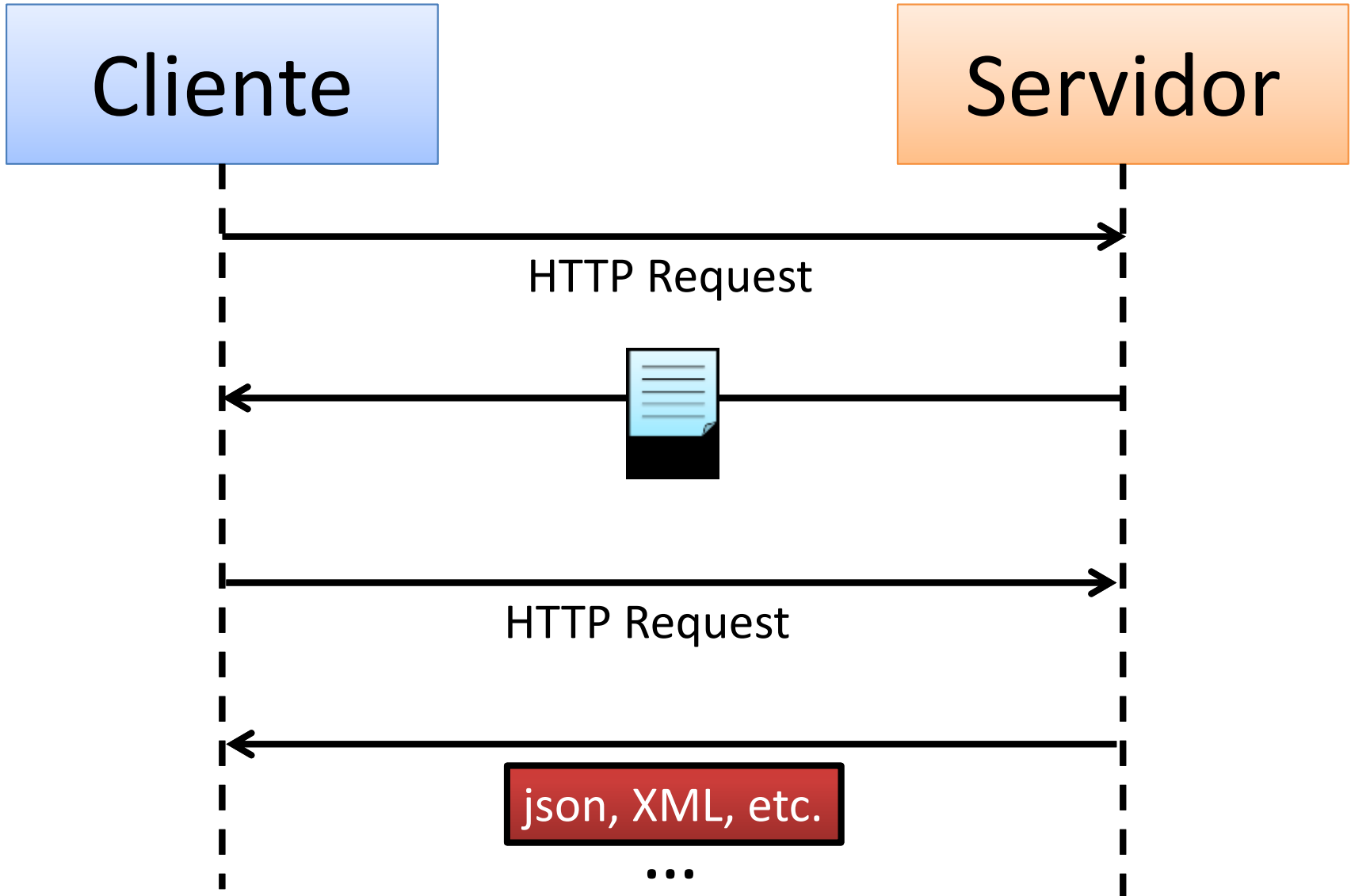
Single-Page Applications & Angular

Jaime A. Pavlich-Mariscal

Multi-Page Applications (MPA)



Single-Page Applications (SPA)



MPA vs SPA

Multi-Page Application (MPA)

- Interacción:
 - Cliente realiza petición
 - Servidor ejecuta petición y envía página como respuesta
- Mayor carga en el servidor para:
 - UI
 - Flujo de interacción con el usuario
- La lógica está en el servidor

Cliente liviano

Single-Page Application (SPA)

- Interacción:
 - Cliente realiza petición
 - Servidor ejecuta petición y envía “datos” como respuesta
- Mayor carga en el cliente para:
 - UI
 - Flujo de interacción con el usuario
- La lógica *debería* concentrarse en el servidor

Cliente pesado

Angular

<https://angular.io/>

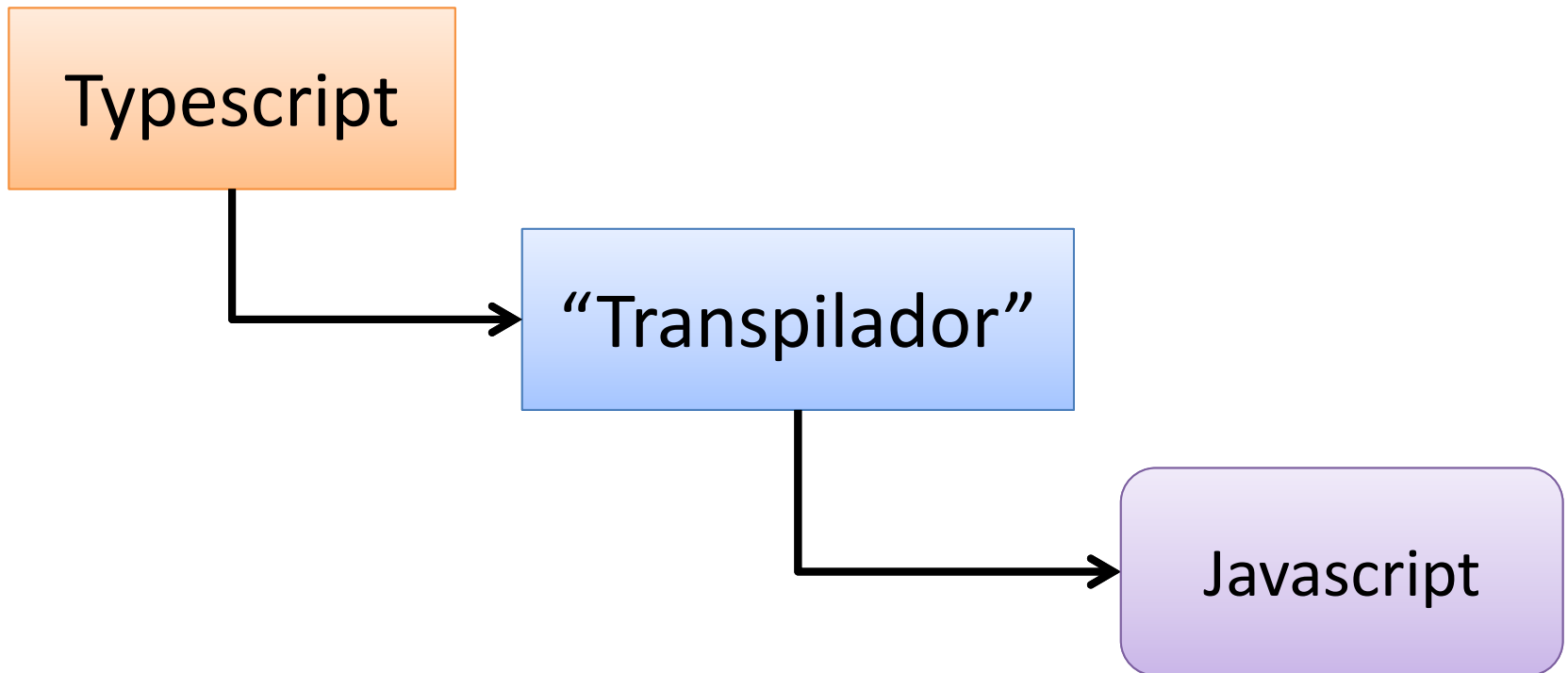
Angular

- Framework basado en Javascript + otros lenguajes (Typescript y otros)
- Creación de SPA
- Creación de UI complejas
- Multiplataforma (Web, móvil y escritorio)
- Desventajas
 - Soporte aún es limitado en las IDE

TypeScript

Typescript

- Superconjunto de Javascript
- Se “transpila” a código Javascript



Principales adiciones/diferencias con Javascript

- Declaración explícita de tipos
- Clases e interfaces
- Tipos de datos genéricos
- Modularización
- Decoradores

Declaración explícita de tipos

```
let variableIncorrecta : boolean = "hola";  
let variableCorrecta : string = "hola";
```

Classes

```
abstract class Animal {  
    nombre: string;  
  
    constructor(nombre: string) {  
        this.nombre = nombre;  
    }  
  
    abstract hacerRuido(): string;  
}
```

```
enum Raza { GOLDEN, SALCHICHA, PASTOR_ALEMAN};  
  
class Perro extends Animal {  
    raza: Raza;  
    constructor(nombre: string, raza: Raza) {  
        super(nombre);  
        this.raza = raza;  
    }  
  
    hacerRuido() {  
        return "guau";  
    }  
}
```

```
let a: Animal = new Perro("Fido", Raza.GOLDEN);  
console.log(a.hacerRuido());
```

Equivalencia estructural de tipos

```
class Animal {  
  nombre: string;  
  id: number;  
}  
  
class Persona {  
  id: number;  
  nombre: string;  
}  
  
let a : Animal = new Animal();  
let p : Persona = a;
```

Interfaces

```
enum Raza { GOLDEN, SALCHICHA, PASTOR_ALEMAN };

interface Animal {
    nombre: string;
    hacerRuido(): string;
}

let a = {
    nombre: "Fido",
    hacerRuido: function() { return "guau"; },
    raza: Raza.GOLDEN
};

function procesar(animal: Animal) {
    console.log(animal.hacerRuido());
}

procesar(a);
```

Tipos de datos genéricos

```
class Pila<T> {  
    private datos: T[] = [];  
    push(dato: T) {  
        this.datos.push(dato);  
    }  
  
    pop() : T {  
        return this.datos.pop();  
    }  
  
    toString() {  
        return JSON.stringify(this.datos);  
    }  
}
```

```
let pila = new Pila<number>();
```

```
pila.push(15);
```

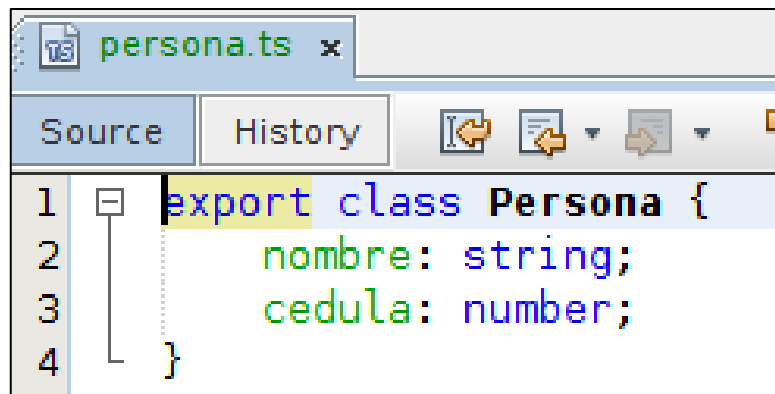
```
pila.push(20);
```

```
console.log("" + pila);
```

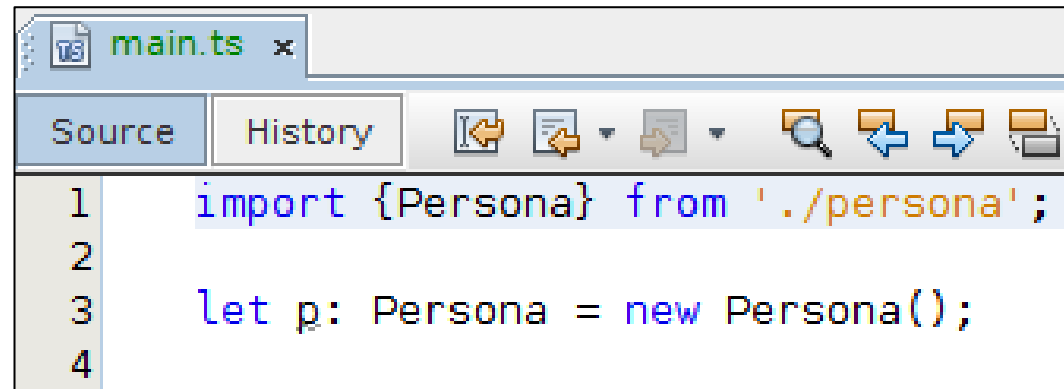
```
pila.push("hola");
```

Modularización

- Cada archivo encapsula clases, variables y funciones
- `import` vs `export`



```
1 export class Persona {  
2     nombre: string;  
3     cedula: number;  
4 }
```



```
1 import {Persona} from './persona';  
2  
3 let p: Persona = new Persona();  
4
```

Decoradores

- Funciones que se aplican a elementos declarados (clases, funciones, atributos, etc.)

```
function Logged<TFunction extends Function>(target: TFunction): TFunction {  
    let newConstructor = function (...args : any[]) {  
        target.apply(this, args);  
        console.log("Object created: " + JSON.stringify(this));  
    };  
  
    newConstructor.prototype = Object.create(target.prototype);  
  
    return <any> newConstructor;  
}
```


Decoradores

```
@logged
class Estudiante {
    nombre: string;
    cedula: number;

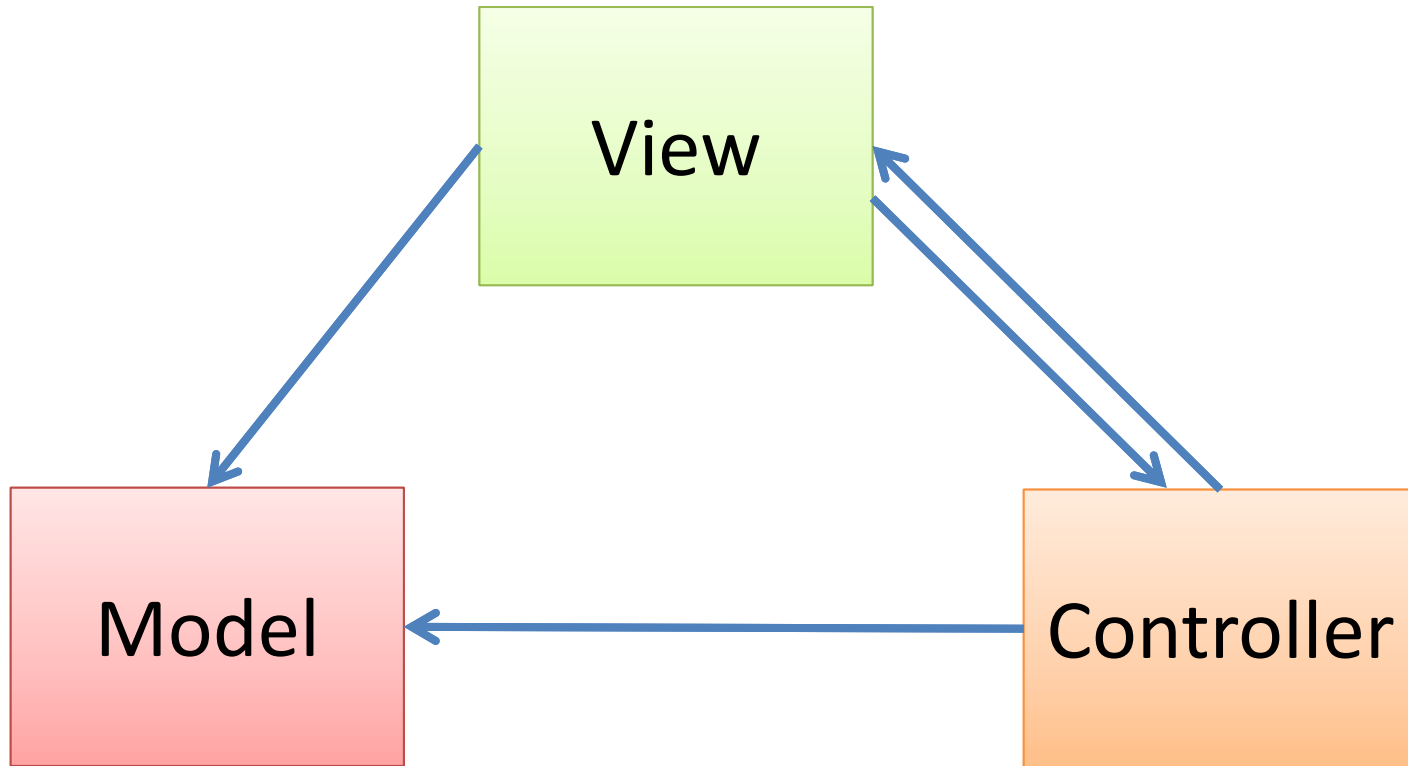
    constructor(nombre: string, cedula: number) {
        this.nombre = nombre;
        this.cedula = cedula;
    }
}

new Estudiante("Alice", 123);
new Estudiante("Bob", 456);
new Estudiante("Charlie", 789);
```

```
Object created: {"nombre":"Alice","cedula":123} (14:31:06:995 | null)
  at http://localhost:8383/TypescriptExample/app/main.ts!transpiled:56
Object created: {"nombre":"Bob","cedula":456} (14:31:06:996 | null)
  at http://localhost:8383/TypescriptExample/app/main.ts!transpiled:56
Object created: {"nombre":"Charlie","cedula":789} (14:31:06:996 | null)
  at http://localhost:8383/TypescriptExample/app/main.ts!transpiled:56
```

Angular + Typescript

Model-View-Controller



Krasner, Glenn E., and Stephen T. Pope. "A description of the model-view-controller user interface paradigm in the smalltalk-80 system." *Journal of object oriented programming* 1.3 (1988): 26-49.

Principales conceptos

HTML

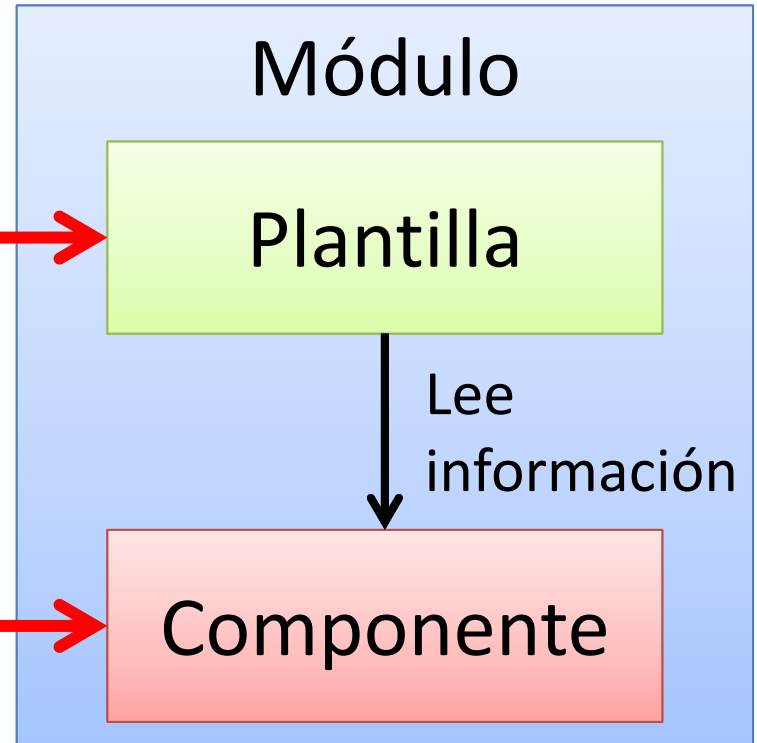
+

extensiones **Angular**

Typescript

+

librerías **Angular**



Componente

- Define un “trozo” de UI que se puede usar dentro de una página web
- Autocontenido
- Reutilizable

Componente

Importa anotación
requerida

Declara la clase
AppComponent
como un componente

Etiqueta que se usará
en HTML para “llamar”
al componente

```
import {Component} from '@angular/core'

@Component({
  moduleId: module.id,
  selector: 'hello-world',
  templateUrl: 'hello-world.component.html'
})
export class HelloWorldComponent {
  helloWorldData = {
    greeting: "Hello world",
    nextSentence: "déjà vu"
  }
}
```

El cuerpo de la clase contendrá la
información a desplegar en la página

Plantilla del
componente

Plantilla

- Indica cómo se va a mostrar en pantalla la información del componente
- Utiliza HTML + extensiones de Angular
 - Puede referenciar elementos del componente correspondiente

Plantilla

```
<div>
  {{ helloWorldData.greeting }}!
</div>
<div>
  {{ helloWorldData.nextSentence }}
</div>
```

```
import {Component} from '@angular/core'

@Component({
  moduleId: module.id,
  selector: 'hello-world',
  templateUrl: 'hello-world.component.html'
})
export class HelloWorldComponent {
  helloWorldData = {
    greeting: "Hello world",
    nextSentence: "déjà vu"
  }
}
```



Módulo

- Agrupa varios componentes (y sus plantillas) en una misma unidad dentro del programa

Módulos externos
requeridos para
funcionar

Módulo

```
import {NgModule} from '@angular/core';
import {BrowserModule} from '@angular/platform-browser';
import {HelloWorldComponent} from './hello-world.component'

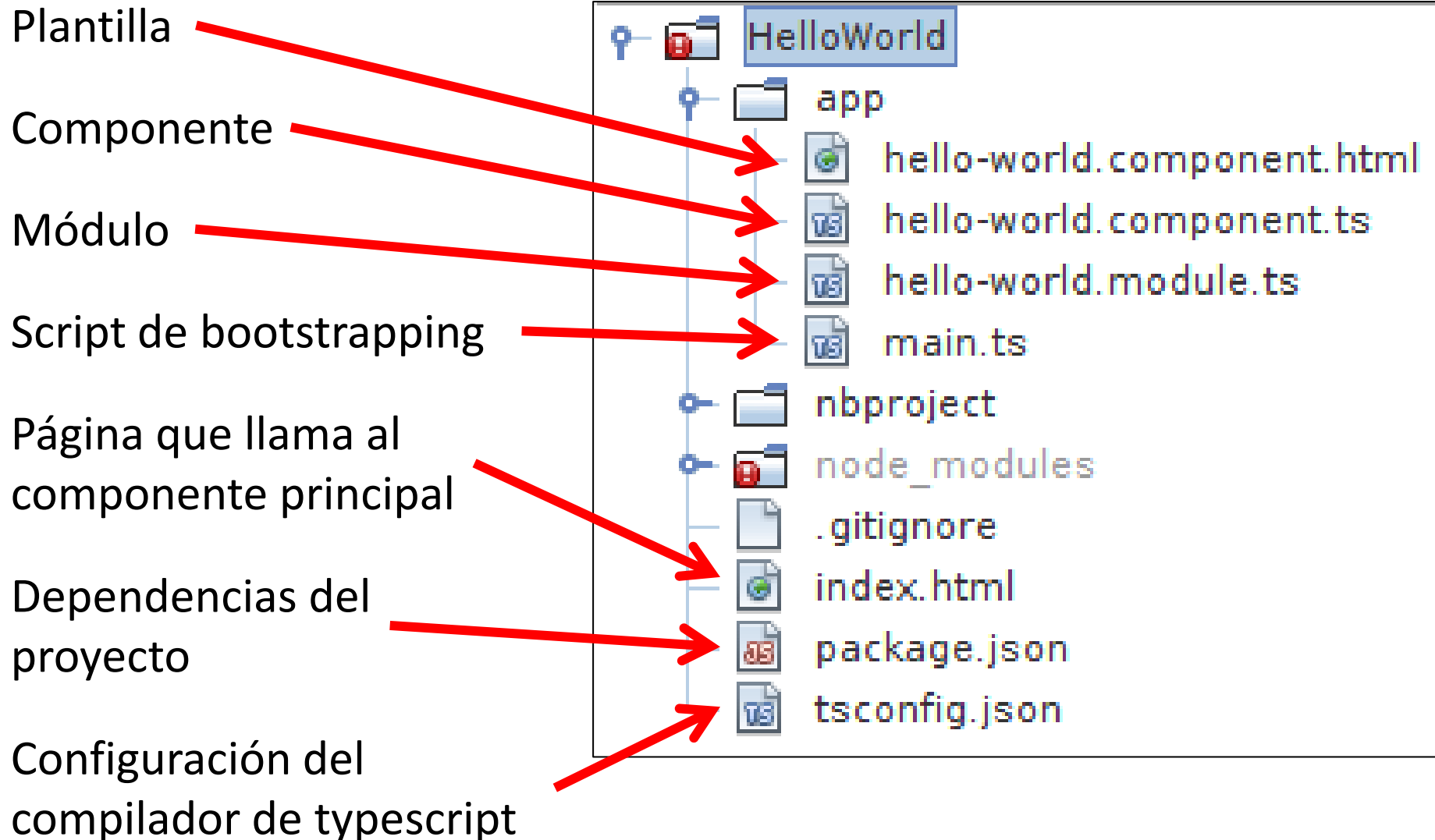
@NgModule({
  imports: [
    BrowserModule
  ],
  declarations: [
    HelloWorldComponent
  ],
  bootstrap: [HelloWorldComponent]
})
export class HelloWorldModule {}
```

Componentes que
pertenecen a este
módulo

Componente principal del
módulo que inicia la
ejecución de la aplicación

Estructura del código fuente

Estructura del código fuente



index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello Angular</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="forms.css">

    <!-- Polyfills for older browsers -->
    <script src="https://unpkg.com/core-js/client/shim.min.js"></script>

    <script src="https://unpkg.com/zone.js@0.7.4?main=browser"></script>
    <script src="https://unpkg.com/reflect-metadata@0.1.8"></script>
    <script src="https://unpkg.com/systemjs@0.19.39/dist/system.src.js"></script>

    <link rel="stylesheet" href="https://unpkg.com/bootstrap@3.3.7/dist/css/boot

    <script src="https://cdn.rawgit.com/angular/angular.io/b3c65a9/public/docs/_
    <script>
      System.import('app').catch(function (err) {
        console.error(err);
      });
    </script>
  </head>

  <body>
    <hello-world>Loading AppComponent content here ...</hello-world>
  </body>

</html>
```

El componente va aquí

Script de bootstrapping

- Compila la aplicación
- Crea una instancia del componente principal del módulo principal
- La inserta en la anotación correspondiente de `index.html`

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  
import { HelloWorldModule }           from './hello-world.module';  
  
platformBrowserDynamic().bootstrapModule(HelloWorldModule);
```

Mostrar datos en una página

<https://angular.io/docs/ts/latest/guide/template-syntax.html>

Mostrar un valor atómico

```
<div>
  {{ helloWorldData.greeting }}!
</div>
<div>
  {{ helloWorldData.nextSentence }}
</div>
```

```
import {Component} from '@angular/core'

@Component({
  moduleId: module.id,
  selector: 'hello-world',
  templateUrl: 'hello-world.component.html'
})
export class HelloWorldComponent {
  helloWorldData = {
    greeting: "Hello world",
    nextSentence: "déjà vu"
  }
}
```


Mostrar colecciones

```
<div>
  Hello:
  <ul>
    <li *ngFor="let person of persons">{{person}}</li>
  </ul>
</div>
```

```
import {Component} from '@angular/core'

@Component({
  moduleId: module.id,
  selector: 'hello-world',
  templateUrl: 'multi-hello-world.component.html'
})
export class MultiHelloWorldComponent {
  persons = ["Alice", "Bob", "Carlos", "Dora"];
}
```

Hello:

- Alice
- Bob
- Carlos
- Dora

Ejercicio: Mostrar datos en una página

- Crear una aplicación en Angular que muestre:
 - Un valor atómico
 - Una colección de elementos
- Utilice Angular-CLI para crear el proyecto y sus componentes
 - <https://cli.angular.io/>

Manejo de eventos

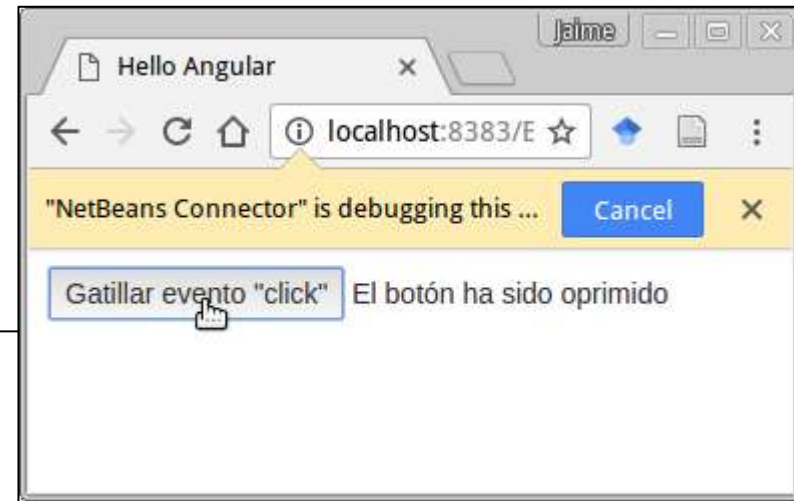
Detección de eventos

```
<button (click)="onClickButton()">
  Gatillar evento "click"
</button>
{{ message }}
```

```
import {Component} from '@angular/core'

@Component({
  moduleId: module.id,
  selector: 'events-example',
  templateUrl: 'events-example.component.html'
})
export class EventsExampleComponent {
  message = "";

  onClickButton() {
    this.message = "El botón ha sido oprimido";
  }
}
```



Acceso a datos del DOM

```
<button #miBoton (click)="onClickButton(miBoton.innerHTML)">
  Gatillar evento "click"
</button>
{{ message }}
```

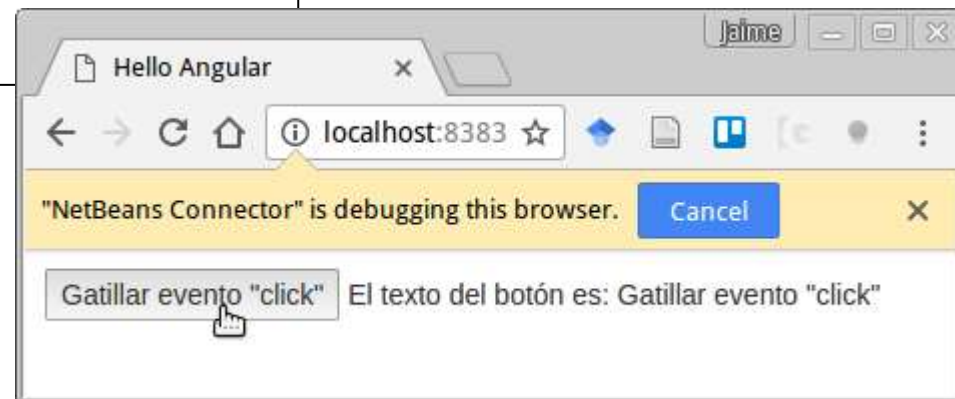
```
import {Component} from '@angular/core'

@Component({
  moduleId: module.id,
  selector: 'events-example',
  templateUrl: 'events-example.component.html'
})
export class EventsExampleComponent {
  message = "";

  onClickButton(texto: string) {
    this.message = `El texto del botón es: ${texto}`;
  }
}
```

Template reference variable

<https://angular.io/guide/template-syntax#ref-vars>



Formularios

<https://angular.io/docs/ts/latest/guide/forms.html>

Agregar módulo FormsModule

```
import {NgModule} from '@angular/core';  
import {BrowserModule} from '@angular/platform-browser';  
import {FormExampleComponent} from './form-example.component';  
import {FormsModule} from '@angular/forms';
```

```
@NgModule({  
  imports: [  
    BrowserModule, FormsModule  
  ],  
  declarations: [  
    FormExampleComponent  
  ],  
  bootstrap: [FormExampleComponent]  
})  
export class FormExampleModule {}
```

```
import {Component} from '@angular/core'
```

```
@Component({
  moduleId: module.id,
  selector: 'form-example',
  templateUrl: 'form-example.component.html'
})
export class FormExampleComponent {
  estudiante = {nombre: "", cedula: 0};
  submitted = false;

  procesarEstudiante() {
    this.submitted = true;
  }
}
```

Ejercicio: Un formulario simple

```
<div [hidden]="submitted">
  <form (ngSubmit)="procesarEstudiante()">
    <label for="txtNombre">Nombre: </label>
    <input type="text" id="txtNombre"
      name="nombre" [(ngModel)]="estudiante.nombre">
    <label for="txtCedula">Nombre: </label>
    <input type="text" id="txtCedula"
      name="cedula" [(ngModel)]="estudiante.cedula">
    <button type="submit">Enviar</button>
  </form>
</div>

<div [hidden]="!submitted">
  Estudiante:
  <div>Nombre: {{estudiante.nombre}}</div>
  <div>Cedula: {{estudiante.cedula}}</div>
</div>
```

Nombre: Pedro

Nombre: 12345

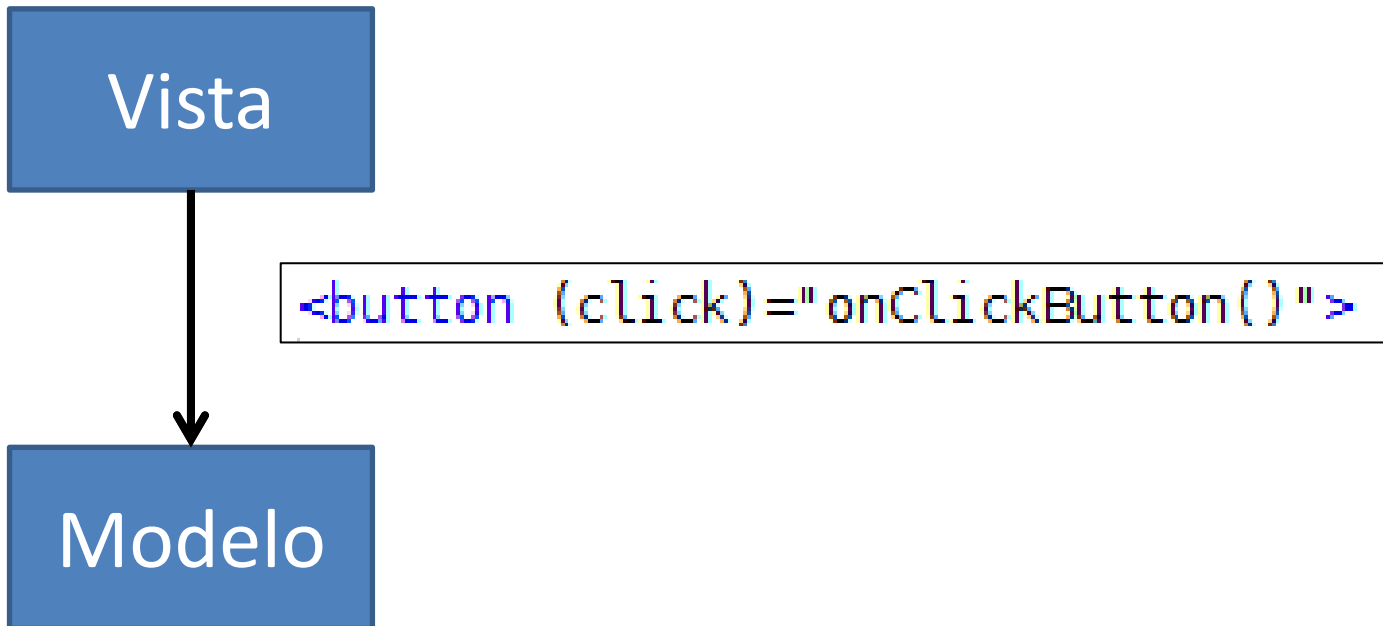
Enviar



Estudiante:
Nombre: Pedro
Cedula: 12345

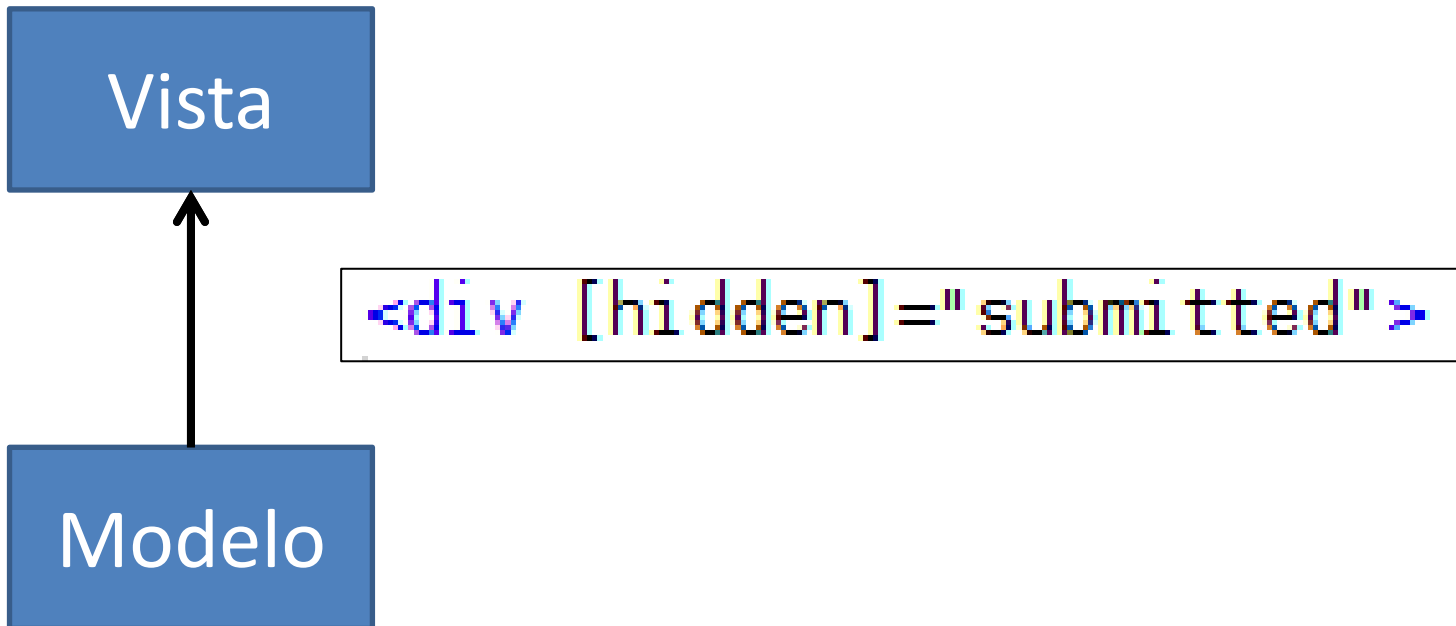
Flujo de información

Vista → Modelo



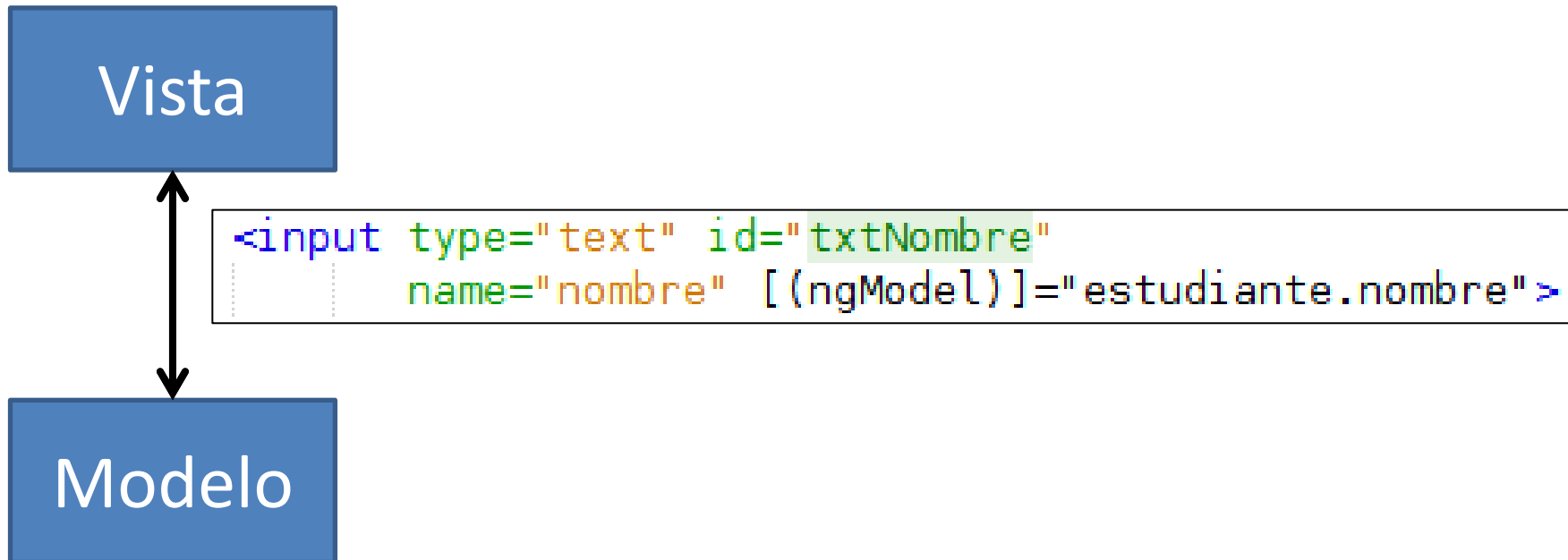
Flujo de información

Modelo → Vista



Flujo de información

Vista \leftrightarrow Modelo



- Ejercicio:
 - Duplique todo el código dentro de `<form>...</form>`
 - Observe cómo se actualizan los campos del formulario cuando son modificados

Ejercicio

- Usando Angular:
 - Agregue a su página web un listado de mensajes (similar a los de redes sociales)
 - Se pueden agregar nuevos mensajes usando un formulario
 - Todo se maneja a nivel del cliente. No hay que guardar nada en un servidor