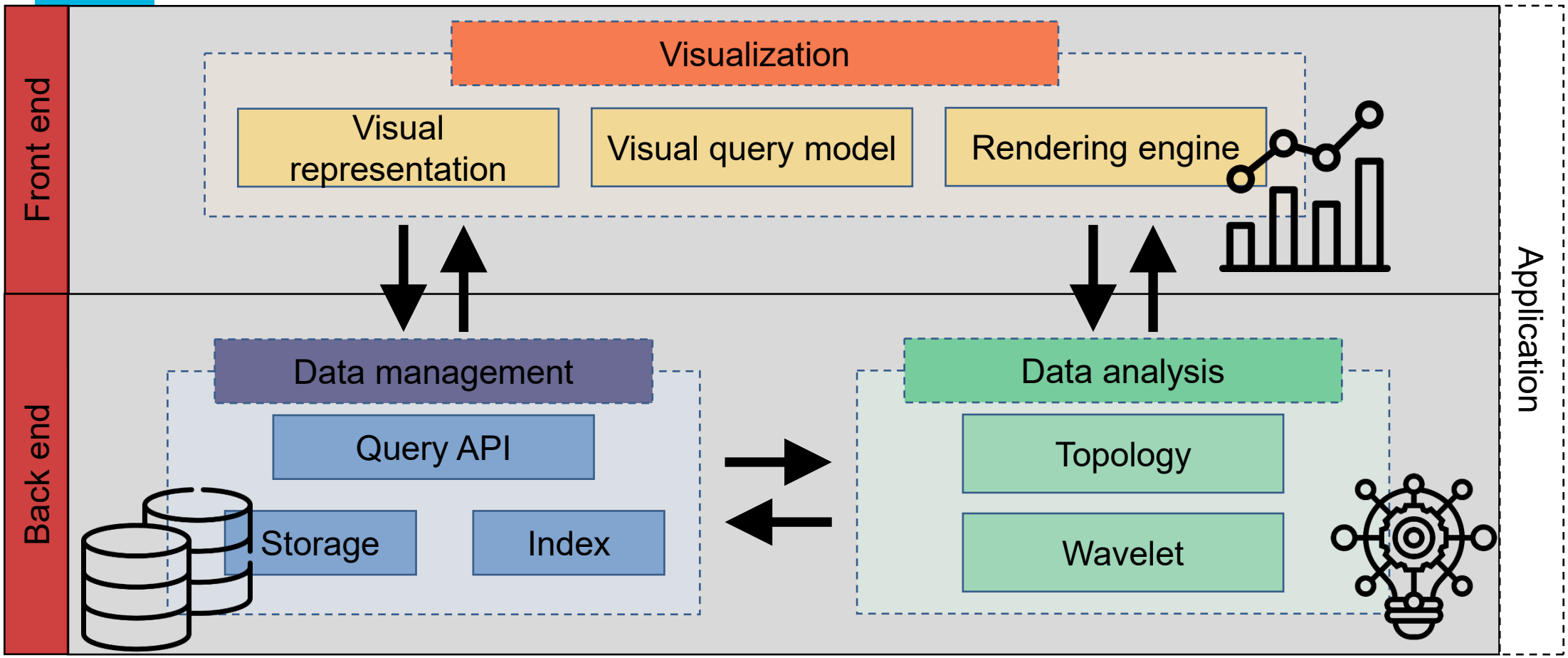
Building blocks: TypeScript and Angular

CS594: Big Data Visualization & Analytics

Fabio Miranda

https://fmiranda.me

Big data visualization system

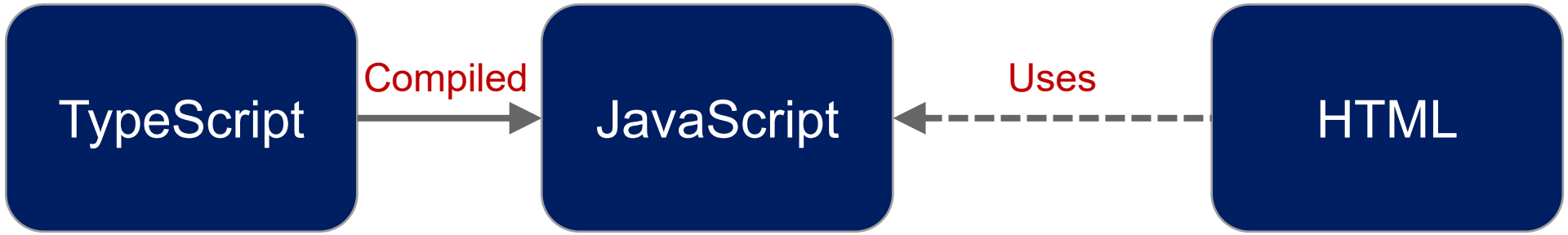


Big data visualization system

- Why separate front-end and back-end development?
  - Separation of concerns between presentation layer (front end) and data layer (back end).
  - Easily mapped to a client-server model.
    - Client: front end
    - Server: back end
  - Easy deployment.

Angular

- Development platform, built on TypeScript (superset of JavaScript).
- Cross-platform component-based framework for building scalable web applications.
- Well-integrated libraries covering a wide variety of features (e.g., client-server communication, DOM, etc.)



TypeScript

- JavaScript-like language: "*JavaScript with Syntax for Types*"
  - Types
  - Classes
  - Imports
- Major change over JavaScript: type checking

```
var name: string;
var age: number;
var address: any;

function hello(name: string): string {
    return 'Hello ' + name;
}
```

TypeScript

Classes may have properties, methods and constructors.

Class inheritance through the `extends` keyword.

```
class Person extends Creature {
    firstName: string;
    belongings: string[];
    age: number;
    constructor(first: string, age: number) {
        super();
        this.firstName = first;
        this.age = age;
    }
    hello() {
        console.log('Hello ' + this.firstName);
    }
    setAge(age: number) {
        this.age = age;
    }
    getAge(): number {
        return this.age;
    }
}
var p: Person = new Person('Arthur', 'Dent', 30);
p.setAge(31);
```

TypeScript: arrow functions

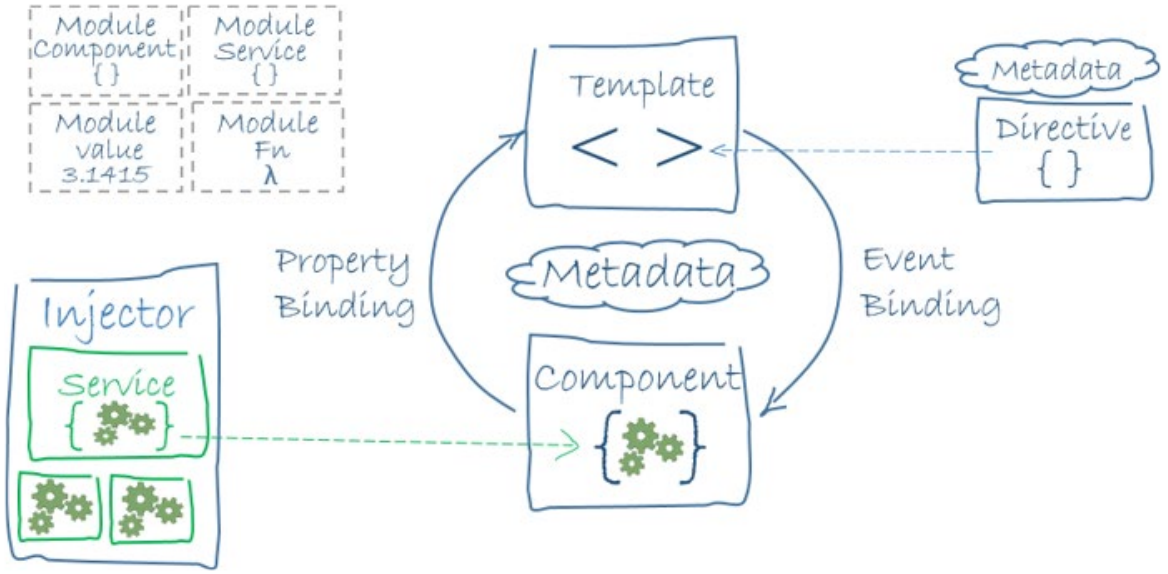
- Arrow notations are used for anonymous functions.
- Drop the need to use the `function` keyword.
- Arrow functions share the same `this` as the surrounding code.

```
printBelongings() {
    var that = this;
    this.belongings.forEach(function(b: string) {
        console.log(that.firstName+' has a '+b);
    });
}
```

```
printBelongings() {
    this.belongings.forEach((b) => {
        console.log(this.firstName+' has a '+b);
    });
}
```

Angular

- An Angular application is a tree of components.
- Top-level component: application itself.
- Components:
  - Composable
  - Reusable
  - Hierarchical
- Angular ≠ AngularJS.



Angular

- Pre-requisites:
  - Node.js: JavaScript runtime environment.
  - npm: package manager for JavaScript.
  - Angular CLI: create projects, generate applications and library code, testing, bundling, and deployment.

```
user@DESKTOP MINGW64 ~/
$ conda install nodejs
```

```
user@DESKTOP MINGW64 ~/
$ npm install -g @angular/cli
```

Angular: creating an initial application

- Create a new application project using angular.
- A project is the set of files that comprise an application or library.

```
user@DESKTOP MINGW64 ~/
$ ng new example
```

ng stands for A**ng**ular!

Angular: creating an initial application

```
user@DESKTOP MINGW64 ~/example
$ ls -lah
total 710K
drwxr-xr-x 1 user 197121    0 Aug 29 21:48 ./
drwxr-xr-x 1 user 197121    0 Aug 29 21:42 ../
-rw-r--r-- 1 user 197121  703 Aug 29 21:42 .browserslistrc
-rw-r--r-- 1 user 197121  274 Aug 29 21:42 .editorconfig
drwxr-xr-x 1 user 197121    0 Aug 29 21:48 .git/
-rw-r--r-- 1 user 197121  631 Aug 29 21:42 .gitignore
-rw-r--r-- 1 user 197121 3.5K Aug 29 21:42 angular.json
drwxr-xr-x 1 user 197121    0 Aug 29 21:42 e2e/
-rw-r--r-- 1 user 197121 1.4K Aug 29 21:42 karma.conf.js
drwxr-xr-x 1 user 197121    0 Aug 29 21:48 node_modules/
-rw-r--r-- 1 user 197121 1.2K Aug 29 21:42 package.json
-rw-r--r-- 1 user 197121 510K Aug 29 21:48 package-lock.json
-rw-r--r-- 1 user 197121 1017 Aug 29 21:42 README.md
drwxr-xr-x 1 user 197121    0 Aug 29 21:42 src/
-rw-r--r-- 1 user 197121  287 Aug 29 21:42 tsconfig.app.json
-rw-r--r-- 1 user 197121  538 Aug 29 21:42 tsconfig.json
-rw-r--r-- 1 user 197121  333 Aug 29 21:42 tsconfig.spec.json
-rw-r--r-- 1 user 197121 3.2K Aug 29 21:42 tslint.json
```

e2e: automated testing

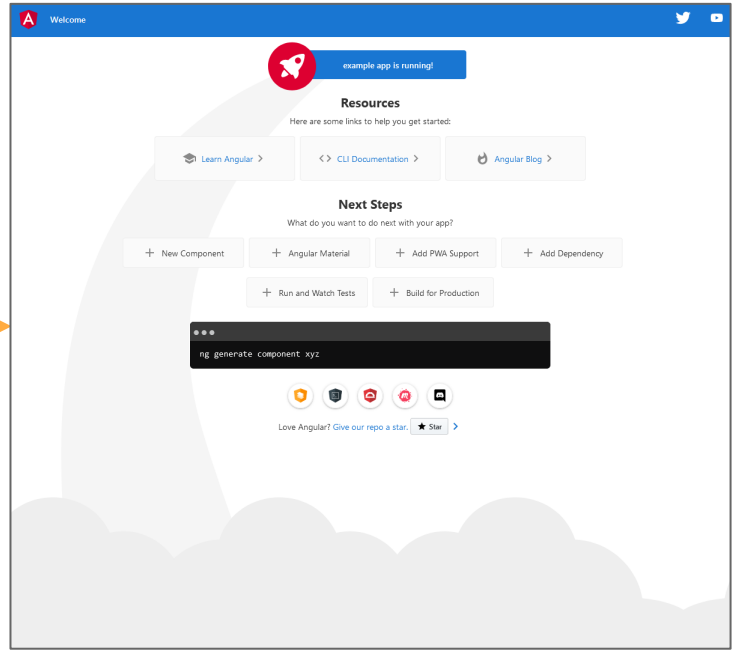
node_modules: development dependencies and dependencies

package.json: information about the libraries added and user in the project, with specified version installed

src: actual project source code, with components, services, etc.

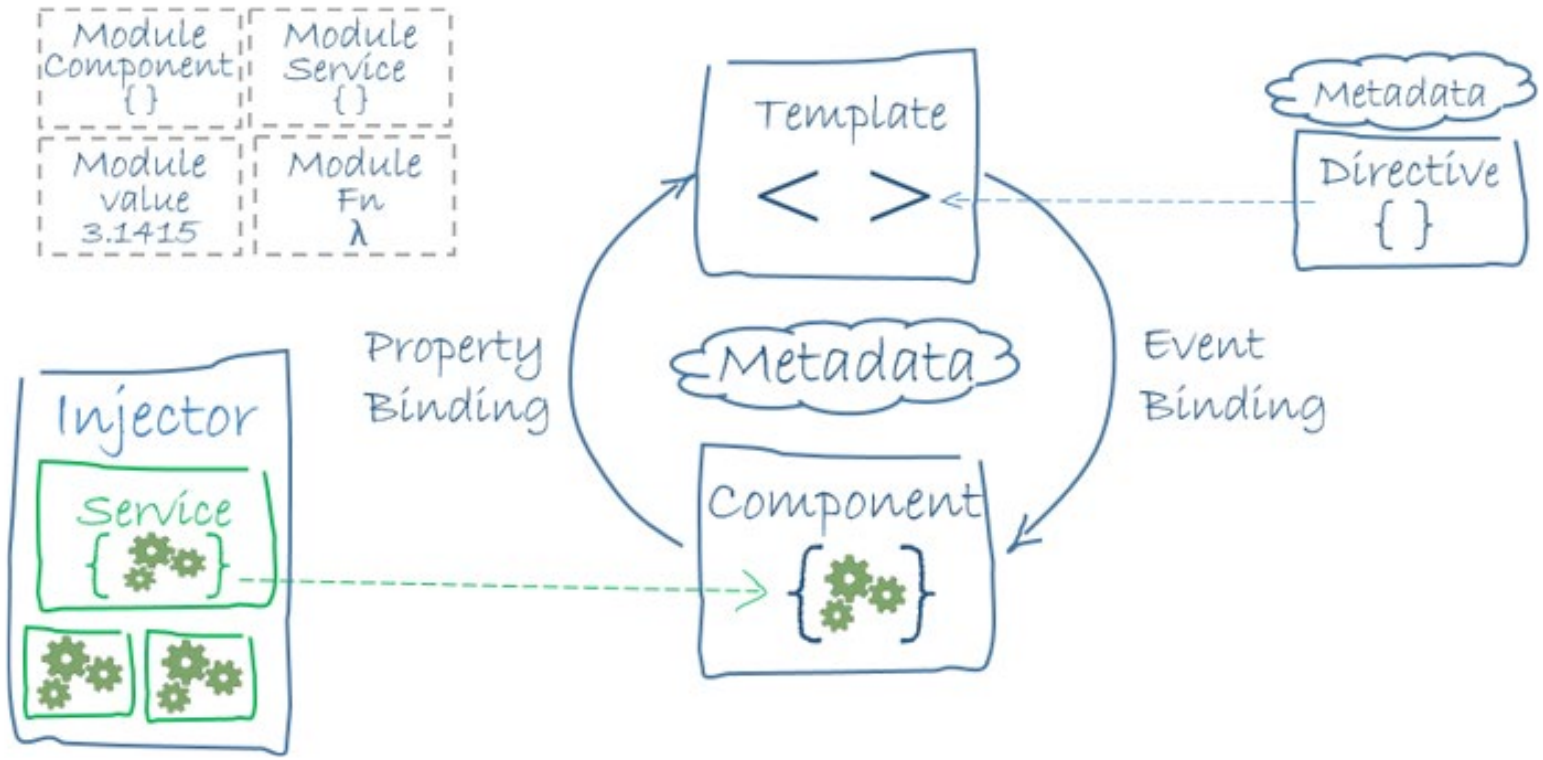
Angular: serving the application

```
user@DESKTOP MINGW64 ~/example
$ ng serve --open
```



Angular architecture

- Modules
- Components
- Templates
- Metadata
- Data binding
- Directives
- Services



Angular architecture

- Creating components:

```
user@DESKTOP MINGW64 ~/example
$ ng generate component map
```

```
user@DESKTOP MINGW64 ~/example
$ ng generate service wrapper
```

Angular architecture: modules

- Angular applications are modular:
  - An application defines a set of modules.
  - Every angular module is a class with `@NgModule` decorator.
- Every angular application has at least one module: root module.
- A module encapsulates a set of components dedicated to an application domain, a workflow, or closely related set of capabilities.
- A module can import functionalities from other modules and allow their own functionalities to be exported.

Angular architecture: modules

- Module properties:
  - Declaration: components, directives, and pipes that belong to the module.
  - Exports: subset of declarations visible and usable by other modules.
  - Imports: external modules.
  - Providers: creators of services.
  - Bootstrap: main application view, the root component.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
@NgModule({
  declarations: [AppComponent, MapComponent],
  imports: [BrowserModule],
  exports: [AppComponent],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Angular architecture: components

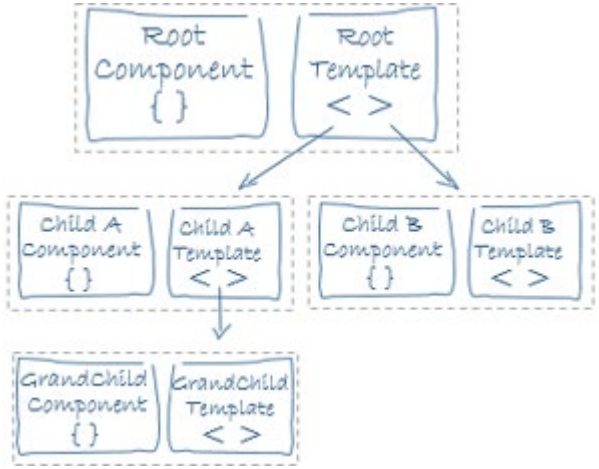
- Components are the main building blocks of Angular applications.
- Each component consist of:
  - HTML template that declares what renders on the page.
  - TypeScript class that defines behavior.
  - CSS selector that defines how the component is used in a template.
  - CSS styles applied to the template.

Angular architecture: components

```
import * as d3 from 'd3’;

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

export class MapComponent implements AfterViewInit {
  map: Map;
  curCity = 'chi';
  curDate = 'jun-21';

  constructor(private router: Router, private location: Location) { }
  ngAfterViewInit() {

  }
  public onCityChange(newValue) {
    this.curCity = newValue;
  }
  public onDayChange(newValue) {
    this.curDate = newValue;
  }
}
```

Angular architecture: templates

- A snippet of the HTML code of a component: tells Angular how to render the component.
  - A component's view is defined with its template.
- Uses Angular's template syntax, with custom elements.



Angular architecture: templates

```
<div>
  <mat-card class="card">
      <div><b>{{example}}</b></div>
      <div>The map shows the accumulated shadow on three different days of the year:</div>
      <div>Jun. 21 (summer solstice), Sep. 22 (autumnal equinox), Dec. 21 (winter solstice)</div>
  </mat-card>
  <div class="menuCity" id="city">
      <mat-button-toggle-group [value]="curCity">
          <mat-button-toggle value="nyc" (change)="onCityChange($event.value)" enabled>NYC</mat-button-toggle>
          <mat-button-toggle value="chi" (change)="onCityChange($event.value)">Chicago</mat-button-toggle>
      </mat-button-toggle-group>
  </div>
  <div class="menuDay" id="day">
      <mat-button-toggle-group [value]="curDate">
          <mat-button-toggle value="jun-21" (change)="onDayChange($event.value)">Summer</mat-button-toggle>
          <mat-button-toggle value="sep-22" (change)="onDayChange($event.value)">Spring/Fall</mat-button-toggle>
          <mat-button-toggle value="dec-21" (change)="onDayChange($event.value)">Winter</mat-button-toggle>
      </mat-button-toggle-group>
  </div>
</div>
<div class="map" id="map"></div>
<div id="popup" class="ol-popup">
  <div id="popup-content"></div>
</div>
```

Angular architecture: data binding

- Mechanism for coordinating parts of a template with parts of a component.
- Four main forms:
  - Interpolation: `{{example}}`
    - Incorporate dynamic string values into HTML templates.
  - Property binding: `[example]`
    - Set values for properties of HTML elements.
  - Event binding: `(click)`
    - Listen for and respond to user actions (keystrokes, mouse movements, clicks, touches).
  - Two-way data binding: `[(ngModel)]`
    - Gives components in application a way to share data, using two-way binding to listen for events and update values simultaneously.

Angular architecture: data binding

```
<div>
  <mat-card class="card">
      <div><b>{{example}}</b></div>
      <div>The map shows the accumulated shadow on three different days of the year:</div>
      <div>Jun. 21 (summer solstice), Sep. 22 (autumnal equinox), Dec. 21 (winter solstice)</div>
  </mat-card>
  <div class="menuCity" id="city">
      <mat-button-toggle-group [value]="curCity">
          <mat-button-toggle value="nyc" (change)="onCityChange($event.value)" enabled>NYC</mat-button-toggle>
          <mat-button-toggle value="chi" (change)="onCityChange($event.value)">Chicago</mat-button-toggle>
      </mat-button-toggle-group>
  </div>
  <div class="menuDay" id="day">
      <mat-button-toggle-group [value]="curDate">
          <mat-button-toggle value="jun-21" (change)="onDayChange($event.value)">Summer</mat-button-toggle>
          <mat-button-toggle value="sep-22" (change)="onDayChange($event.value)">Spring/Fall</mat-button-toggle>
          <mat-button-toggle value="dec-21" (change)="onDayChange($event.value)">Winter</mat-button-toggle>
      </mat-button-toggle-group>
  </div>
</div>
<div class="map" id="map"></div>
<div id="popup" class="ol-popup">
  <div id="popup-content"></div>
</div>
```

Interpolation

Property binding

Event binding

Angular architecture: services

- Components shouldn’t fetch or save data directly.
- Components should focus on presenting data, and delegate data access to a service.
- Dependency injection: provide components with services they need.

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

@Injectable({
  providedIn: 'root',
})
export class DataService {
  constructor() { }
}
```