
ACCELERATING THROUGH
ANGULAR 2

Level 3

Splitting to Two Components

Section 1

Splitting Things Into Pieces

We've been developing Angular 2 in one single file: main.ts**. This isn't going to scale, so let's split things up into pieces and get organized.**

main.ts

We will take our single file and split it into three.

main.ts
Where we'll bootstrap our app, loading our first component.

app.component.ts
This component contains our page header.

car-parts.component.ts
This contains our list of car parts.

After this, we will have two components instead of one.

Trimming Down Our main.ts

This is where we bootstrap our app, loading our first component.

main.ts TypeScript

```
import { NgModule, Component } from ‘@angular/core’;
import { BrowserModule } from '@angular/platform-browser';
import { platformBrowserDynamic } from ‘@angular/platform-browser-dynamic';

@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    ...`
})
class AppComponent {
  title = 'Ultra Racing';
  carParts = [...];
  totalCarParts() { ... };
 }
...
```

There’s a bunch of code we need to move elsewhere.

Creating Our app.component.ts

We move most of our code into app.component.ts**.**

main.ts TypeScript

```
 ...


@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule ],
  bootstrap: [ AppComponent ]
})
class AppModule { }
```

app.component.ts TypeScript

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

@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    ...`
})
class AppComponent {
  title = 'Ultra Racing';
  carParts = [...];
  totalCarParts() { ... };
}
```

However, this code is broken. We're bootstrapping our AppComponent, but we don't have access to this class.

How do we get access to a class from another file?

Exporting & Importing

We need to use the ES2015 feature of exporting and importing.

main.ts TypeScript

```
...
import { AppComponent } from './app.component';

@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule ],
  bootstrap: [ AppComponent ]
})
class AppModule { }
```

app.component.ts TypeScript

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

@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    ...`
})
export class AppComponent {
  title = 'Ultra Racing';
  carParts = [...];
  totalCarParts() { ... };
}
```

First, we export the class we want to import.

Then, we import this class into our `main.ts`.

The names must be the same in each file.

One More File to Create

We need to create a car-parts.component.ts**.**

app

main.ts
Where we'll bootstrap our app, loading our first component.

app.component.ts
This component contains our page header.

car-parts.component.ts
This contains our list of car parts.

We need to split our `AppComponent` *into two components.*

Ultra Racing

There are 9 total parts in stock.

- **SUPER TIRES**

  These tires are the very best

  €4.99

  5 in Stock

- **REINFORCED SHOCKS**

  Shocks made from kryptonite

  €9.99

  4 in Stock

- **PADDED SEATS**

  Super soft seats for a smooth ride

  €24.99

  Out of Stock

Splitting Out Our Components

We need to remove the car parts-specific code from app.component.ts**.**

app.component.ts TypeScript

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

@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <p>There are {{totalCarParts()}} total parts in stock.</p>
    <ul>...</ul>`
})
export class AppComponent {
  title = 'Ultra Racing';
  carParts = [...];
  totalCarParts() { ... };
}
```

Splitting Out Our Car Parts Component

car-parts.component.ts TypeScript

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

@Component({
  selector: 'car-parts',
  template: `
    <p>There are {{totalCarParts()}} total parts in stock.</p>
    <ul>...</ul>`
})
export class CarPartsComponent {
  carParts = [...];
  totalCarParts() { ... };
}
```

Notice our new selector!

How do we tell our application about this component so it can be used?

Importing our Car Parts Component

main.ts TypeScript

```typescript
import { AppComponent } from ‘./app.component';
import { CarPartsComponent } from './car-parts.component';

@NgModule({
  declarations: [
    AppComponent,
    CarPartsComponent
  ],
  imports: [ BrowserModule ],
  bootstrap: [ AppComponent ]
})
class AppModule { }
```

car-parts.component.ts TypeScript

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

@Component({
  selector: 'car-parts',
  template: `...`
})
export class CarPartsComponent {
  carParts = [...];
  totalCarParts() { ... };
}
```

Two things we need to do inside our `main.ts` file to make it work:

- Import our new component to *main.ts*
- Add CarPartsComponent to our module declarations array so it can be used through the rest of our application.

Using Our New Component

app.component.ts — TypeScript

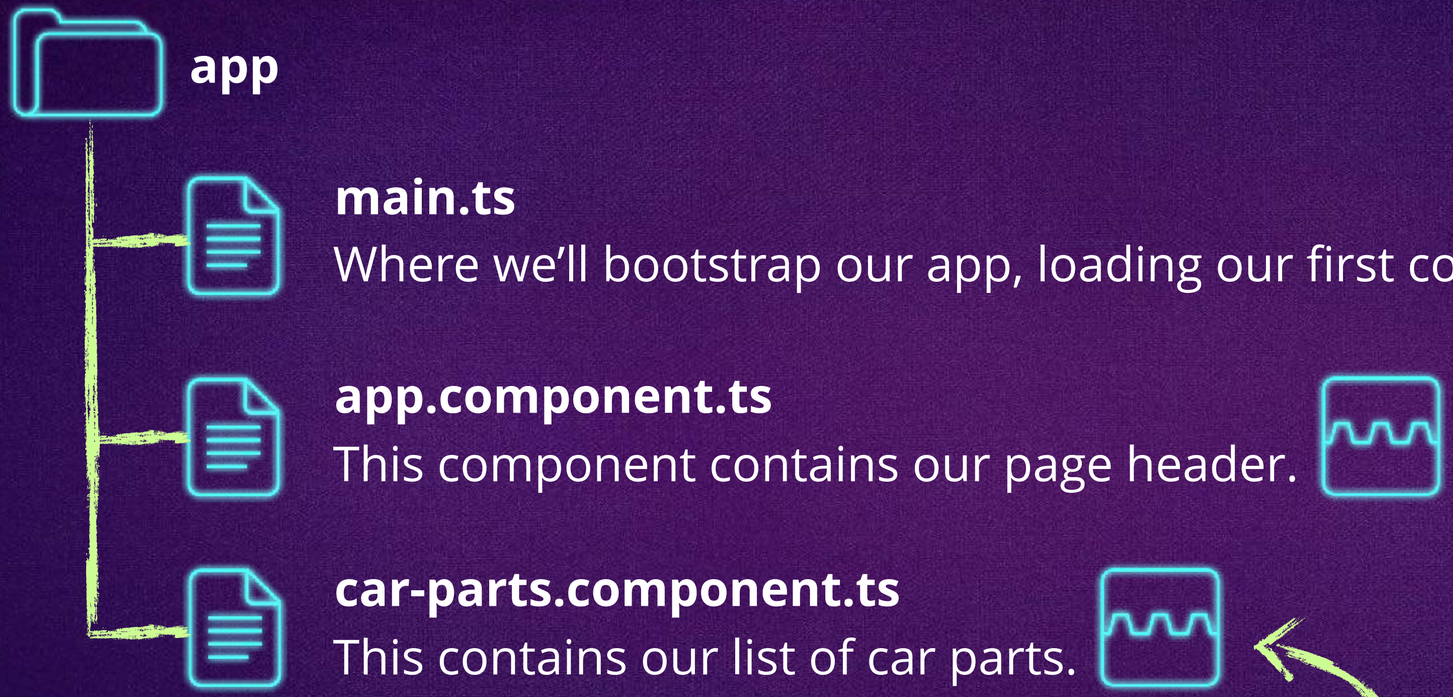
```ts
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>
    <car-parts></car-parts>`
})
export class AppComponent {
  title = 'Ultra Racing';
}
```

Our new selector

Our app is rendering the CarPartsComponent by using the <car-parts> selector.

We've Separated Our Concerns!



And we've created our first reusable component.

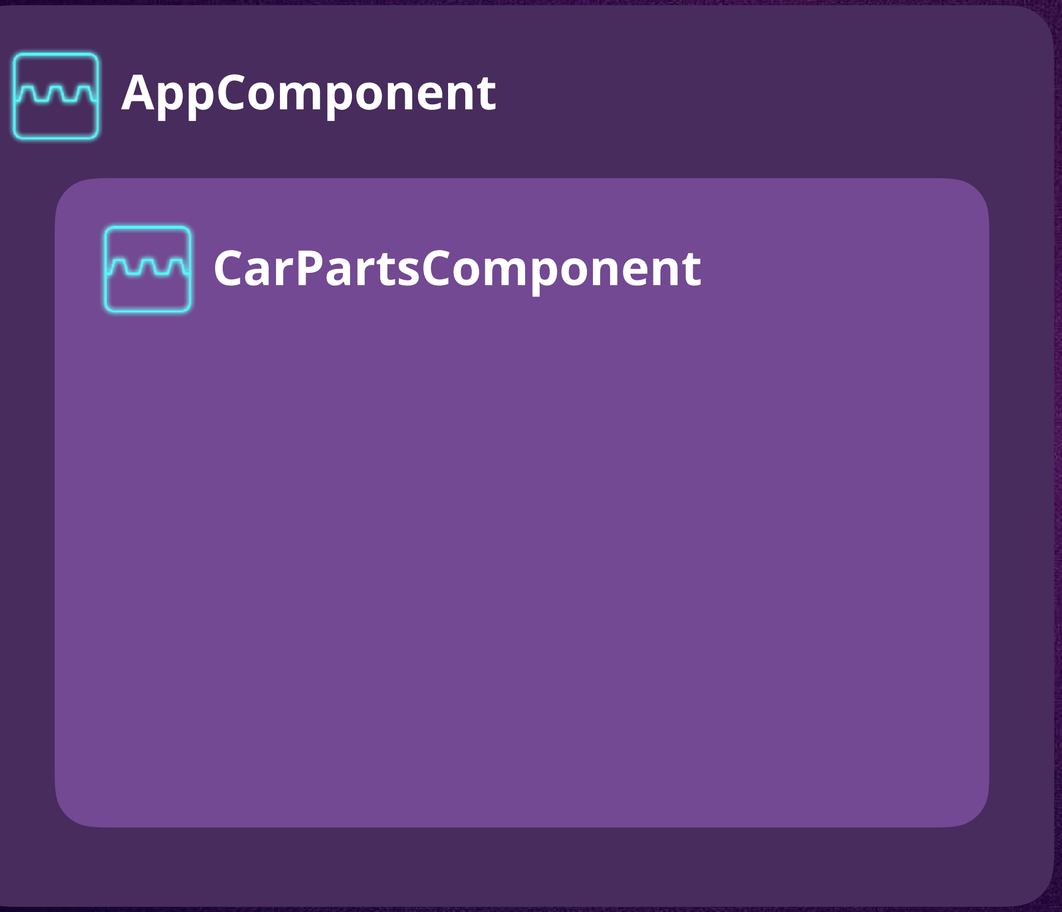
Components are meant to be reusable, so we could use them in different parts of our application.

Angular 2 Uses Component-based Architecture

Components can be all over the place in your application.

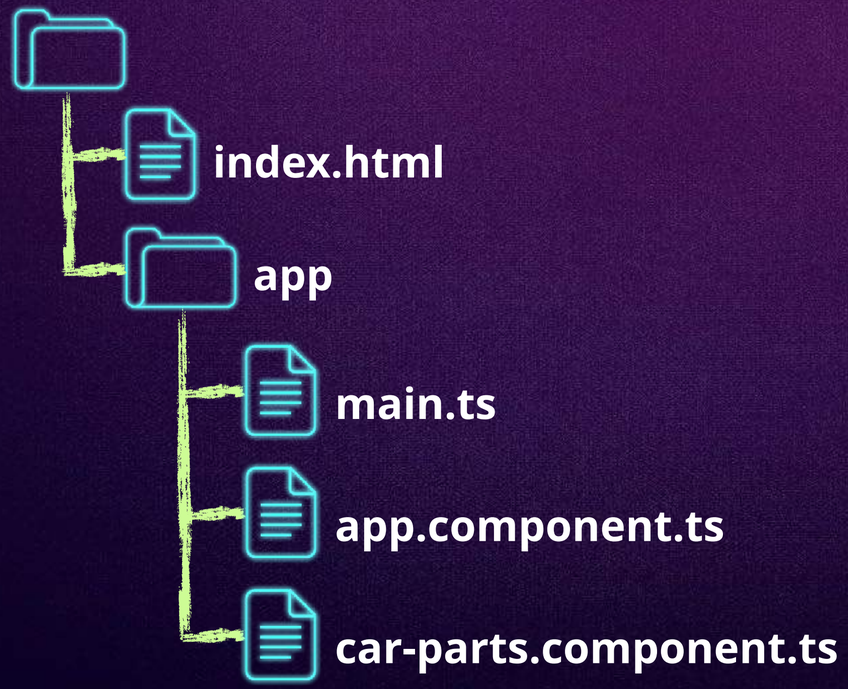
This isn't what we're building — we'll keep it even simpler.

Our Two Components



What'd We Learn?

- Our `main.ts` is where we import our first component and bootstrap it.
- In order to import a class, we must give it the `export` keyword.
- We use the `directives` metadata to list the directives our component uses.
- Components are the building blocks of our application.




ACCELERATING THROUGH
ANGULAR 2

Level 3

Component HTML & CSS

Section 2

How Do We Tie CSS to a Component?

car-parts.component.ts TypeScript

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

@Component({
  selector: 'car-parts',
  template: `...
              <p>{{carPart.description}}</p>
              <p>{{carPart.price | currency:'EUR':true}}</p>
            ...`

})
export class CarPartsComponent {
...
```

We have an HTML template, and we can even include CSS.

Adding Styles Array Metadata

car-parts.component.ts TypeScript

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

@Component({
  selector: 'car-parts',
  template: `...
    <p class="description">{{carPart.description}}</p>
    <p class="price">{{carPart.price | currency:'EUR':true}}</p>
          ...`,
  styles:[`
    .description {
      color: #444;
      font-size: small;
    }
    .price {
      font-weight: bold;
    }
  `]
})
export class CarPartsComponent {
```

New CSS classes

Notice this is an array.

The CSS Is Scoped to the Component

The HTML Source

```
<p _ngcontent-dcy-2 class="description">These tires are the very best</p>
<p _ngcontent-dcy-2 class="price">€4.99</p>
```

Notice the custom attribute.

The CSS Source

```
.description[_ngcontent-dcy-2] {
  color: #444;
  font-size: small;
}
.price[_ngcontent-dcy-2] {
    font-weight: bold;
}
```

Angular 2 adds this custom attribute to scope the CSS to only this component.

Kinda like properties are scoped, the CSS is scoped too!

Mind Blown

Splitting Things Into More Pieces

Up until now, we've included the HTML and CSS alongside our code.

car-parts.component.ts

I don't know about you, but this feels messy to me.

Let's split out our HTML and CSS into different files.

car-parts.component.html
Where our HTML for the component lives.

car-parts.component.css
Where our CSS for the component lives.

Our Current Component

car-parts.component.ts TypeScript

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

@Component({
  selector: 'car-parts',
  template: `
    <p>There are {{totalCarParts()}} total parts in stock.</p>
    <ul>...</ul>`,
   styles:[`
     .description {
       color: #444;
       font-size: small;
     }
     .price {
       font-weight: bold;
     }
   `]
})
export class CarPartsComponent {
  ...
```

We need to move out the HTML and CSS.

Moving Out the HTML & CSS

car-parts.component.ts TypeScript

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

@Component({
  selector: 'car-parts',
  templateUrl: 'app/car-parts.component.html',
  styleUrls:['app/car-parts.component.css']
})
export class CarPartsComponent {
  ...
```

car-parts.component.html HTML

```
<p>There are {{totalCarParts()}} total parts in stock.</p>
<ul>...</ul>
```

car-parts.component.css CSS

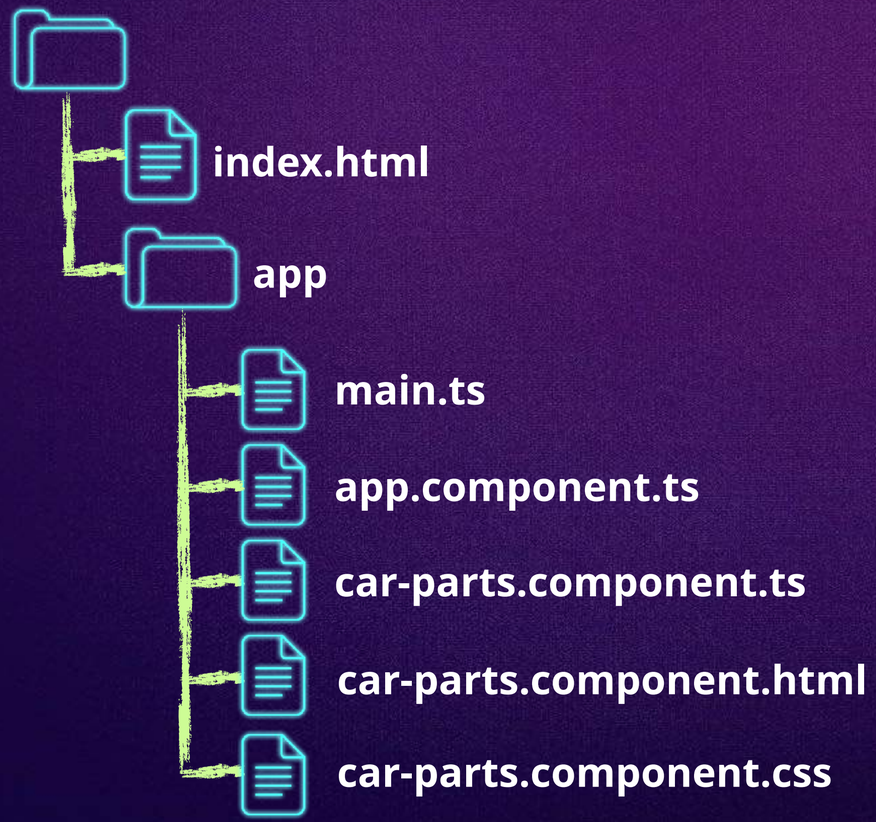
```
.description {
  color: #444;
  font-size: small;
}
.price {
  font-weight: bold;
}
```

Once we create new files for our HTML and CSS, we can reference them inside our component metadata.

The CSS still gets scoped just like before.

What’d We Learn?

- We can include CSS just like we include our HTML template.
- CSS automatically gets scoped to our component.
- HTML and CSS can get split out into their own files.




ACCELERATING THROUGH
ANGULAR 2

Level 3

Mocks & Models

Section 3

Getting More Object Oriented

TypeScript gives us the ability to be more object oriented with our data, so let's create a model.

Which is basically a class in JavaScript.

car-part.ts TypeScript

```
export class CarPart {
  id: number;
  name: string;
  description: string;
  inStock: number;
  price: number;
}
```

Notice we're declaring what type each of our properties are.
This is TypeScript.

This will allow our compiler to check our code and ensure we're getting and setting things properly.

Our Previous Code

car-parts.component.ts TypeScript

```
import { Component } from '@angular/core';
...
})
export class CarPartsComponent {
  carParts = [{
    "id": 1,
    "name": "Super Tires",
    "description": "These tires are the very best",
    "inStock": 5,
    "price": 4.99
  }, { ... }, { ... }];
  ...
```

How do we use our new model?

car-part.ts TypeScript

```
export class CarPart {
  id: number;
  name: string;
  description: string;
  inStock: number;
  price: number;
}
```

Import the CarPart Model & Define Type

car-parts.component.ts TypeScript

```
import { Component } from '@angular/core';
import { CarPart } from './car-part';
...
})
export class CarPartsComponent {
  carParts: CarPart[] = [{
    "id": 1,
    "name": "Super Tires",
    "description": "These tires are the very best",
    "inStock": 5,
    "price": 4.99
  }, { ... }, { ... }];
  ...
```

Import the CarPart *model*

Tells TypeScript to treat this like an array of CarParts

car-part.ts TypeScript

```
export class CarPart {
  id: number;
  name: string;
  description: string;
  inStock: number;
  price: number;
}
```

Nothing Else Needs to Change

car-parts.component.ts TypeScript

```
...
carParts: CarPart[] = [{
  "id": 1,
  "name": "Super Tires",
  "description": "These tires are the very best",
  "inStock": 5,
  "price": 4.99
}, { ... }, { ... }];
```

car-part.ts TypeScript

```
export class CarPart {
  id: number;
  name: string;
  description: string;
  inStock: number;
  price: number;
}
```

car-parts.component.html HTML

```
<p>There are {{totalCarParts()}} total parts in stock.</p>
<ul>
  <li *ngFor="let carPart of carParts" >
    <h2>{{carPart.name | uppercase}}</h2>
    <p class="description">{{carPart.description}}</p>
    <p class="price">{{carPart.price | currency:'EUR':true }}</p>
    <p *ngIf="carPart.inStock > 0">{{carPart.inStock}} in Stock</p>
    <p *ngIf="carPart.inStock === 0">Out of Stock</p>
  </li>
</ul>
```

Cleaning Up Our Mock Data

Eventually we want to call out to a web service (API) to get the latest car parts, so it's a good practice to move our mock (fake) data out into its own file.

car-parts.component.ts TypeScript

```typescript
import { Component } from '@angular/core';
import { CarPart } from './car-part';
...
})
export class CarPartsComponent {
  carParts: CarPart[] = [{
    "id": 1,
    "name": "Super Tires",
    "description": "These tires are the very best",
    "inStock": 5,
    "price": 4.99
  }, { ... }, { ... }];
  ...
```

mocks.ts TypeScript

Let's create a new file and move our mock data here.

Using Our Mock Data

car-parts.component.ts TypeScript

```
import { Component } from '@angular/core';
import { CarPart } from './car-part';
import { CARPARTS } from './mocks';

...
})
export class CarPartsComponent {
  carParts: CarPart[];

  ngOnInit() {
    this.carParts = CARPARTS;
  }
  ...
```

mocks.ts TypeScript

```
import { CarPart } from './car-part';

export const CARPARTS: CarPart[] = [{
  "id": 1,
  "name": "Super Tires",
  "description": "These tires are the very best",
  "inStock": 5,
  "price": 4.99
}, { ... }, { ... }];
```

Notice we use `const` instead of `let` — this is an ES2015 feature that makes sure CARPARTS can't be reassigned.

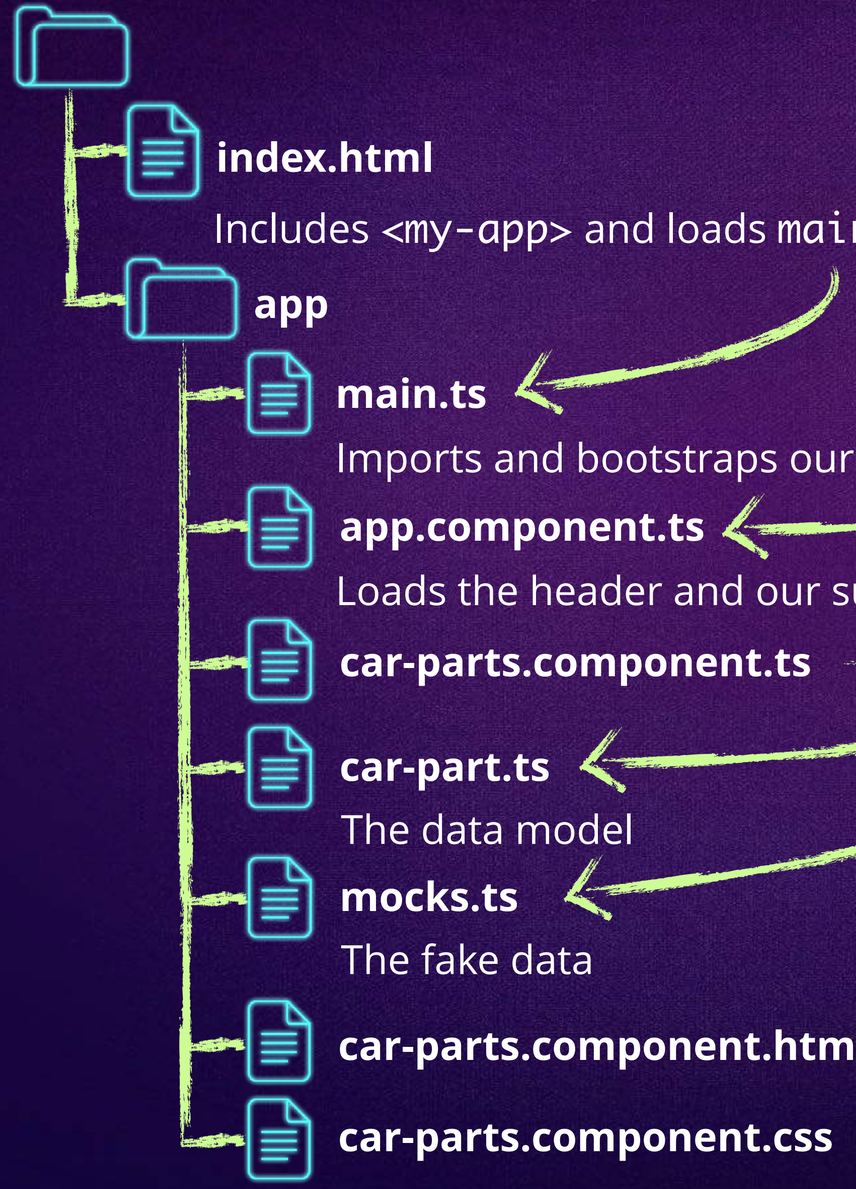
`ngOnInit` **is invoked after the component is constructed and is the best place to initialize property values.**

We could have initialized in the constructor, but that'd be harder to test.

Yay, It Still Works!

We didn’t add any more functionality, but our code became a lot easier to maintain and scale.

Our Component Architecture



app/car-parts.component.ts

```
import { Component } from '@angular/core';
import { CarPart } from './car-part';
import { CARPARTS } from './mocks';

@Component({
  selector: 'car-parts',
  templateUrl: 'app/car-parts.component.html',
  styleUrls:['app/car-parts.component.css']
})
export class CarPartsComponent {
  carParts: CarPart[];
  ...
```

What'd We Learn?

- In TypeScript, we can use classes to model our data.
- TypeScript helps us specify class property types that help our compiler ensure we're writing good code.
- It's a good practice to keep your mock data separate from your model and your components, in its own file.


ACCELERATING THROUGH
ANGULAR 2