Angular 2 Development with TypeScript

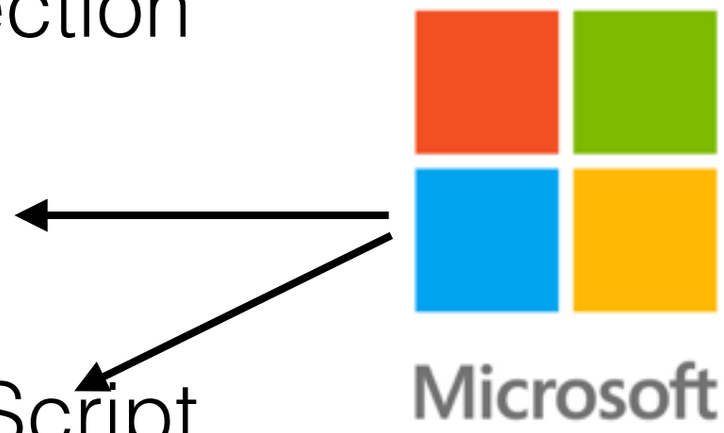
Yakov Fain, Farata Systems

The Agenda

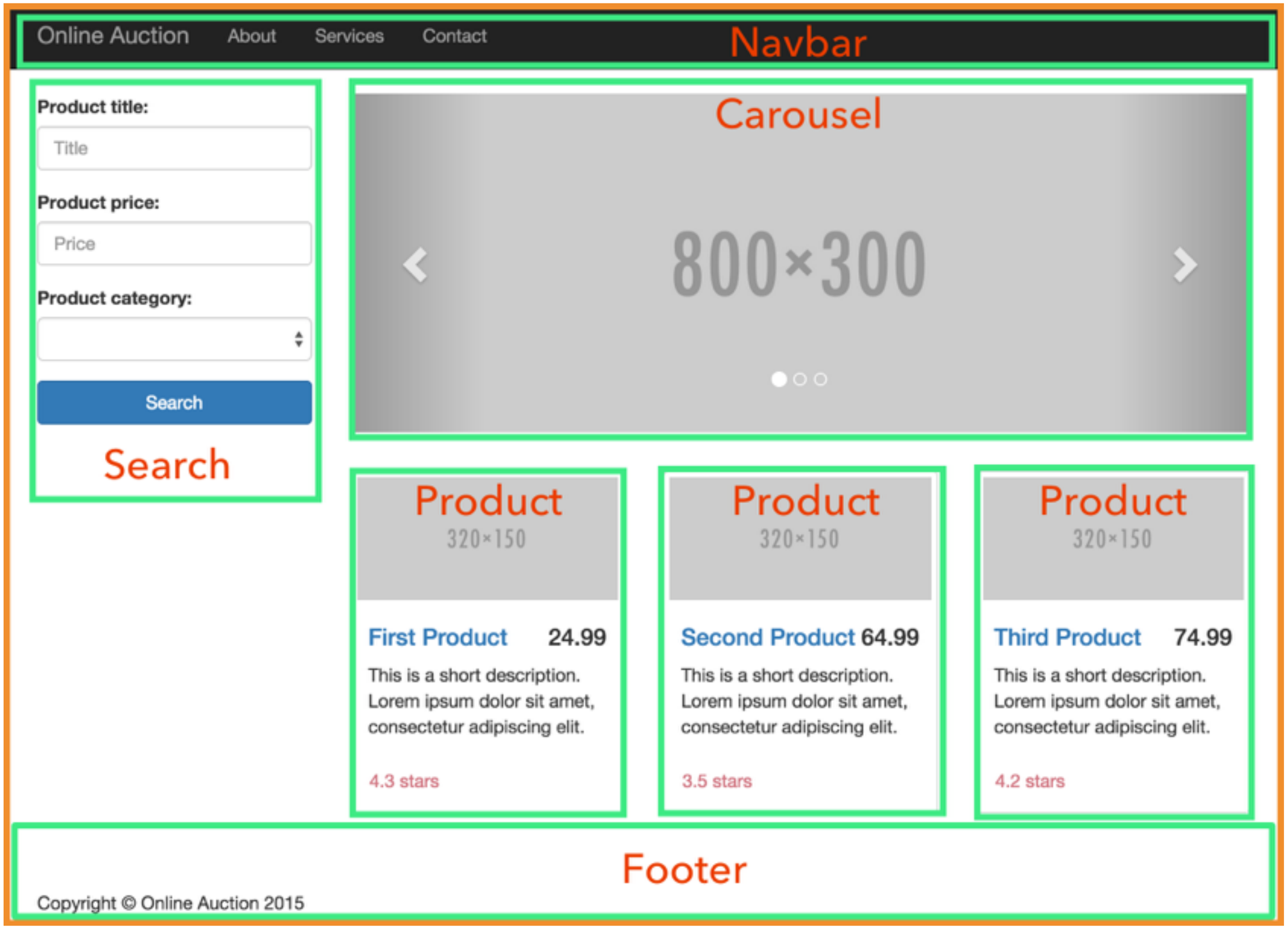
- High-level overview of one Angular app
- Getting familiar with TypeScript
- Templates and Bindings
- Router and Dependency Injection
- Observables and HTTP
- Deployment

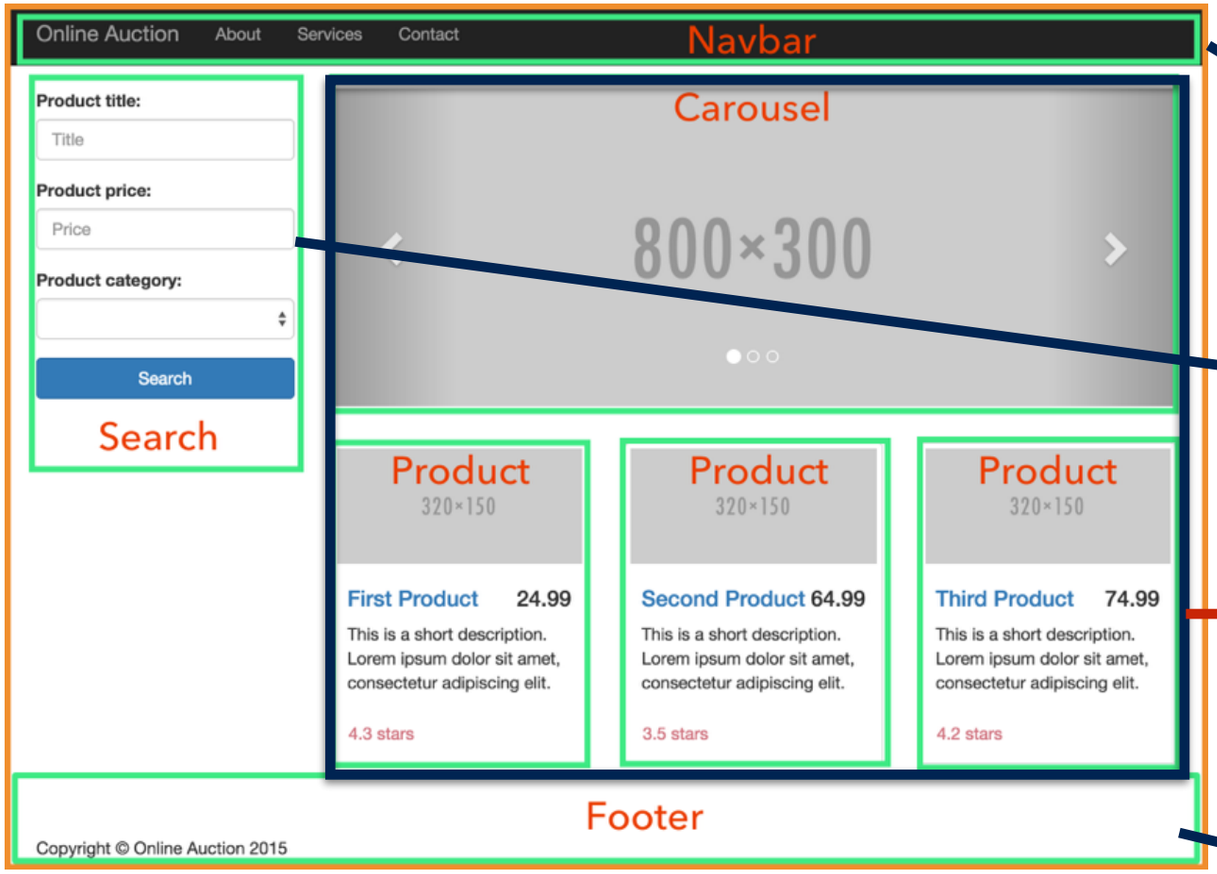
Angular 2

- Complete re-write of AngularJS ← Google
- Component-based
- No controllers, scopes
- Better Dependency Injection
- Integrated RxJS
- Can write apps in TypeScript, Dart, or JavaScript (ES5 or ES6)



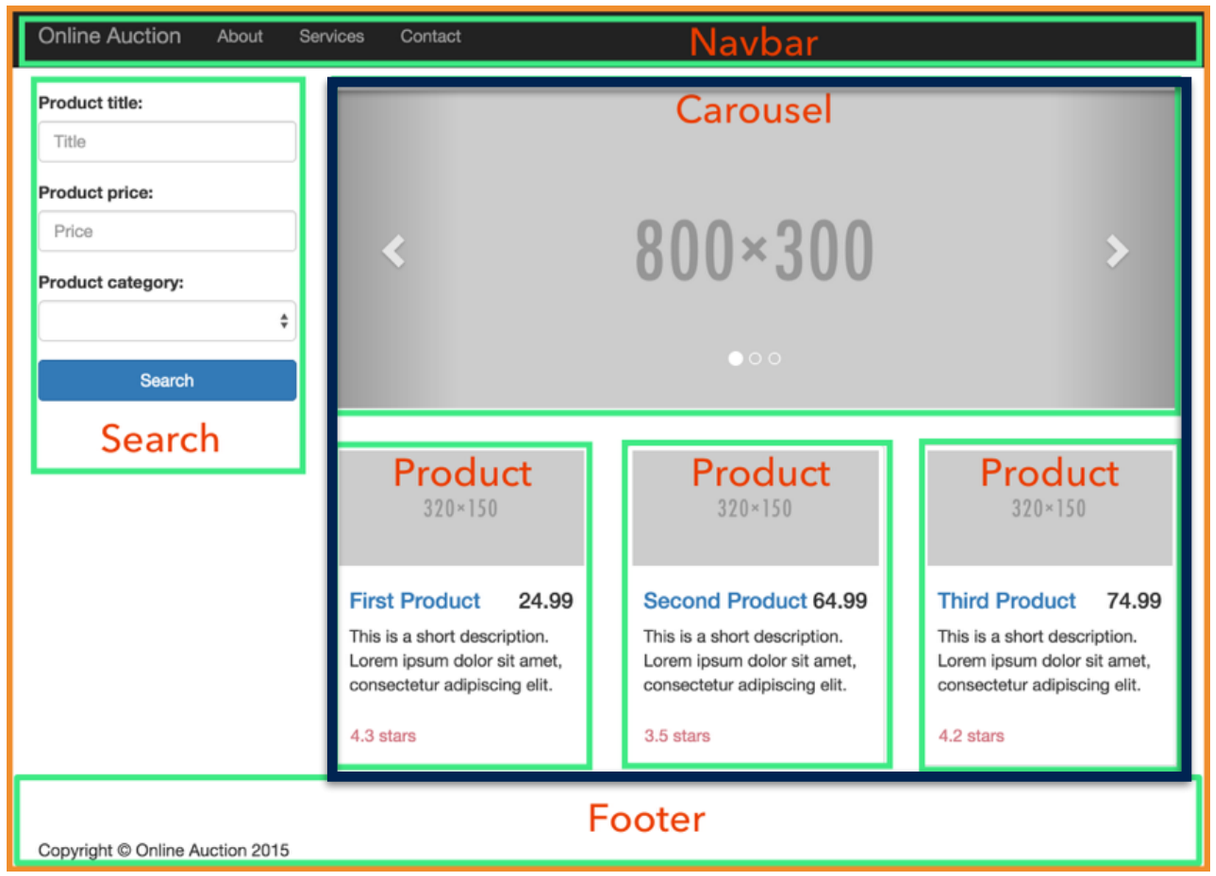
An app is a tree of components






```
<auction-navbar></auction-navbar>

<div class="container">
  <div class="row">
    <div class="col-md-3">
      <auction-search></auction-search>
    </div>

    <div class="col-md-9">
      <router-outlet></router-outlet>
    </div>
  </div>
</div>

<auction-footer></auction-footer>
```

HTML

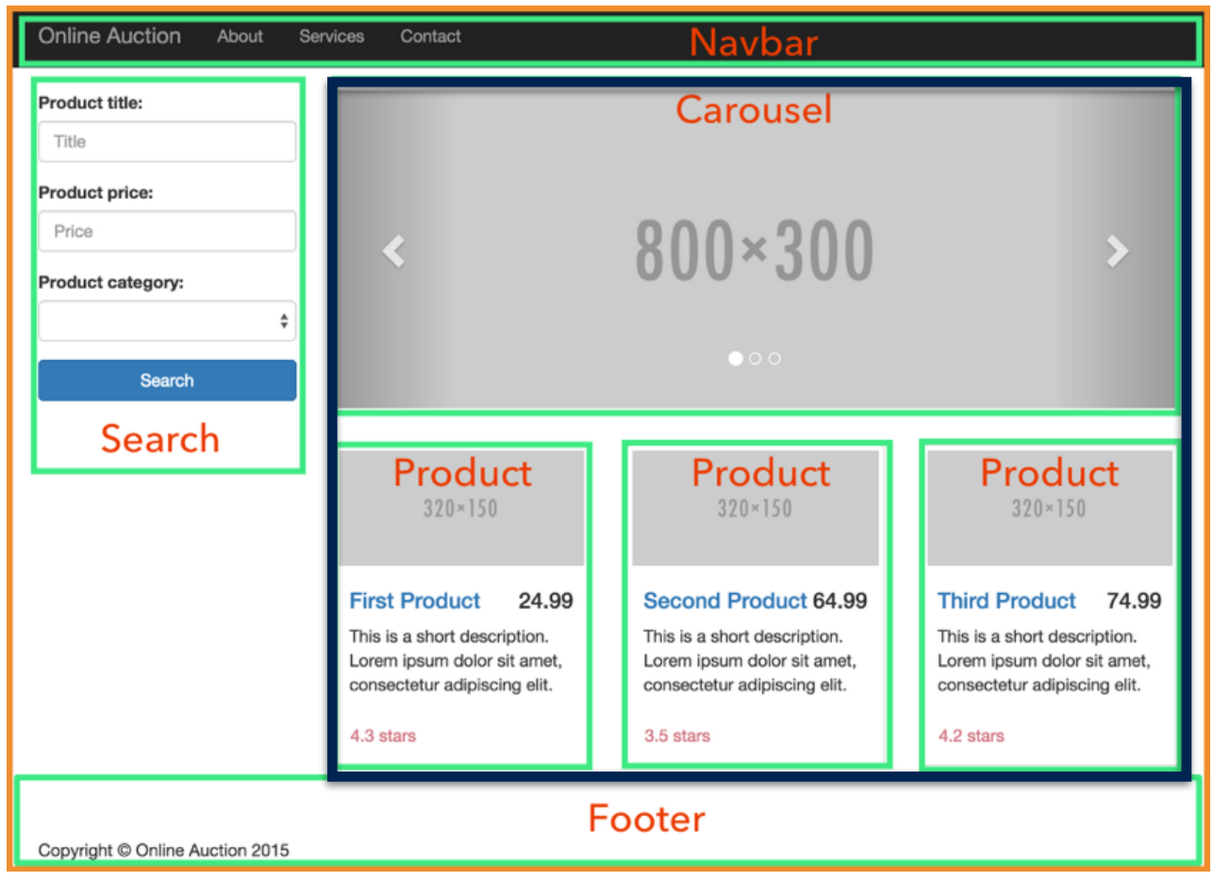



```
<auction-navbar></auction-navbar>

<div class="container">
  <div class="row">
    <div class="col-md-3">
      <auction-search></auction-search>
    </div>

    <div class="col-md-9">
      <router-outlet></router-outlet>
    </div>
  </div>
</div>

<auction-footer></auction-footer>
```

```
import {Component} from 'angular2/core';
import {Route, RouteConfig, RouterOutlet} from 'angular2/router';
import NavbarComponent from '../navbar/navbar';
import FooterComponent from '../footer/footer';
import SearchComponent from '../search/search';

import HomeComponent from '../home/home';
import ProductDetailComponent from "../product-detail/product-detail";

@Component({
  selector: 'auction-application',
  templateUrl: 'app/components/application/application.html',
  directives: [
    RouterOutlet,
    NavbarComponent,
    FooterComponent,
    SearchComponent,
    HomeComponent
  ]
})
@RouteConfig([
  {path: '/', component: HomeComponent, as: 'Home'},
  {path: '/products/:productId', component: ProductDetailComponent, as: 'ProductDetail'}
])
export default class ApplicationComponent {}
```

HTML

A TypeScript class



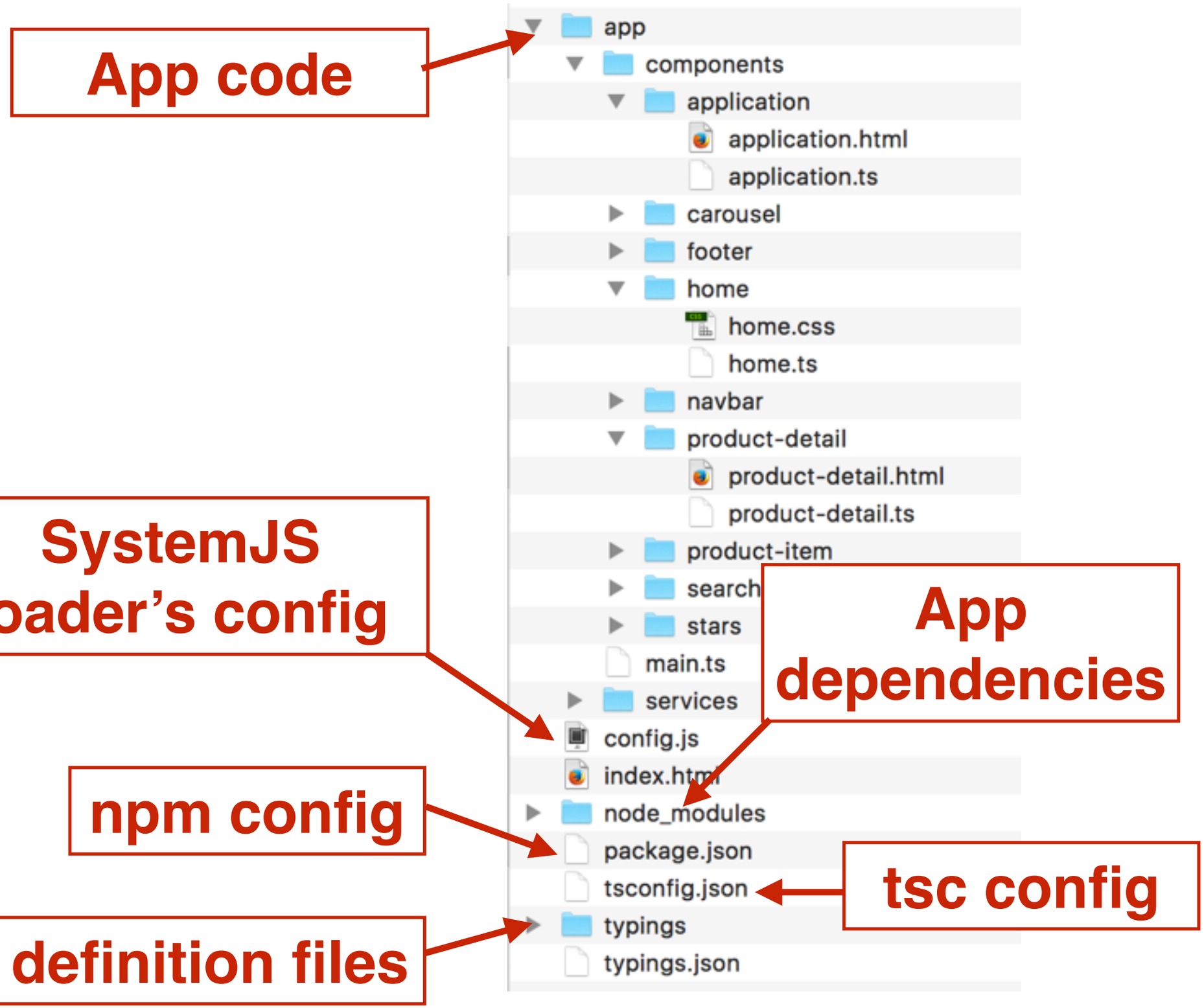

```
<auction-navbar></auction-navbar>

<div class="container">
  <div class="row">
    <div class="col-md-3">
      <auction-search></auction-search>
    </div>

    <div class="col-md-9">
      <router-outlet></router-outlet>
    </div>
  </div>
</div>

<auction-footer></auction-footer>
```

```
import {Component} from 'angular2/core';
import {Route, RouteConfig, RouterOutlet} from 'angular2/router';
import NavbarComponent from '../navbar/navbar';
import FooterComponent from '../footer/footer';
import SearchComponent from '../search/search';

import HomeComponent from '../home/home';
import ProductDetailComponent from "../product-detail/product-detail";

@Component({
  selector: 'auction-application',
  templateUrl: 'app/components/application/application.html',
  directives: [
    RouterOutlet,
    NavbarComponent,
    FooterComponent,
    SearchComponent,
    HomeComponent
  ]
})
@RouteConfig([
  {path: '/', component: HomeComponent, as: 'Home'},
  {path: '/products/:productId', component: ProductDetailComponent, as: 'ProductDetail'}
])
export default class ApplicationComponent {}
```

Importing modules

Class annotations

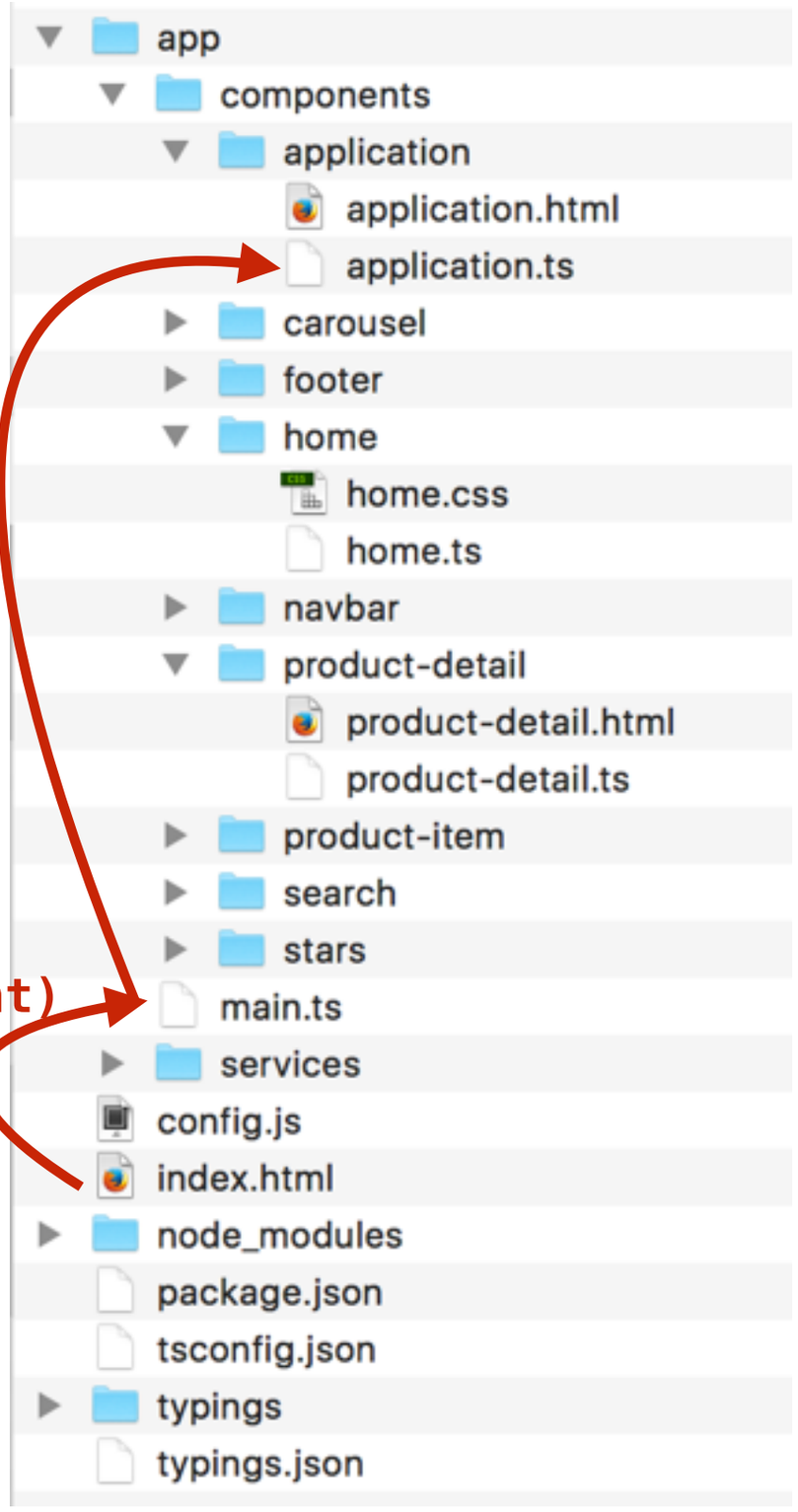
Project Structure



Project Structure

SystemJS transpiles TS and loads JS

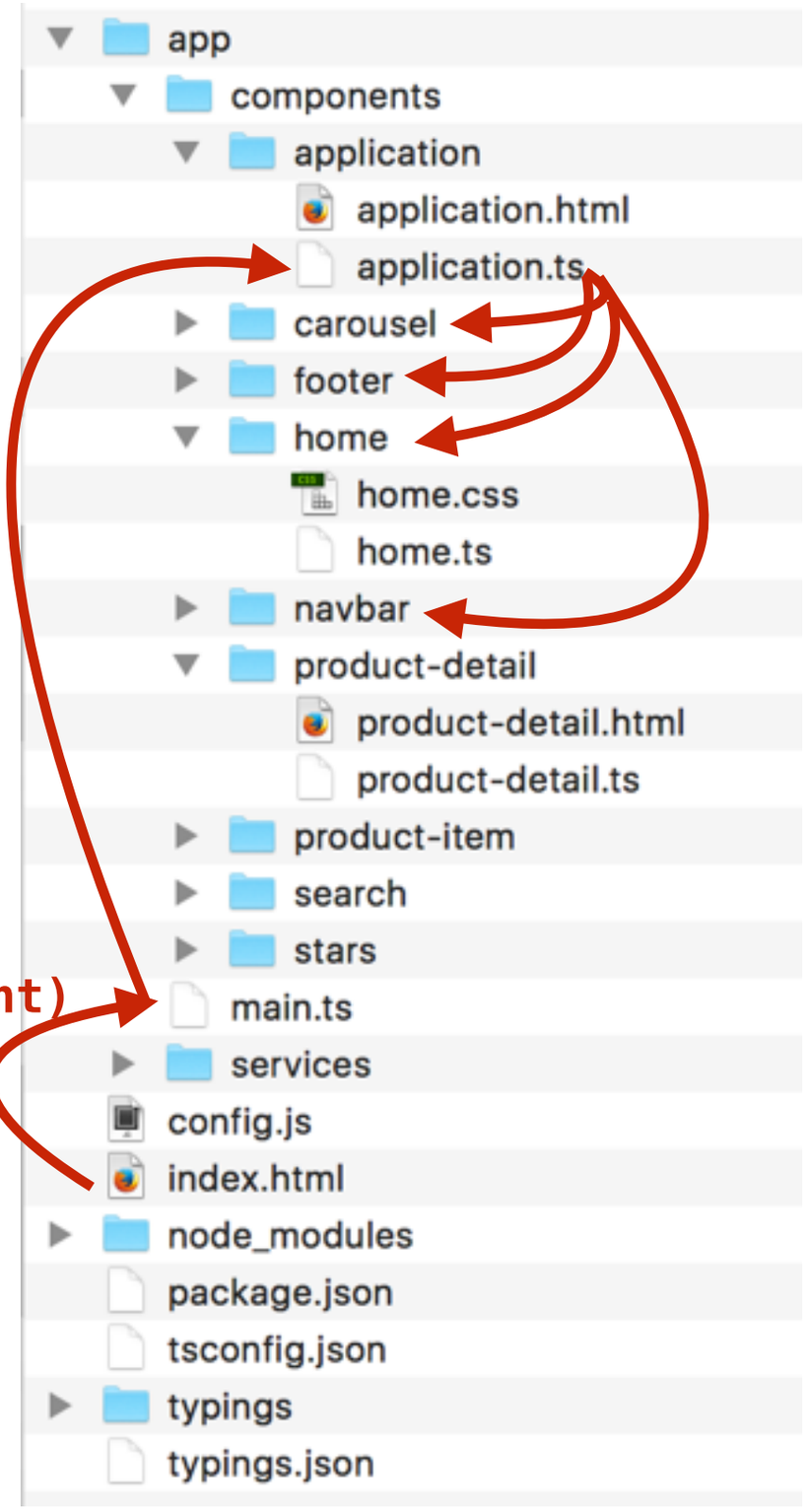
```
bootstrap(ApplicationComponent)
```



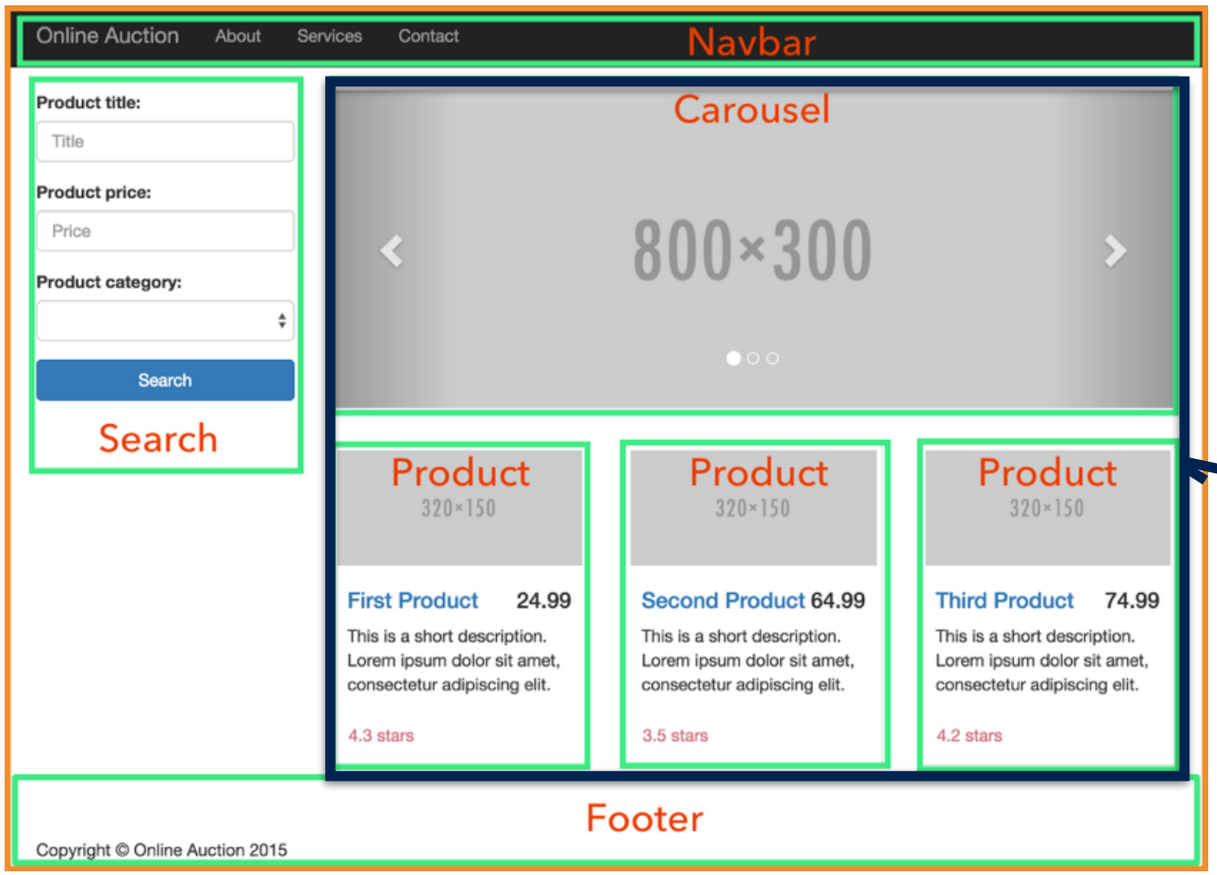
Project Structure

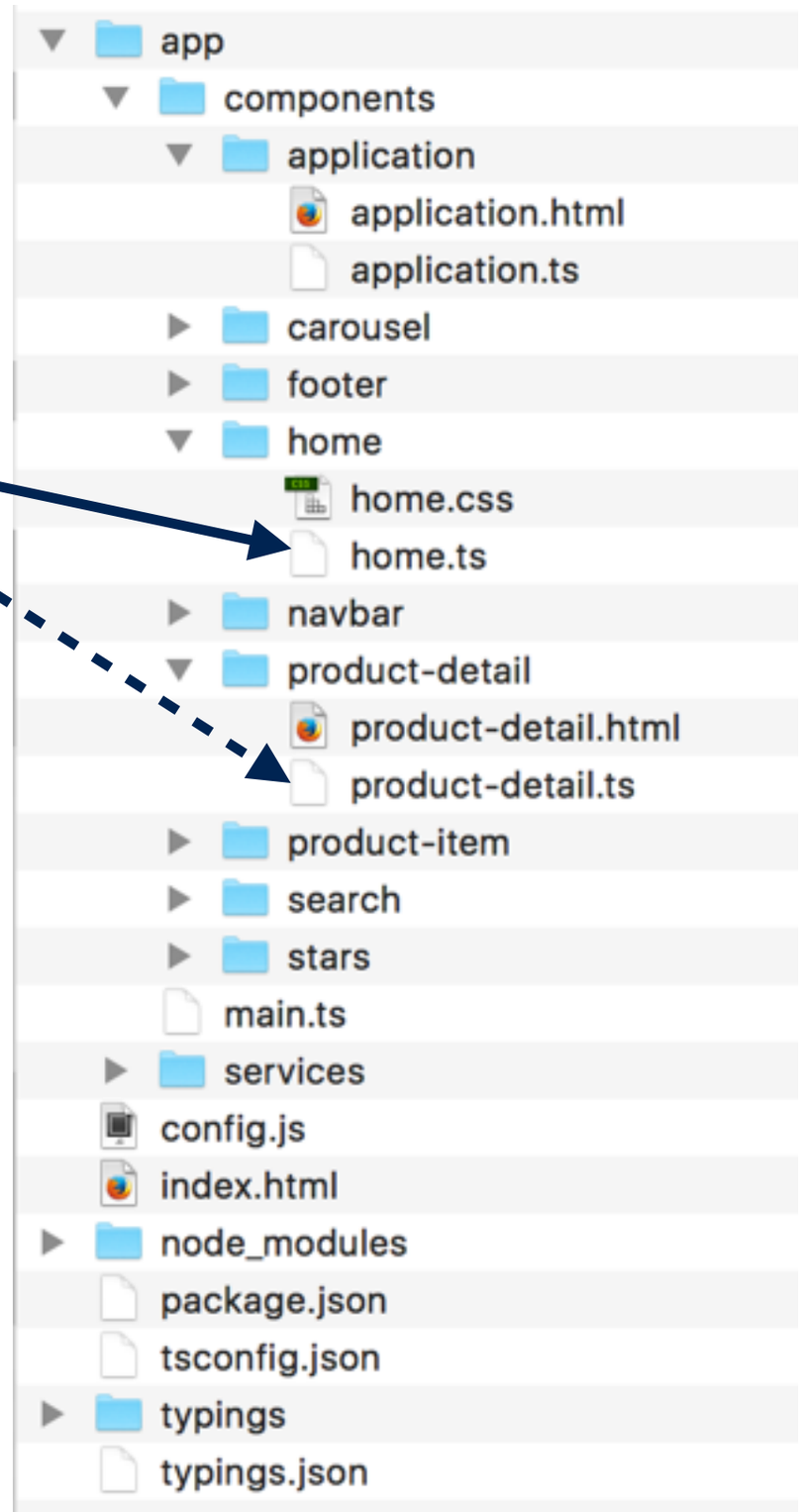
SystemJS transpiles TypeScript and loads JavaScript

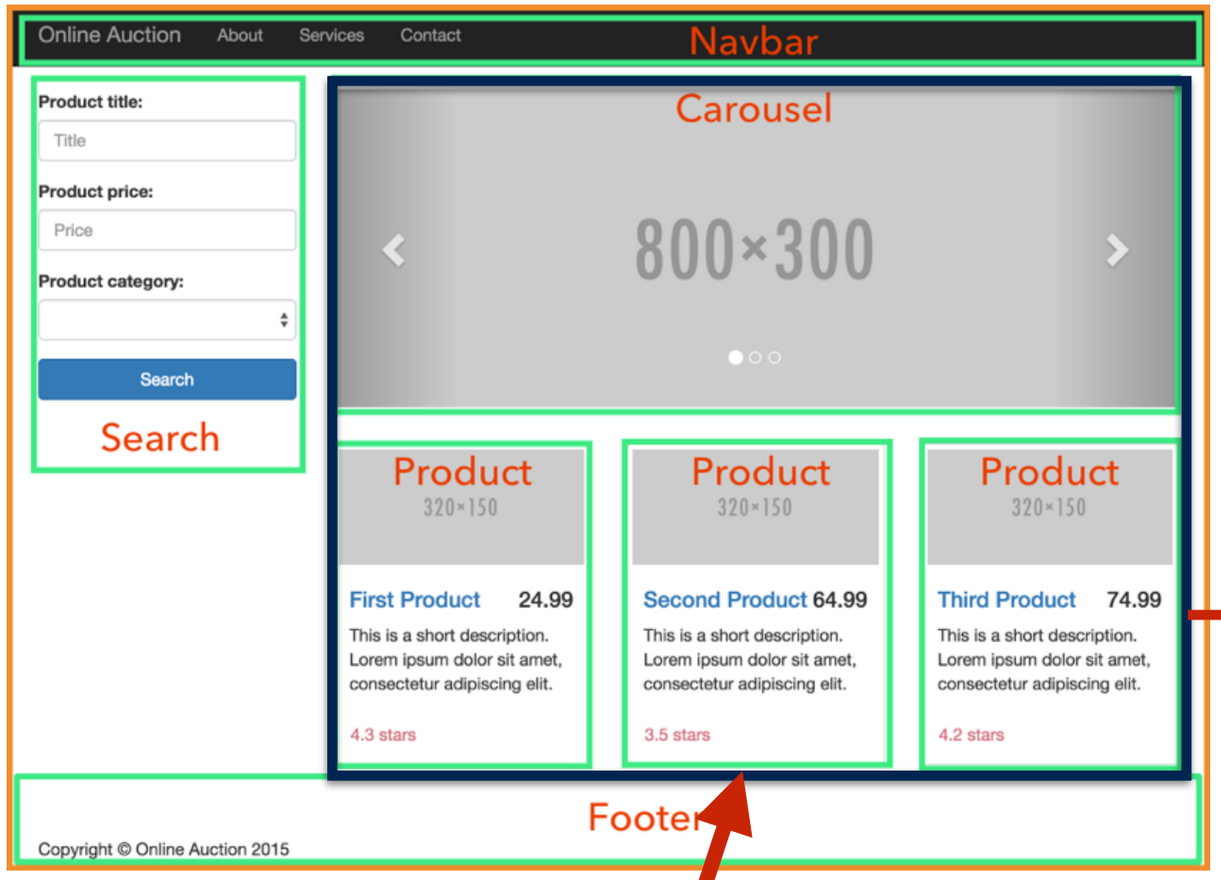
```
bootstrap(ApplicationComponent)
```



Navigating with Component Router



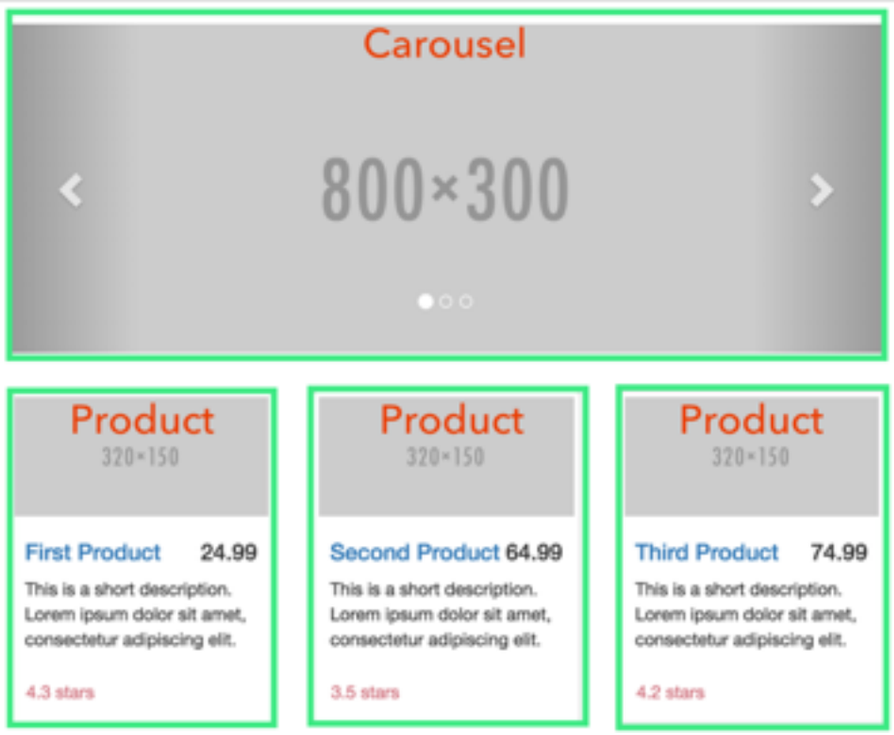




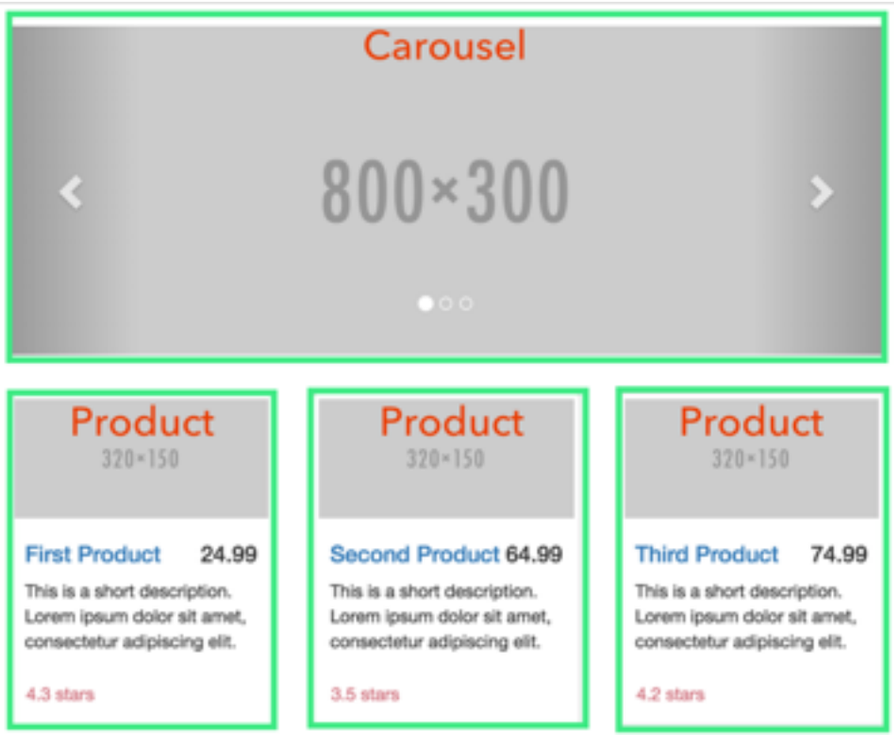

```
<auction-navbar></auction-navbar>

<div class="container">
  <div class="row">
    <div class="col-md-3">
      <auction-search></auction-search>
    </div>

    <div class="col-md-9">
      <router-outlet></router-outlet>
    </div>
  </div>
</div>

<auction-footer></auction-footer>
```

```
import {Component} from 'angular2/core';
import {Route, RouteConfig, RouterOutlet} from 'angular2/router';
import NavbarComponent from '../navbar/navbar';
import FooterComponent from '../footer/footer';
import SearchComponent from '../search/search';

import HomeComponent from '../home/home';
import ProductDetailComponent from "../product-detail/product-detail";

@Component({
  selector: 'auction-application',
  templateUrl: 'app/components/application/application.html',
  directives: [
    RouterOutlet,
    NavbarComponent,
    FooterComponent,
    SearchComponent,
    HomeComponent
  ]
})
@RouteConfig([
  {path: '/', component: HomeComponent, as: 'Home'},
  {path: '/products/:productId', component: ProductDetailComponent, as: 'ProductDetail'}
])
export default class ApplicationComponent {}
```

HomeComponent



HomeComponent

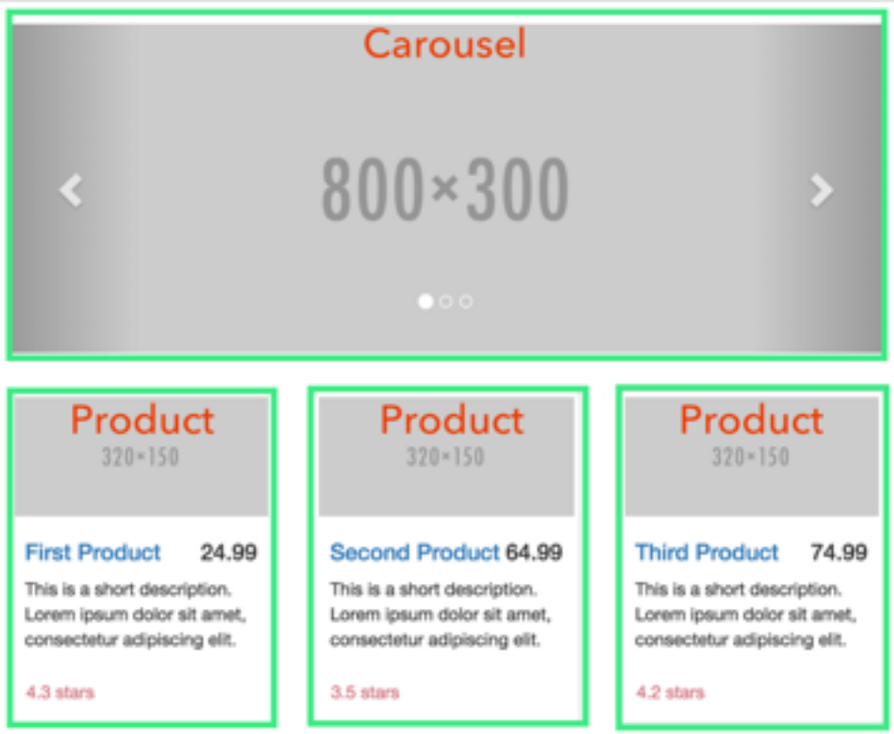


Exporing a module

```
import {Component} from 'angular2/core';
import { NgFor} from 'angular2/common';
import {Product, ProductService} from 'app/services/product-service';
import CarouselComponent from '../carousel/carousel';
import ProductItemComponent from '../product-item/product-item';
import {ProductService} from '../../services/product-service';

@Component({
  selector: 'auction-home-page',
  directives: [
    NgFor,
    CarouselComponent,
    ProductItemComponent
  ],
  styleUrls: ['app/components/home/home.css'],
  template: `
    <div class="row carousel-holder">
      <div class="col-md-12">
        <auction-carousel></auction-carousel>
      </div>
    </div>
    <div class="row">
      <div *ngFor="#product of products" class="col-sm-4 col-lg-4 col-md-4">
        <auction-product-item [product]="product"></auction-product-item>
      </div>
    </div>
  `
})
export default class HomeComponent {
  products: Product[] = [];

  constructor(private productService: ProductService) {
    this.products = this.productService.getProducts();
  }
}
```

Importing Modules

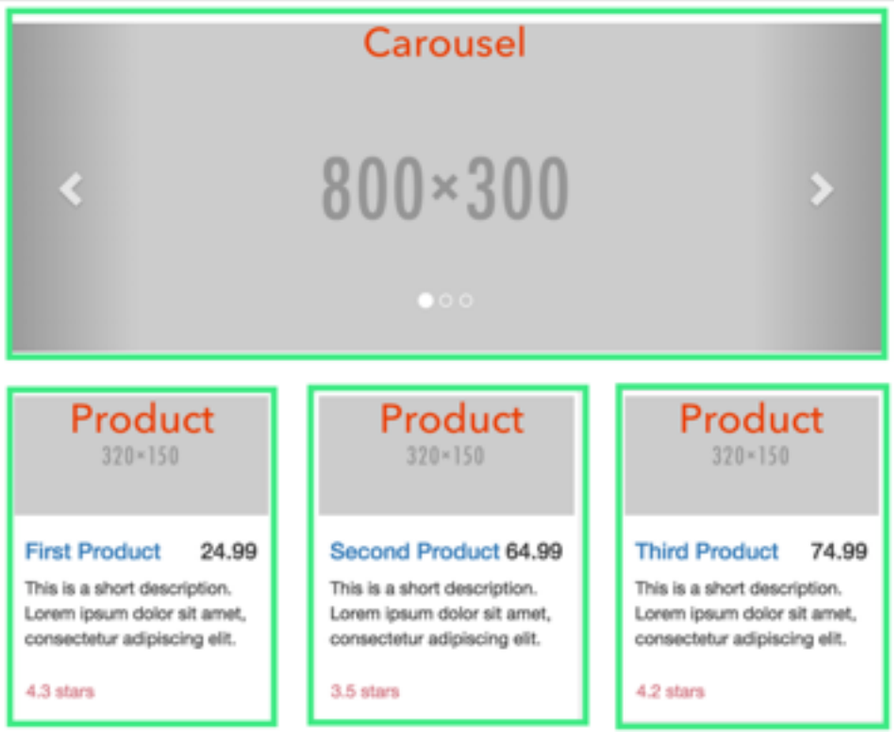
HomeComponent



Dependency Injection

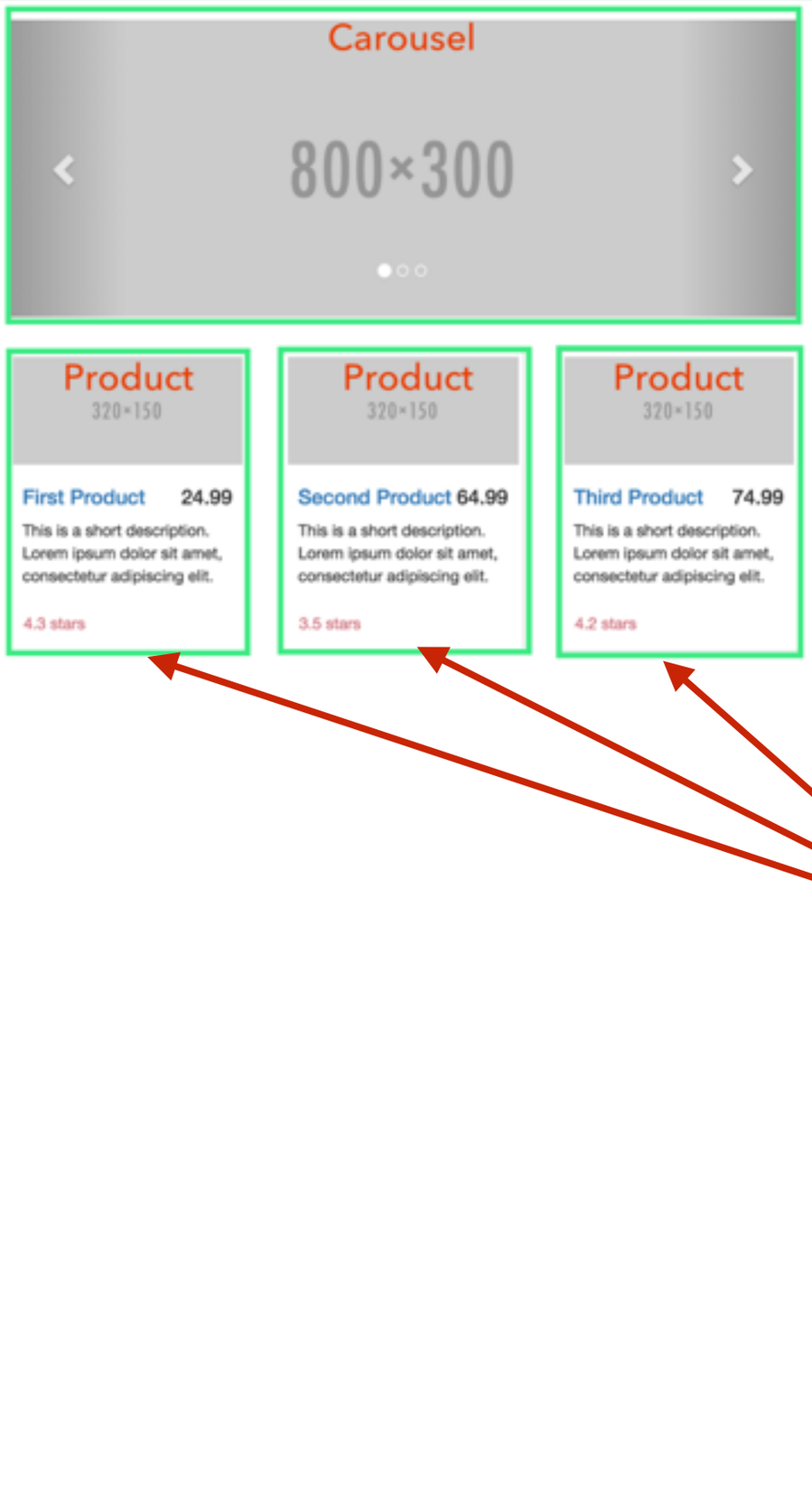
```
import {Component} from 'angular2/core';
import { NgFor} from 'angular2/common';
import {Product, ProductService} from 'app/services/product-service';
import CarouselComponent from '../carousel/carousel';
import ProductItemComponent from '../product-item/product-item';
import {ProductService} from '../../services/product-service';

@Component({
  selector: 'auction-home-page',
  directives: [
    NgFor,
    CarouselComponent,
    ProductItemComponent
  ],
  styleUrls: ['app/components/home/home.css'],
  template: `
    <div class="row carousel-holder">
      <div class="col-md-12">
        <auction-carousel></auction-carousel>
      </div>
    </div>
    <div class="row">
      <div *ngFor="#product of products" class="col-sm-4 col-lg-4 col-md-4">
        <auction-product-item [product]="product"></auction-product-item>
      </div>
    </div>
  `
})
export default class HomeComponent {
  products: Product[] = [];

  constructor(private productService: ProductService) {
    this.products = this.productService.getProducts();
  }
}
```

Injectable ProductService



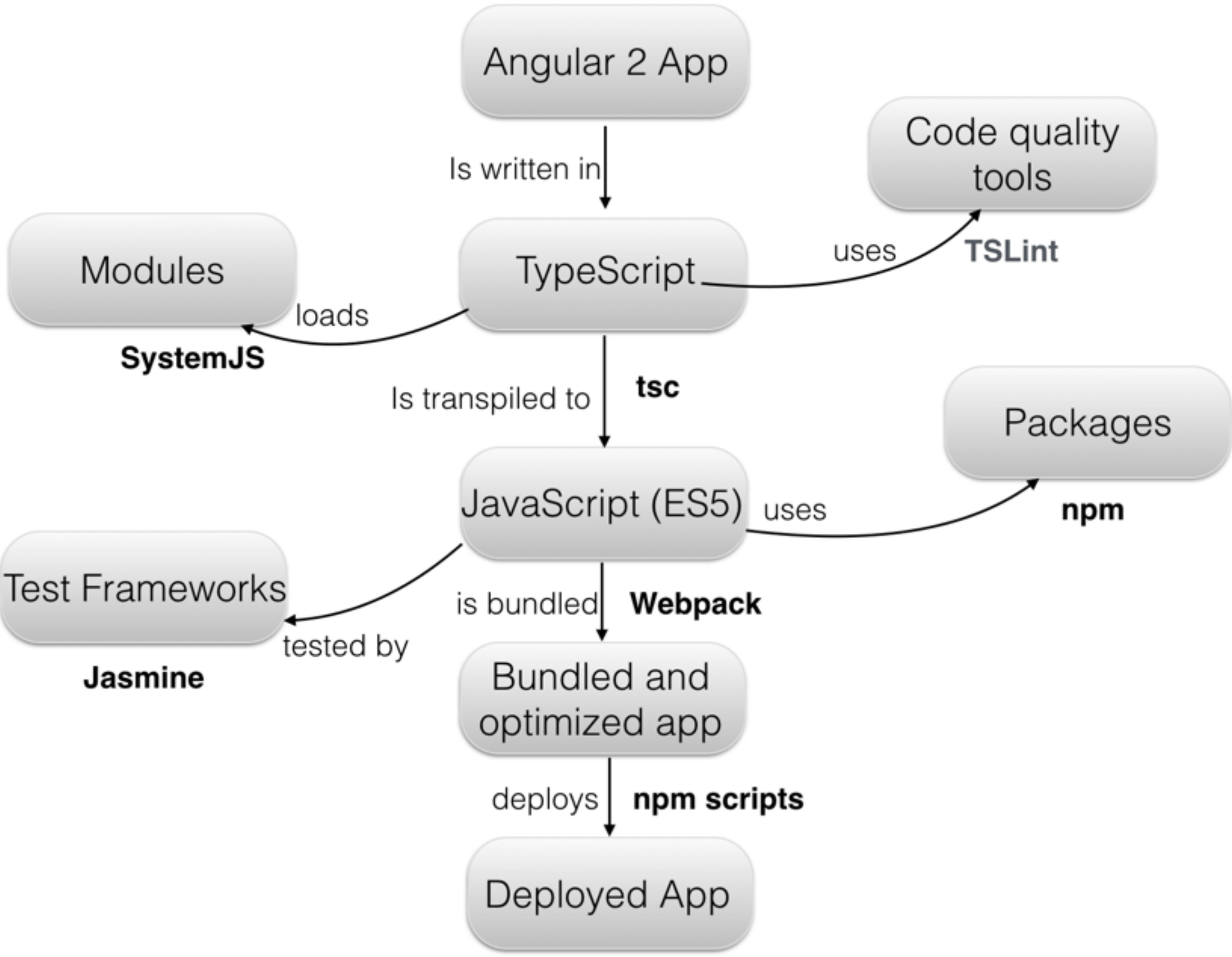

```
@Injectable()
export class ProductService {
  getProducts(): Product[] {
    return products.map(p => new Product(p.id, p.title, p.price, p.rating, p.description, p.categories));
  }

  getReviewsForProduct(productId: number): Review[] {
    return reviews
        .filter(r => r.productId === productId)
        .map(r => new Review(r.id, r.productId, new Date(r.timestamp), r.user, r.rating, r.comment));
  }
}

var products = [
  {
    "id": 0,
    "title": "First Product",
    "price": 24.99,
    "rating": 4.3,
    "description": "This is a short description. Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
    "categories": ["electronics", "hardware"]
  },
  {
    "id": 1,
```

Looping with *ngFor




```
import {Component} from 'angular2/core';
import { NgFor} from 'angular2/common';
import {Product, ProductService} from 'app/services/product-service';
import CarouselComponent from '../carousel/carousel';
import ProductItemComponent from '../product-item/product-item';
import {ProductService} from '../../services/product-service';

@Component({
  selector: 'auction-home-page',
  directives: [
    NgFor,
    CarouselComponent,
    ProductItemComponent
  ],
  styleUrls: ['app/components/home/home.css'],
  template: `
    <div class="row carousel-holder">
      <div class="col-md-12">
        <auction-carousel></auction-carousel>
      </div>
    </div>
    <div class="row">
      <div *ngFor="#product of products" class="col-sm-4 col-lg-4 col-md-4">
        <auction-product-item [product]="product"></auction-product-item>
      </div>
    </div>
  `
})
export default class HomeComponent {
  products: Product[] = [];

  constructor(private productService: ProductService) {
    this.products = this.productService.getProducts();
  }
}
```

A Sample Toolbox



Transpilers

- Convert the source code from one language to another
- Allow writing in TypeScript or ES6 and deploy in ES5 **today**
- Popular transpilers: Babel, Traceur, TypeScript

A 10-min Intro to TypeScript

http://www.typescriptlang.org

What's TypeScript?

- An open-source language developed at Microsoft
- Designed by Anders Hejlsberg, the creator of C#, Delphi, and Turbo Pascal
- A superset of JavaScript
- Static type analysis
- Well supported by IDEs

Benefits of Writing in TypeScript

- Increases your productivity in producing JavaScript
- Supports types, classes, interfaces, generics, annotations
- Compiles into a human-readable JavaScript
- Easy to learn by Java, C#, and C++ developers
- Supports most of the ES6 and some ES7 syntax

Transpiling TypeScript Interactively

http://www.typescriptlang.org/Playground

TypeScript → JavaScript (E5)

```
1  var foo: string;
2
3  class Bar{
4
5  }
```

```
1 var foo;
2 var Bar = (function () {
3     function Bar() {
4     }
5     return Bar;
6 })();
7
```

Arrow Function Expressions (lambdas)

```
var getName = () => 'John Smith';

console.log(`The name is ` + getName());
```

TypeScript | Select... | Share

```
1 var getName = () => 'John Smith';
2 console.log(`The name is ` + getName());
```

Run | JavaScript

```
1 var getName = function () { return 'John Smith'; };
2 console.log("The name is " + getName());
3
```

Arrow Functions fix the `this` problem

```
function StockQuoteGeneratorArrow(symbol: string){

    this.symbol = symbol;

    setInterval(() => {

        console.log("StockQuoteGeneratorArrow. The price quote for "
                        + this.symbol  + " is " + Math.random());

    }, 1000);

}
```

Classes

TypeScript → JavaScript (E5)

```typescript
class Person {
    firstName: string;
    lastName: string;
    age: number;
    ssn: string;
}

var p = new Person();

p.firstName = "John";
p.lastName = "Smith";
p.age = 29;
p.ssn = "123-90-4567";
```

```javascript
var Person = (function () {
    function Person() {
    }
    return Person;
})();
var p = new Person();
p.firstName = "John";
p.lastName = "Smith";
p.age = 29;
p.ssn = "123-90-4567";

```

A Class With Constructor

TypeScript → JavaScript (E5)

```typescript
class Person {
    firstName: string;
    lastName: string;
    age: number;
    ssn: string;

    constructor(firstName:string, lastName: string,
        age: number, ssn: string) {

        this.firstName = firstName;
        this.lastName;
        this.age = age;
        this.ssn = ssn;
    }
}

var p = new Person("John", "Smith", 29, "123-90-4567");
```

```javascript
var Person = (function () {
    function Person(firstName, lastName, age, ssn) {
        this.firstName = firstName;
        this.lastName;
        this.age = age;
        this.ssn = ssn;
    }
    return Person;
})();
var p = new Person("John", "Smith", 29, "123-90-4567");

```

Inheritance

Classical syntax → Prototypal

TypeScript | Select... | Share

```typescript
class Person {

    constructor(public firstName: string,
        public lastName: string, public age: number,
        private _ssn: string) {
    }
}

class Employee extends Person{

}
```

Run | JavaScript

```javascript
var __extends = this.__extends || function (d, b) {
    for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
    function __() { this.constructor = d; }
    __.prototype = b.prototype;
    d.prototype = new __();
};
var Person = (function () {
    function Person(firstName, lastName, age, _ssn) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this._ssn = _ssn;
    }
    return Person;
})();
var Employee = (function (_super) {
    __extends(Employee, _super);
    function Employee() {
        _super.apply(this, arguments);
    }
    return Employee;
})(Person);
```

Generics

TypeScript | Select... | Share

```ts
class Person {
    name: string;
}

class Employee extends Person{
    department: number;
}

class Animal {
    breed: string;
}

var workers: Array<Person> = [];

workers[0] = new Person();
workers[1] = new Employee();
workers[2] = new Animal();
```

Error

Run | JavaScript

```js
var __extends = (this && this.__extends) || function (d, b) {
    for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
    function __() { this.constructor = d; }
    d.prototype = b === null ? Object.create(b) : (__.prototy
};
var Person = (function () {
    function Person() {
    }
    return Person;
})();
var Employee = (function (_super) {
    __extends(Employee, _super);
    function Employee() {
        _super.apply(this, arguments);
    }
    return Employee;
})(Person);
var Animal = (function () {
    function Animal() {
    }
    return Animal;
})();
var workers = [];
workers[0] = new Person();
workers[1] = new Employee();
workers[2] = new Animal();
```

No Errors

Interfaces as Custom Types

TypeScript | Select... | Share

```typescript
interface IPerson {

    firstName: string;
    lastName: string;
    age: number;
    ssn?: string;
}

class Person  {

    constructor(public config: IPerson) {
    }
}

var aPerson: IPerson = {
    firstName: "John",
    lastName: "Smith",
    age: 29
}

var p = new Person(aPerson);
console.log("Last name: " + p.config.lastName );
```

Run | JavaScript

```javascript
var Person = (function () {
    function Person(config) {
        this.config = config;
    }
    return Person;
})();
var aPerson = {
    firstName: "John",
    lastName: "Smith",
    age: 29
};
var p = new Person(aPerson);
console.log("Last name: " + p.config.lastName);

```

Interfaces and `implements`

```typescript
interface IPayable{

  increasePay(percent: number): void
}

class Employee implements IPayable{

    increasePay(percent: number): void {
        // increase salary
    }
}

class Contractor implements IPayable{

    increasePay(percent: number): void {
        // increase hourly rate
    }
}

var workers: Array<IPayable> = [];
workers[0] = new Employee();
workers[1] = new Contractor();

workers.forEach(worker => worker.increasePay(30));
```

```javascript
var Employee = (function () {
    function Employee() {
    }
    Employee.prototype.increasePay = function (percent) {
        // increase salary
    };
    return Employee;
})();
var Contractor = (function () {
    function Contractor() {
    }
    Contractor.prototype.increasePay = function (percent) {
        // increase hourly rate
    };
    return Contractor;
})();
var workers = [];
workers[0] = new Employee();
workers[1] = new Contractor();
workers.forEach(function (worker) { return worker.increasePay(30); });

```

TypeScript Compiler: tsc

- Install Node.js from nodejs.org. It comes with the **npm.**
- Install the typescript compiler:

  npm install -g typescript

- Compile **main.ts** into **main.js** specifying target language syntax:

  tsc —t ES5 main.ts

- The watch mode allows to auto-compile as file changes:

  tsc -w *.ts

TypeScript Config in tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES5",
    "module": "commonjs",
    "experimentalDecorators": true
  }
}
```

Hello World in Angular with TypeScript

HelloWorldComponent: main.ts

```
import {bootstrap} from 'angular2/platform/browser';
import {Component} from 'angular2/core';

@Component({
  selector: 'hello-world',
  template: '<h1>Hello {{ name }}!</h1>'
})
class HelloWorldComponent {
  name: string;

  constructor() {
    this.name = ‘World!';
  }
}

bootstrap(HelloWorldComponent);
```

In HTML:

<hello-world></hello-world>

HelloWorldComponent: main.ts

```
import {bootstrap} from 'angular2/platform/browser';
import {Component} from 'angular2/core';

@Component({
  selector: 'hello-world',
  template: '<h1>Hello {{ name }}!</h1>'
})
class HelloWorldComponent {
  name: string;

  constructor() {
    this.name = ‘World!';
  }
}

bootstrap(HelloWorldComponent);
```

A browser has to

1. Load Angular framework

2. Load Transpiled main.ts

SystemJS: a Universal Module Loader

- ES6 defines modules, but not the loader
- ES7 should include the System object for loading modules
- SystemJS is a polyfill that loads modules

Index.html Take 1

```
<!DOCTYPE html>
<html>
<head>
  <script src="//npmcdn.com/angular2@2.0.0-beta.6/bundles/angular2-polyfills.js"></script>
  <script src="//npmcdn.com/typescript@1.7.5/lib/typescript.js"></script>
  <script src="//npmcdn.com/systemjs@0.19.8/dist/system.src.js"></script>
  <script src="//npmcdn.com/rxjs@5.0.0-beta.0/bundles/Rx.js"></script>
  <script src="//npmcdn.com/angular2@2.0.0-beta.6/bundles/angular2.dev.js"></script>

  <script>
    System.config({
      transpiler: 'typescript',
      typescriptOptions: {emitDecoratorMetadata: true}
    });
    System.import('main.ts');
  </script>

</head>
<body>
  <hello-world></hello-world>
</body>
</html>
```

Angular from CDN

SystemJS

HelloWorldComponent

Starting a new project with npm

1. Generate *package.json* for your project:
   npm init -y

2. Add Angular dependencies to *package.json*

3. Download dependencies into the dir *node_modules*:
   npm install

4. Install live-server
   npm install live-server -g

package.json

```
{
  "name": "HelloWorld",
  "version": "1.0.0",
   "dependencies": {
    "es6-shim": "0.33.13",
    "es6-promise": "^3.0.2",
    "reflect-metadata": "0.1.2",
    "rxjs": "5.0.0-beta.0",
    "systemjs": "0.19.8",
    "zone.js": “0.5.14”,
    "angular2": "2.0.0-beta.6"
  },
  "devDependencies": {
    "live-server": "^0.8.2",
    "typescript": "^1.7.5"
  }
}
```

Index.html Take 2

Angular's local (after npm install)

```
<!DOCTYPE html>
<html>
<head>
  <script src="node_modules/angular2/bundles/angular2-polyfills.js"></script>
  <script src="node_modules/typescript/lib/typescript.js"></script>
  <script src="node_modules/systemjs/dist/system.src.js"></script>
  <script src="node_modules/rxjs/bundles/Rx.js"></script>
  <script src="node_modules/angular2/bundles/angular2.dev.js"></script>

  <script>
    System.config({
      transpiler: 'typescript',
      typescriptOptions: {emitDecoratorMetadata: true}
    });
    System.import('main.ts');
  </script>
</head>
<body>
  <hello-world></hello-world>
</body>
</html>
```

Demo

Templates

- A place to write HTML
- Rendering is separated from the core framework
- Angular team works with Teleric on rendering for iOS and Android using NativeScript
- Angular team is working on precompiling templates

Unidirectional Binding

From code to template:

<h1>Hello {{ **name** }}!</h1>

<span **[hidden]**="“isZipcodeValid”>Zip code is not valid

From template to code:

<button **(click)**="placeBid()">Place Bid</button>

<input placeholder= "Product name" **(input)**="onInputEvent()">

Two-way Bindings

Synchronizing Model and View:

<input [value]="myComponentProperty"
(input)="onInputEvent($event)>

<input [(ngModel)] = "myComponentProperty">

Dependency Injection

- Angular creates objects, not need to use the **new** operator
- Can inject objects into components via constructors only
- Each component has an **injector**
- **Providers** specify how to inject
- If a component has no provider, Angular checks its parent

ProductService

```
export class Product {
  constructor(
    public id: number,
    public title: string,
    public price: number,
    public description: string) {
  }
}
```

```
export class ProductService {
  getProduct(): Product {
    return new Product( 0, "iPhone 7", 249.99,
                "The latest iPhone, 7-inch screen");
  }
}
```

Injecting ProductService

```
import {Component, bind} from 'angular2/core';
import {ProductService, Product} from "../services/product-service";

@Component({
  selector: 'di-product-page',
  template: `<div>
  <h1>Product Details</h1>
  <h2>Title: {{product.title}}</h2>
  <h2>Description: {{product.description}}</h2>
  <h2>Price: \${{product.price}}</h2>
</div>`,
  providers:[ProductService]
})

export default class ProductComponent {
  product: Product;

  constructor( productService: ProductService) {

    this.product = productService.getProduct();
  }
}
```

Injecting ProductService

```
import {Component, bind} from 'angular2/core';
import {ProductService, Product} from "../services/product-service";

@Component({
  selector: 'di-product-page',
  template: `<div>
  <h1>Product Details</h1>
  <h2>Title: {{product.title}}</h2>
  <h2>Description: {{product.description}}</h2>
  <h2>Price: \${{product.price}}</h2>
</div>`,
  providers:[ProductService]
})

export default class ProductComponent {
  product: Product;

  constructor( productService: ProductService) {

    this.product = productService.getProduct();
  }
}
```

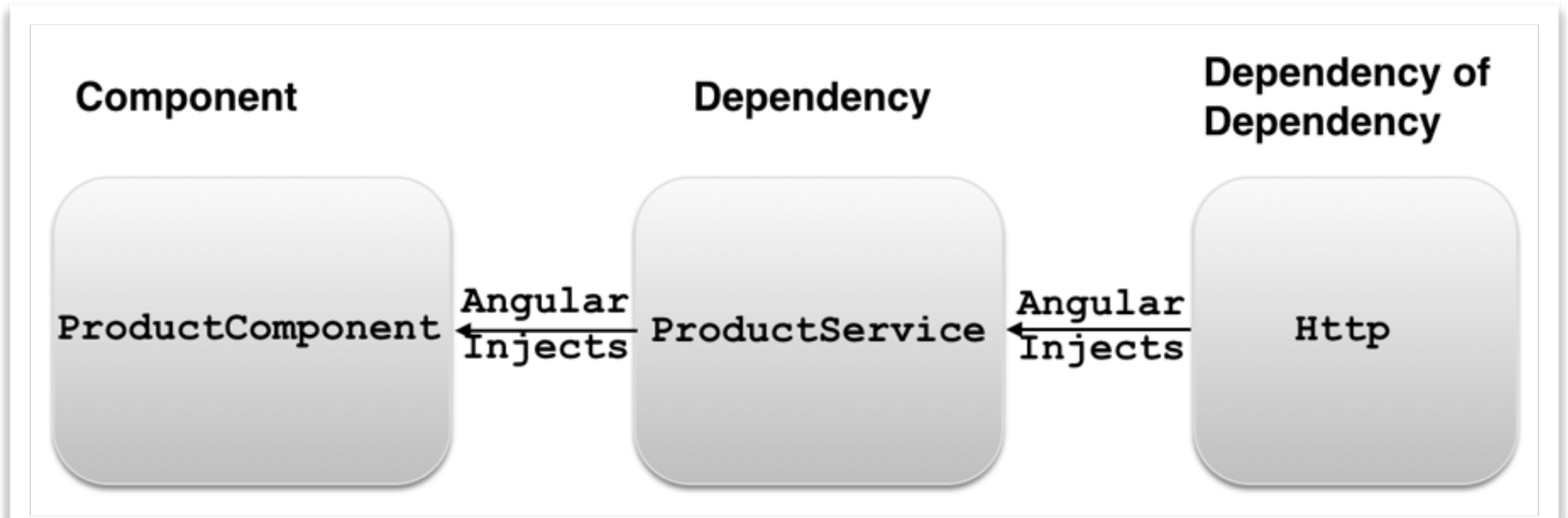
A provider can be a class, factory, or a value

public/private and DI

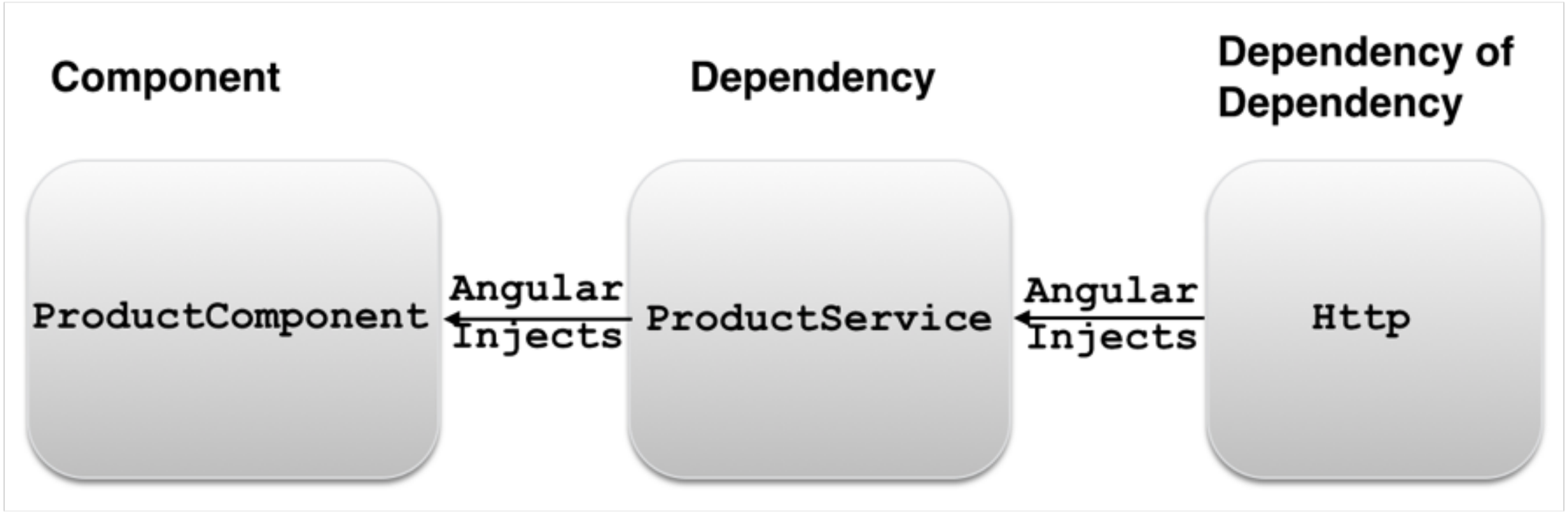
```
export default class ProductComponent {
  constructor( private productService: ProductService) {
  }
}
```

Adding private or public implicitely creates an instance var

Injecting Dependencies of Dependencies



Injecting Dependencies of Dependencies




```
import {Http} from 'angular2/http';

@Injectable
export class ProductService {
    constructor(private http:Http){
        let products = http.get('products.json');
    }
   …
}
```

Routing Directives

- **RouterOutlet** (<router-outlet>) - where the router should render the component

- **@RouteConfig** - map URLs to components to be rendered inside the <router-outlet>

- **RouterLink** (<router>) - a link to a route.

- **RouteParams** - a map of key-value pairs to pass parameters to a component

- **RouteData** - a map of key-value pairs used to pass additional data from @RouteConfig to a route

Basic Routing

```
@Component({
    selector: ‘basic-routing',

    template: `<a [routerLink]="['/Home']">Home</a>
               <a [routerLink]="['/ProductDetail']">Product Details</a>
               <router-outlet></router-outlet>`,
    directives: [ ROUTER_DIRECTIVES]
})

@RouteConfig([
    {path: '/',         component: HomeComponent, as: 'Home'},
    {path: '/product', component: ProductDetailComponent, as: 'ProductDetail'}
])
class RootComponent{
}

bootstrap(RootComponent, [ROUTER_PROVIDERS, provide(LocationStrategy,
{useClass: HashLocationStrategy})]);
```

Passing Data to a Route

```
@Component({
    selector: 'basic-routing',
    template: `<a [routerLink]="['/Home']">Home</a>
              <a [routerLink]="['/ProductDetail',
                                {id:1234}]">Product Details</a>
              <router-outlet></router-outlet>`,
    directives: [ ROUTER_DIRECTIVES]
})
@RouteConfig([
    {path: '/',            component: HomeComponent, as: 'Home'},
    {path: '/product/:id', component: ProductDetailComponent,
                                                    as: 'ProductDetail'}

])
class RootComponent{}

bootstrap(RootComponent, [ROUTER_PROVIDERS,
   provide(LocationStrategy, {useClass: HashLocationStrategy})]);
```

Receiving Data in a Route

```
@Component({
    selector: ‘product',

    template: `<h1 class="product">Product Detail for Product:
                                    {{productID}}</h1>`,

    styles: ['product {background: cyan}']
})
```

3

```
export class ProductDetailComponent {
    productID: string;

    constructor(params: RouteParams){

        this.productID = params.get('id');
    }
}
```

1

2

Observable Streams

- A observable stream emits the data over time to the observer (subscriber).
- Angular uses RxJS
- Multiple operators can transform the stream's data
- Stream samples: UI events, HTTP responses, data arriving over websockets

Observable Streams

A observable stream can:

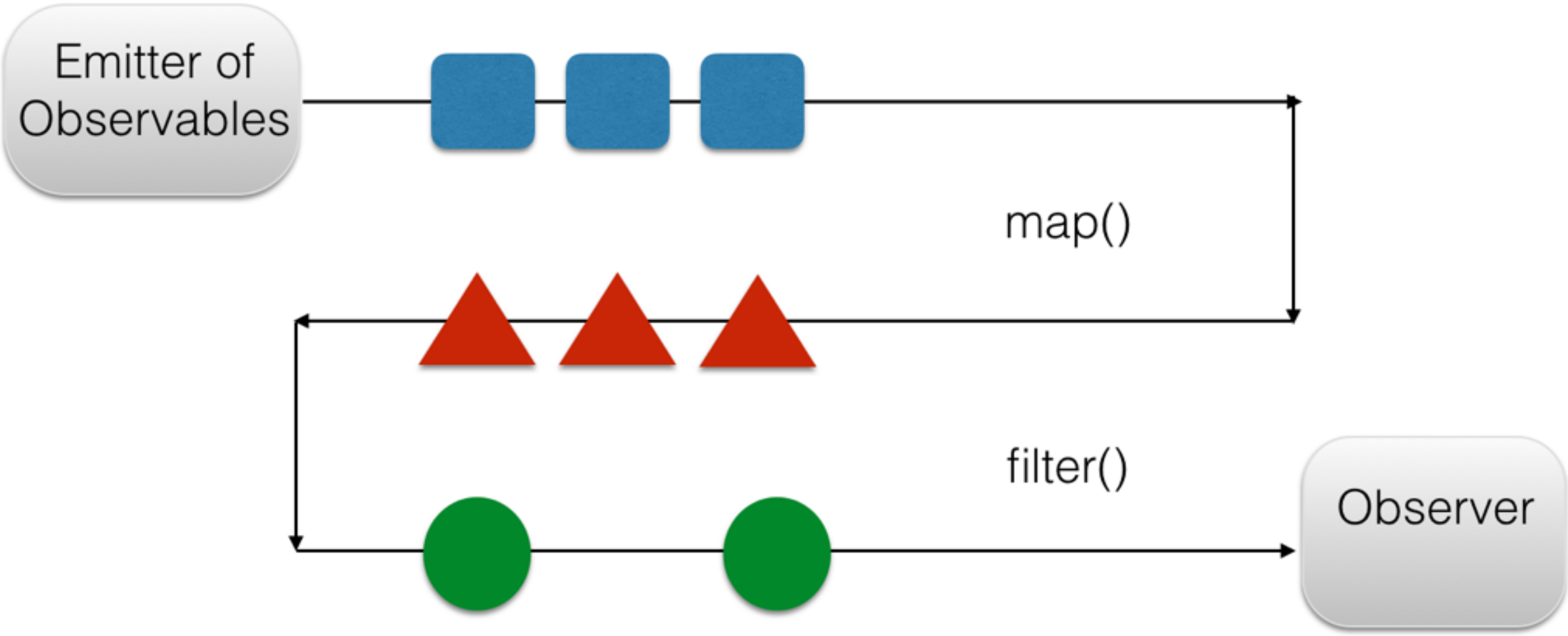
- Emit the next element
- Throw an error
- Send a signal that the streaming is over

Observers

A observer provides:

- A function to handle streaming object
- Error handling
- End-of-stream handling

Transforming the stream

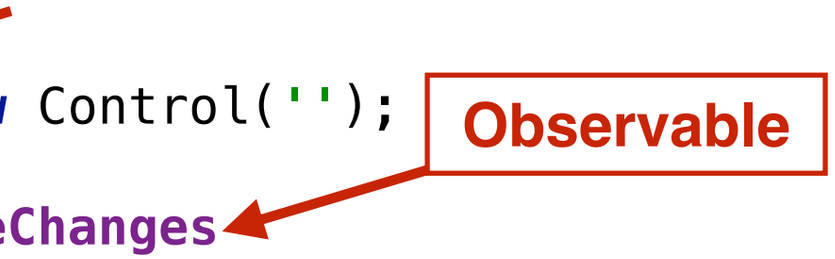


Observables vs Promises

- Observable can return multiple values
- Observables can be cancelled
- Observables allow to handle end-of-stream (similar to `finally` in Java

Observable Events

```
@Component({
    selector: "app",
    template: `
       <h2>Observable events demo</h2>
      <input type="text" placeholder="Enter stock" [ngFormControl]="searchInput">
    `
})
class AppComponent {

    searchInput: Control;

    constructor(){
        this.searchInput = new Control('');

        this.searchInput.valueChanges
            .debounceTime(500)
            .subscribe(stock => this.getStockQuoteFromServer(stock));
    }

    getStockQuoteFromServer(stock) {

        console.log(`The price of ${stock} is ${100*Math.random().toFixed(4)}`);
    }
}
```



Http and Observables

```
class AppComponent {

  products: Array<string> = [];

  constructor(private http: Http) {

    this.http.get('http://localhost:8080/products')
        .map(res => res.json())
        .subscribe(
          data => {
                this.products=data;
            },
           err =>
              console.log("Can't get products. Error code: %s, URL: %s ",
                                                    err.status, err.url),
           () => console.log('Product(s) are retrieved')
        );
  }
}
```

DI

Demo

Preparing deployment with Webpack

- Input: your app (multiple .ts, .js files)
- Output: transpiled bundle (a .js file)
- Resources (css, images, html) can be inlines in the bundle

Preparing deployment with Webpack

- Input: the entry point of your app
- Output: transpiled bundle (a .js file)
- Resources (css, images, html) can be inlined in the bundle
- Usually, the app will have at least two bundles:

  - your code (e.g. **bundle.js**)

  - frameworks and libraries (e.g. **vendor.bundle.js**)

Webpack Loaders & Plugins

- Loaders operate on a single file (e.g. transpile TS into JS)
- Plugins can operate on multiple files and be invoked at different stages of the Webpack lifecycle

webpack.config.js for dev

```
…
module.exports = {
  debug: true,
  devServer: {
    contentBase: 'src',
    historyApiFallback: true,
    host: metadata.host,
    port: metadata.port
  },
  devtool: 'source-map',
  entry: {
    'main'  : './src/main.ts',
    'vendor': './src/vendor.ts'
  },
  module: {
    loaders: [
      {test: /\.css$/,  loader: 'raw'},
      {test: /\.html$/, loader: 'raw'},
      {test: /\.ts$/,   loader: 'ts'}
    ],
    noParse: [path.join(__dirname, 'node_modules', 'angular2', 'bundles')]
  },
  output: {
    path    : './dist',
    filename: 'bundle.js'
  },

  plugins: [
    new CommonsChunkPlugin({name: 'vendor', filename: 'vendor.bundle.js', minChunks: Infinity}),
    new CopyWebpackPlugin([{from: './src/index.html', to: 'index.html'}]),
    new DefinePlugin({'webpack': {'ENV': JSON.stringify(metadata.ENV)}})
  ],
  resolve: {
    extensions: ['', '.ts', '.js']
  }
};
```

Dev Server

Entry points

Loaders

Output

Plugins

webpack.config.js for prod

```
…
module.exports = {
  debug: false,
  devtool: 'source-map',
  entry: {
    'main'  : './src/main.ts',
    'vendor': './src/vendor.ts'
  },
  metadata: metadata,
  module: {
    loaders: [
      {test: /\.css$/,  loader: 'to-string!css'},
      {test: /\.html$/, loader: 'raw'},
      {test: /\.ts$/,   loader: 'ts'}
    ],
    noParse: [path.join(__dirname, 'node_modules', 'angular2', 'bundles')]
  },
  output: {
    path    : './dist',
    filename: 'bundle.js'
  },
  plugins: [
    new CommonsChunkPlugin({name: 'vendor', filename: 'vendor.bundle.js', minChunks: Infinity}),
    new CompressionPlugin({regExp: /\.css$|\.html$|\.js$|\.map$/, threshold: 1500}),
    new CopyWebpackPlugin([{from: './src/index.html', to: 'index.html'}]),
    new DedupePlugin(),
    new DefinePlugin({'webpack': {'ENV': JSON.stringify(metadata.ENV)}}),
    new OccurenceOrderPlugin(true),
    new UglifyJsPlugin({
      compress : {screw_ie8 : true},
      mangle: false
    })
  ],
  resolve: {
    extensions: ['', '.ts', '.js']
  }
};
```

More plugins than in dev

npm scripts in package.json

```
"scripts": {

  "clean": "rimraf dist",

  "postinstall": "typings install",

  "prebuild": "npm run clean",
  "build": "webpack --config webpack.prod.config.js --progress -profile --display-cached",

  "start": "webpack-dev-server --inline --progress --display-cached --port 8080",

  "preserve:dist": "npm run build",
  "serve:dist": "static dist -z"
}
```

To run a script from the command line:

```
npm start
```

or

```
npm run build
```

or

```
npm run serve:dist
```

Demo

Can I upgrade my app from A1 to A2?

\

Can I upgrade my app from A1 to A2?

Source: https://www.youtube.com/watch?v=MsetYODpj8E

Can I upgrade my app from A1 to A2?

Everything's possible!

Upgrade strategy: http://bit.ly/1KpJrYS

Links

- Code examples: https://github.com/Farata/angular2typescript
- Angular consultng/training: faratasystems.com
- Twitter: @yfain
- Blog: yakovfain.com