Angular 2 Fundamentals

Training Date

Getting to know...

- Your Name
- Background
  - School
  - Degree
  - Work Experience
  - Role/Position
- Expectation

Expectations / Ground Rules

- Start on time. Stay on time. Stop on time.
- Participate in the discussion, ask questions.
- Attendance is important.
- Cell Phones on silent mode.

Module 1:
TypesScript Basics

What is Typescript?

- Typescript is a typed superset of Javascript that compiles to plain Javascript – typescriptlang.org

Why use Typescript?

- Problem 1: Dynamic Data Types
- Problem 2: Modular Programming in Javascript

Problem 1: Javascript Dynamic Types

- Pros
  - Variables can hold any object
  - Types determined on the fly.
  - Implicit type coercion (ex: string to number)
- Cons
  - Types needs to be tested
  - Not all developers use the ===
  - Can be difficult to maintain especially for Enterprise-scale apps.

Problem 2: Server-Side to Client Side

- Developers coming from other languages such as JAVA,.NET and other OOP might have some difficulty coding in Javascript.

- Concepts such as Classes, Constructors,Interfaces and other OOP features are not directly supported in Javascript.

Typescript Alternatives

- Pure Javascript(Javascript Patterns)
- CoffeeScript
- ECMA
- DART

Key Typescript Features

- Supports standard Javascript code
- Static Typing
- Encapsulation Through classes and modules
- Support for constructors, properties, functions
- Define interfaces
- Support for arrow functions (lambdas) =>
- Intellisense and syntax checking

Typescript Compiler

```
class Greeter {
    greeting:string;
    constructor(message:string){
        this.greeting = message;
    }
    greet(){
        return "Hello, " + this.greeting;
    }
}
```

greeter.ts

tsc greeter.ts

```
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
```

greeter.js

Typescript Syntax Rules

- Typescript is a Superset of Javascript
- Follows the same syntax rules as Javascript
  - { } brackets define code blocks
  - Semi-colon end code expressions
- Same keywords in Javascript
  - for
  - If

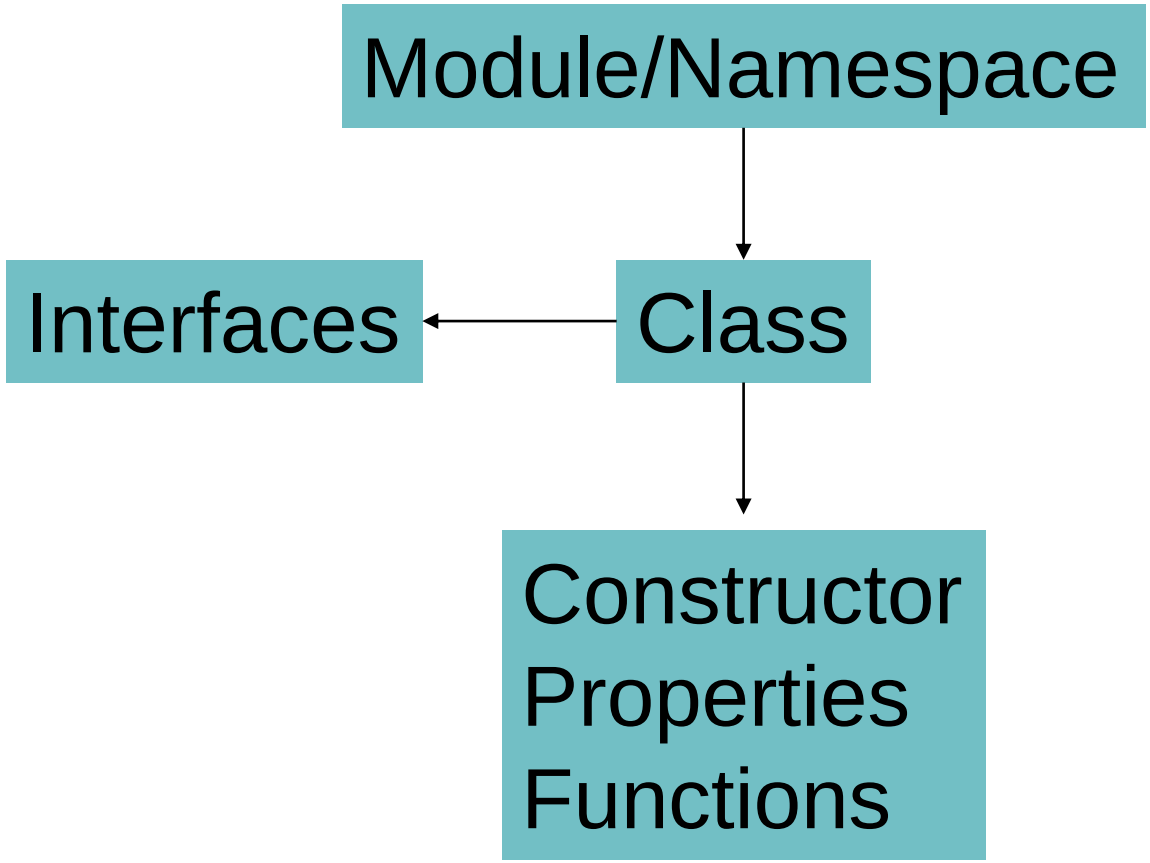
Important Keywords

Keyword	Description
class	Container for members such as properties and functions
constructor	Provides initialization functionality in a class
exports	Export a member from a module
extends	Extend a class or interface
implements	Implements an interface
imports	Imports a module
interface	Defines code contract that can be implemented by types
module/namespace	Container for classes and other code

Cont. Keywords

public/private	Member visibility modifiers
...	Rest parameter syntax
=>	Arrow syntax used with definitions and functions
<typeName>	< > characters use to cast/convert between types
:	Separator between variable/parameter names and types

Typescript Code Hierarchy



Tooling/Frameworks Support

Node.js

The command-line TypeScript compiler can be installed as a Node.js package.

INSTALL

```
npm install -g typescript
```

COMPILE

```
tsc helloworld.ts
```

Visual Studio

- Visual Studio 2015
- Visual Studio 2013
- Visual Studio Code

And More...

- Sublime Text
- Emacs
- Atom
- WebStorm
- Eclipse
- Vim

Typescript using the Playground

- Go to https://www.typescriptlang.org/play/

TypeScript | Documentation | Samples | Download | Connect | Playground | Fork me on GitHub

TypeScript 2.1 is now available. Download our latest version today!

Using Classes | TypeScript | Share | Options | Run | JavaScript

```typescript
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

let greeter = new Greeter("world");

let button = document.createElement('button');
button.textContent = "Say Hello";
```

```javascript
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
var greeter = new Greeter("world");
var button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function () {
    alert(greeter.greet());
```

Typescript using NodeJS

```
class HelloWorld{
    message:string;
    constructor(message:string){
        this.message = message;
    }

    greet(){
        console.log("Hello " + this.message);
    }
}

var firstApp = new HelloWorld("John Wick");
firstApp.greet();
```

helloworld.ts

Node.js

The command-line TypeScript compiler can be installed as a Node.js package.

INSTALL

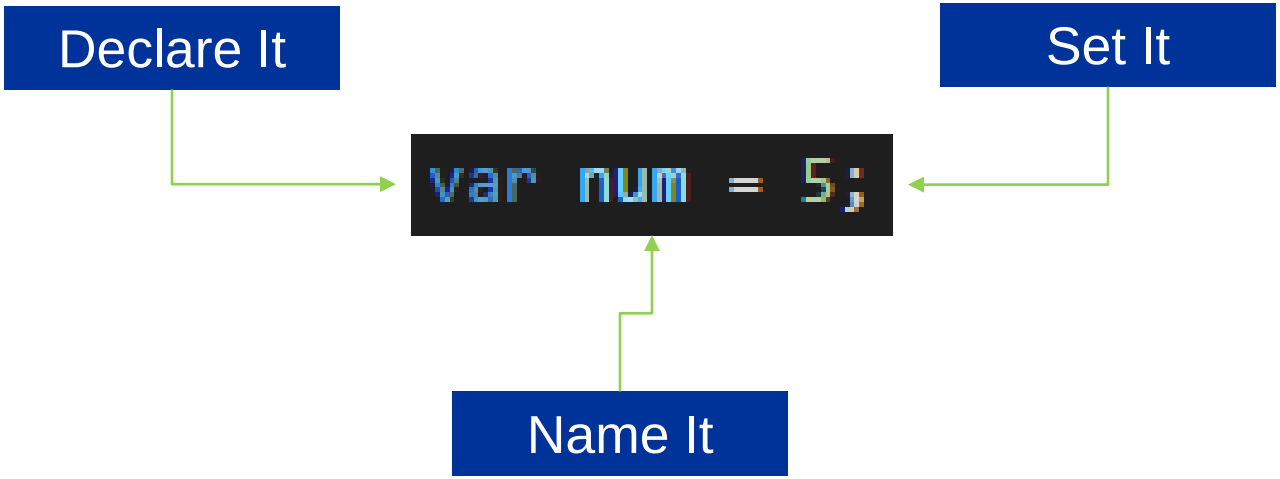
```
npm install -g typescript
```

COMPILE

```
tsc helloworld.ts
```

Typescript Grammar (Declarations & Annotations)

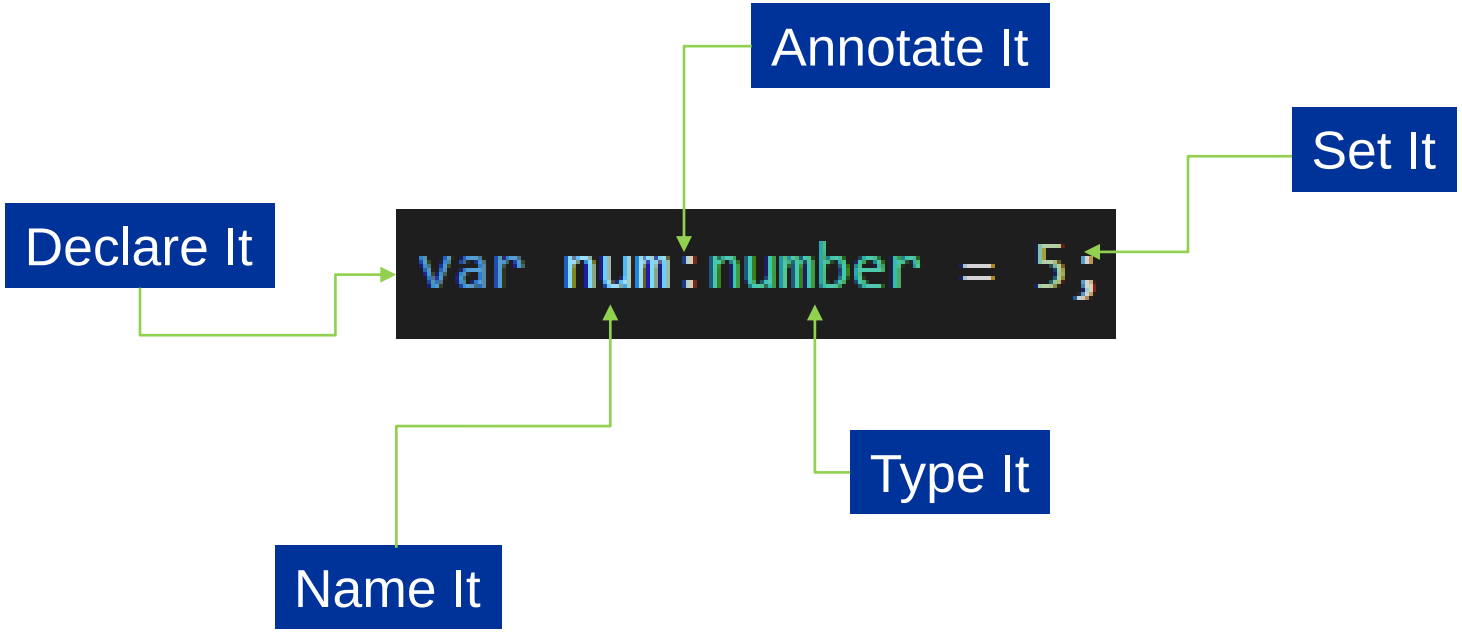
- Type Inference



Typescript using Javascript syntax

Typescript Grammar (Declarations & Annotations) Cont.

- Type Annotations



Examples: Inferences and Annotations

```
var morphius;
var num1:number;
var num2:number = 5;
var num3 = 3;
var num4 = num3 + 200;
var str1 = num1 + ' whats up!';
var wronghere:number = num1 + 'not supposed to be here';
```

- `var morphius;` — Can hold any type(dynamic)
- `var num1:number;` — Type annotation (number)
- `var num2:number = 5;` — Type annotation with value
- `var num3 = 3;` — Type Inference (number)
- `var num4 = num3 + 200;` — Type Inference (number)
- `var str1 = num1 + ' whats up!';` — Type Inference (string)
- `var wronghere:number = num1 + 'not supposed to be here';` — Error

The “any” Type

- Represents any Javascript value

```
var data:any;
var data2;
```

Primitive Types

```
var age:number = 21;
var price=12;

var isLogin:boolean = true;
var isReady = false;

var fname:string ="John";
var lname = "Doe";

var team:string[] = ['Kobe','Jordan','Shaq','Malone','Steph'];
```

Primitive Types - Null

```
var num:number = null;
var str:string = null;
var isPromoted:boolean=null;
var person:{} = null;
```

Primitive Types - undefined

```
var score:number;
var points = undefined;
```

Object Types

```
//object literal
var square = {h:10,w:20};
var point:Object = {x:10,y:20};

//Functions
var sum = function(x:number){
    return x+x;
}

var sumAgain:Function;
sumAgain = function(x:number){
    return x+x;
}
```

Typescript ES2015 Feature Support(Optional Parameters)

```
//optional parameters
function myFunc(param1:any,param2:any=5,param3:any="typescript"){
    console.log("param1=",param1);
    console.log("param2=",param2);
    console.log("param3=",param3);

}
myFunc("cool");
```

With optional paramters, you can specify default parameters.

Typescript ES2015 Feature Support (Template Strings)

```
//template strings
var todo = {
    id:123,
    name:"Walk the dog",
    completed:true
}
var dispTasks = `Task ${todo.id}`;
```

Enclose the string in Backquote simble beside the number 1 key on your keyboard

With template string, you can combine strings and expressions in a single value

Typescript ES2015 Feature Support (let and const)

```
//let and const
for(var i=0;i<3;i++){
    var ctr=i;
}

//this would still print 3
console.log(ctr);
```

```
//let and const
for(var i=0;i<3;i++){
    let ctr=i;
}

//typescript would report that
//it cannot find ctr
console.log(ctr);
```

```
const fname:string="John";
fname="Jane;"
```

Typescript ES2015 Feature Support (for of loop)

```
var tasks =[
    "Walk the dog",
    "Do the laundry",
    "Clean the house"
    ];

    for(var task in tasks){
        console.log(task);
        //this would print the index, not the value
    }

    for(var task of tasks){
        console.log(task)
        //this would print the value
    }
```

Typescript ES2015 Feature Support (arrow functions)

```
//arrow function (lambda)
function myFunc(e:any){
     e.addEventListener('click',
      ()=>{
          console.log("arrow function");
      }
     )
}

var filter = [1,2,3].filter(x=>x>2);
```

Typescript ES2015 Feature Support (spread operator)

```
//the spread operator
function add(...values:any[]){
    var total=0;
    for (var value of values){
        total+=value;
    }

    return total;
}
```

Functions

- Parameter types (required and optional)
- Arrow functions (lambda)
  - Compact form of function expression
  - Omit the function keyword
  - Have scope of “this”
- Void
  - Used for functions that return no value.

Functions Example

```
//Simple function
var squareSimple =  function(h:number,w:number){
    return h*w;
}

//Arrow functions (lambda)
var squareLambda = (h:number,w:number) => h*w;

//optional parameters
var helloWorld :(name?:string)=> void;
helloWorld = (name?:string)=>{
    console.log('Hello ' + (name || ' uknown person'));
}

helloWorld();
helloWorld('John Doe');
```

Omit the function keyword

Return statement

? Means optional

Classes

- Classes in Typescript act as containers for different components/members that are related such as:
  - Fields
  - Constructors
  - Properties
  - Functions

Defining Classes

```
class Vehicle {
    //1. Fields

    //2. Constructor

    //3. Properties

    //4. Functions

}
```

Defining Classes Cont.

```
class Vehicle {
    //1. Fields
    engine:string;
    private _model:string;
    //2. Constructor
    constructor(engine:string){
        this.engine = engine;
    }
    //3. Properties
    get model():string{
        return this._model;
    }
    set model(model:string){
        //set additional logic/filters
        if(model===undefined) throw " You need to supply a model";
        this._model = model;
    }
    //4. Functions
    start():string{
        return "Started " + this.engine;
    }
    stop():string{
        return "Stopped " + this.engine;
    }
}
```

Instantiating a Type

```
var vehicle = new Vehicle('v8');
vehicle.model = "Audi";
vehicle.start();
```

Extending a Type

Types can be extended using the "extends" keyword

```
class BMW extends Vehicle{
    constructor(){
        super('v8');
    }
}
```

Defining Interfaces

- Interfaces provide a way to define a contract that other objects should implement

```
interface IEngine{
    start():void;
    stop():void;
}

class Engine implements IEngine{
    start():void{
        console.log("Engine is starting");
    };
    stop():void{
        console.log("Engine is stoping");
    }
}
```

Interfaces can be used using the "implements" keyword

Extending Interfaces

```
interface IEngine{
    start():void;
    stop():void;
}

interface IPlaneEngine extends IEngine{
    fly():void;
    glide():void;
}
```

```
class Truck implements IEngine{
    start():void{
       console.log("Engine is starting");
    };
    stop():void{
        console.log("Engine is stoping");
    }
}

class Jet implements IPlaneEngine{
    start():void{
       console.log("Jet engine is starting");
    }
    stop():void{
      console.log("Jet engine is stoping");
    }
    fly():void{
        console.log("Jet  is flying");
    }
    glide():void{
         console.log("Jet is gliding");
    }
}
```

Modules/Namespace

- Modules helps you organize your code base on its role/responsibilities.
- Makes your code more maintainable, testable and reusable.

Internal –Named Module

```
namespace Shapes{
    class Square {
        height:number;
        width:number;
        constructor(height:number,width:number){
            this.height = height;
            this.width = width;
        }
    }
    var square = new Square(5,5);
}

var mySquare = new Shapes.Square(5,5);
```

Will not work outside the Namespace

Exporting Internal Modules

```
namespace Shapes{
    export class Square {
        height:number;
        width:number;
        constructor(height:number,width:number){
            this.height = height;
            this.width = width;
        }
    }

}

var mySquare = new Shapes.Square(5,5);
```

Use the “export” keyword to expose classes inside a namespace

Now this will work outside the Namespace

Extending Internal Modules

```
namespace Shapes{
    export class Square {
        height:number;
        width:number;
        constructor(height:number,width:number){
            this.height = height;
            this.width = width;
        }
    }

}

var mySquare = new Shapes.Square(5,5);

namespace Shapes{
    export class Circle{
        radius:number;
        constructor(radius:number){
            this.radius=radius;
        }
    }
}

var myCircle = new Shapes.Circle(23);
```

Separating Internal Modules

- Namespaces/Modules across files. Ideal for large projects
- Important to load them in the proper sequence
  - Script tags
- Reference them
  - ///<reference path="[ts file]"/>

Separating Internal Modules

```
namespace Shapes{
    export class Square {
        height:number;
        width:number;
        constructor(height:number,width:number){
            this.height = height;
            this.width = width;
        }
        getArea():number{
            return this.height * this.width;
        }
    }

}
```

shapes.ts

Step 1. Export the component

Step 2. Reference the file
///<reference path="[ts file]"/>

Step 3. Access the component

```
///<reference path="shapes.ts"/>

namespace shapeManager{
    var mySquare2 = new Shapes.Square(5,8);
    console.log(`Area is = ${mySquare2.getArea()}`);

}
```

shapeManager.ts

External Modules

- Gives you the ability to load modules separately
- Allows you to “import” exported entities into other modules.
- Better solution than remembering the sequence of javascript files for dependencies.

Defining External Modules

- Step 1: Configure your project to use External Modules in the tsconfig.json

```
{
    "compilerOptions": {
        "module": "amd",
        "target": "es5",
        "outDir": "scripts"
    }

}
```

Add the module option with either “amd” , “commonjs” or es2015 value

Defining External Modules (Cont.)

- Step 2: Remove the “Namespace” keyword from the ts files.

shapes.ts

```
export class Square {
    height:number;
    width:number;

    constructor(height:number,width:number){
        this.height = height;
        this.width = width;
    }
    getArea():number{
        return this.height * this.width;
    }
}
```

No more “Namespace” keyword. The file itself will be the module boundary in this example “shape.ts”.

Defining External Modules (Cont.)

- Step 3: Use the “import” statement to access the exported components.

shapeManager.ts

```
import myShapes = require("./shapes");
namespace shapeManager{
    var myShapes = new myShapes.Square(5,8);
    console.log(`Area is = ${myShapes.getArea()}`);

}
```

You can also use the ES2015 syntax for importing module

import {myShapes} from ‘./shapes’

To access the Square class from shapes.ts, use the import statement and specify the name of the ts file omitting the file ext. “ts”.

ECMAScript 2015

- Not widely supported yet
- No standard definition for loading modules
- You have to use “Module Loader”
  - System.js
  - requireJS

Using requireJS

- Install requireJS using node package manager
  - npm install requirejs
- Reference requirejs in the script tag and define the main file to run

```
<!doctype html>
<html>
    <head>
        <title>ES6 features</title>
    </head>
    <body>
        <div id="container">
        </div>
      <script data-main="scripts/shapeManager"
        src="/node_modules/requirejs/require.js"></script>

        </body>
</html>
```

Using the data-main attribute, specify the main js file to run

Reference require.js

Using Systemjs

- Install systemjs using node package manager
  - npm install systemjs
- Reference systemjs in the script tag and define the main file to run

```
  <script src="systemjs.config.js"></script>
  <script>
    System.import('main.js').catch(function(err){ console.error(err); });
  </script>
</head>
```

Module 2:
Angular2 Fundamentals

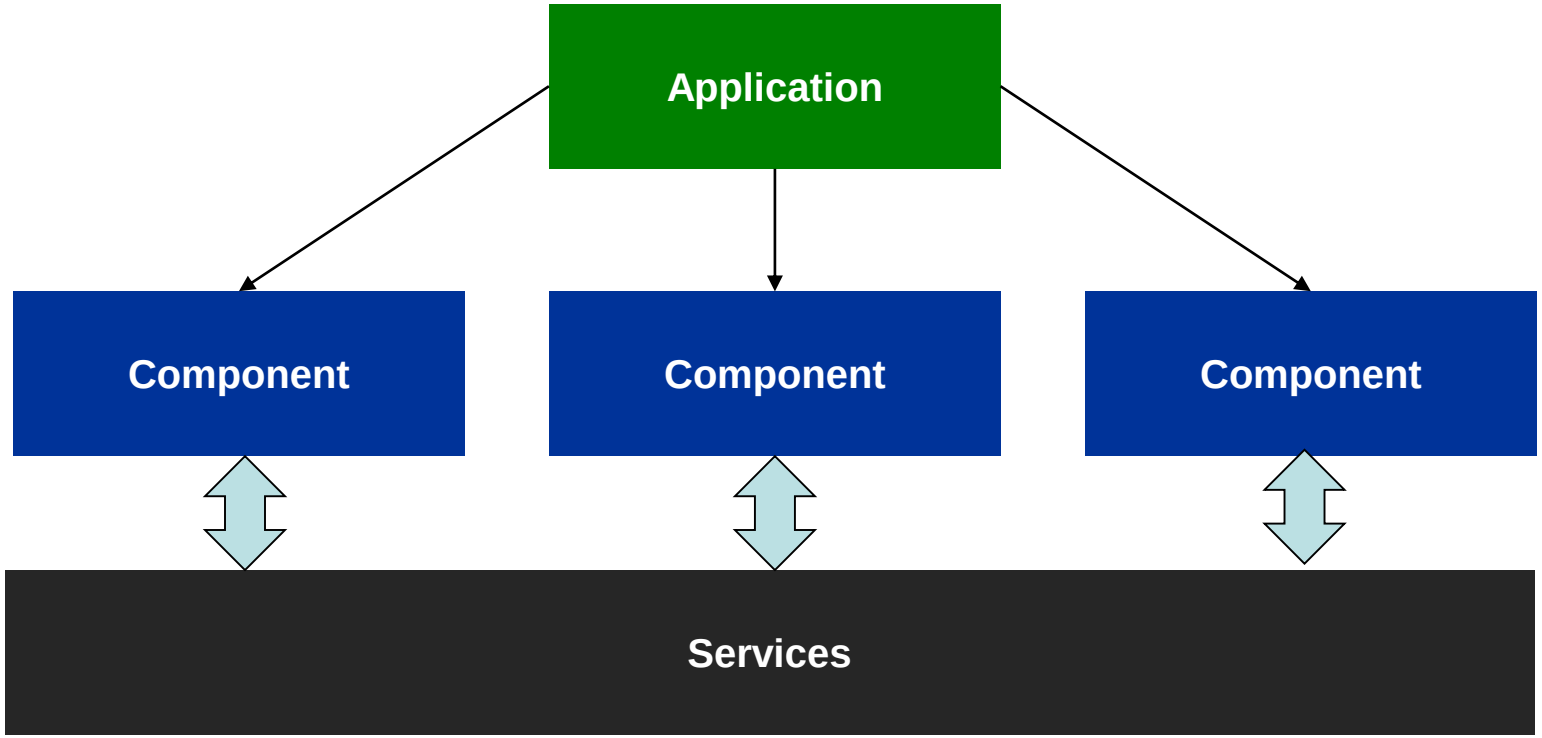
What is Angular?

- It is a Javascript framework for building client side applications using HTML,CSS and Javascript
- A platform that supports multiple languages(ES5,Typescript),Mobile Web, Native Mobile Apps
- Main features
  - Expressive HTML
  - Powerful Databinding
  - Modular By Design
  - Built-in Back-End Integration

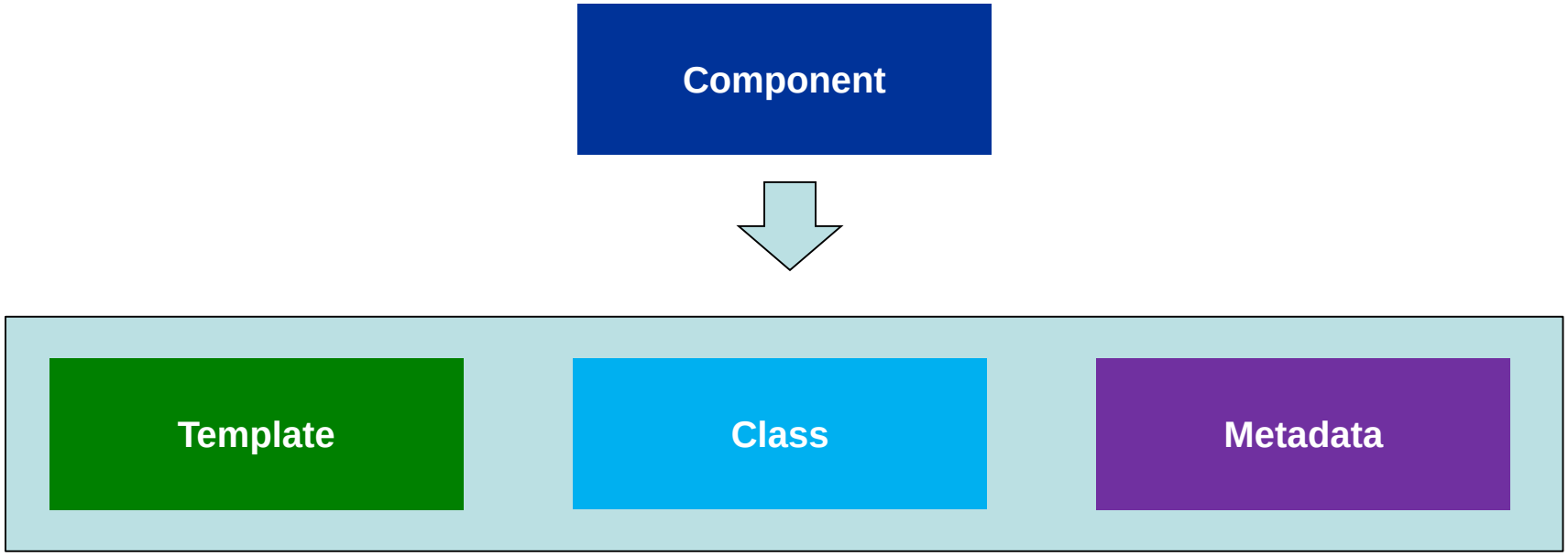
Why Angular 2

- Built for Speed
- Modern(uses the latest features of the ECMAScript standard)
- Simplified API
- Enhances productivity

Anatomy of an Angular 2 Application



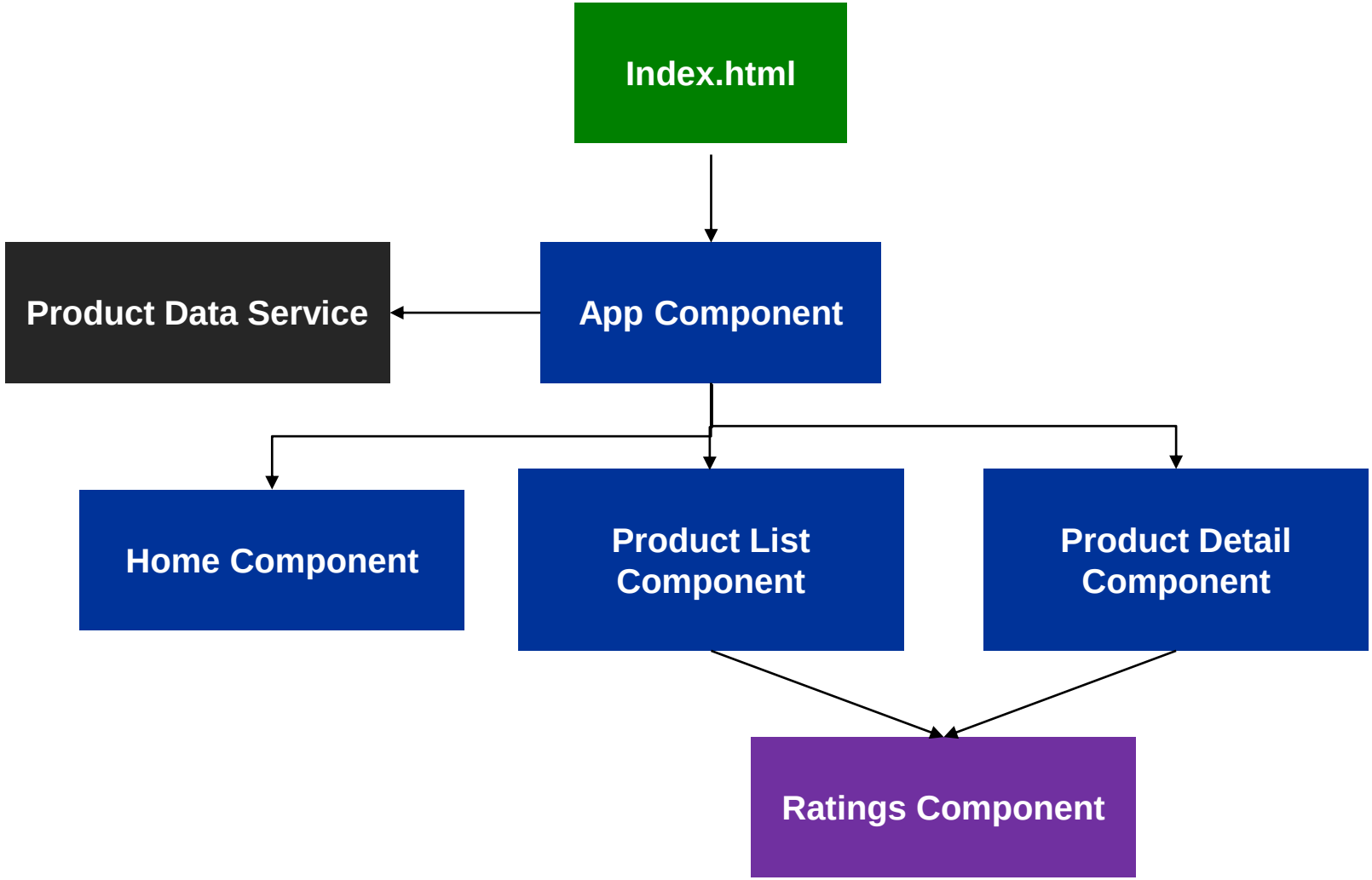
What is a Component



Mini Project (ToyLand)

- ToyLand is the leading manufacturer and distributor of toys in the world. Estimated revenue in 2016 was around 5 billion dollars. As part of their strategy for coming year, management wants to expand their presence by having their own ecommerce platform that is both web and mobile ready. They wanted to test the market first and decided to develop a web based product catalog system to showcase their new products and to get market insights.

Sample App Architecture



Angular 2 Application Setup (Manual)

- Create an application folder
- Create the tsconfig.json file
- Create the package.json file
- Create the typings.json file
- Install the libraries and typings
- Create the host Web page (index.html)
- Create the main.ts file (app entry point)

Angular 2 Application Setup (angular-cli)

- Install the Nodejs v.5 or higher
- Install angular-cli using the node package manager

```
npm install -g angular-cli
```

- Create a new project

```
ng new my-app
```

- Serve the application

```
ng-serve
```

Go to http://localhost:4200 to check

Angular 2 Application Setup (quickstart app)

```
git clone https://github.com/angular/quickstart.git quickstart
cd quickstart
npm install
npm start
```

Angular 2 App Structure

- src
  - app
    - app.component.ts
    - app.module.ts
  - main.ts
  - index.html

Angular2 App Structure Cont.

- app.components.ts
  - The root component which will be the main component to be called when the app starts.
- app.module.ts
  - The root module that tells how Angular how to assemble the application.
- main.ts
  - The entry point. Compiles the apps main module to run in the browser.
- index.html
  - HTML page that will host the app.

Angular 2 Modules

- Angular 2 uses the ES2015 standard modules
- In ES2015, the file is the module.

recipe.ts

```
export class Recipe{

}
```

recipelist.ts

```
import {Recipe} from './recipe'
```

Common Angular2 Modules

@angular/core

@angular/http

@angular/common

@angular/router

Module 3:
Angular 2 Components

Defining Components

Import the Component function from the 'angular core' module

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

@Component({
  selector: 'app-root',
  template: '<div><h1>{{title}}</h1><h1>My First Component</h1>',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app works!';
}
```

Metadata
Component Decorator

Directive name used in HTML

View Layout

Binding

Bootstrapping the App Component

- Step 1: Host the application (index.html)
- Step 2 : Load the root module (main.ts)

Hosting the Application

app.component.ts

```
import {Component} from '@angular/core';
@Component({
  selector:'pm-app',
  template:'<div><h1>{{pageTitle}}</h1><div>My First Component</div></div>'

})

export class AppComponent{
    pageTitle:string = "RecipeRUs";
}
```

template get inserted inside the selector

index.html

```
<body>
    <pm-app>Loading the app...</pm-app>

</body>
```

Bootstrap the Application

```
<script>
  System.import('main.js')
</script>
```

index.html

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule);
```

main.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent }  from './app.component';

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

app.module.ts

Defining a View for the Component

```
template:"<h1>{{pageTitle}}"
```

Inline Template

```
template:`
<div>
  <h1>{{pageTitle}}</h1>
  <div>
    My First Component
  </div>
</div>
`
```

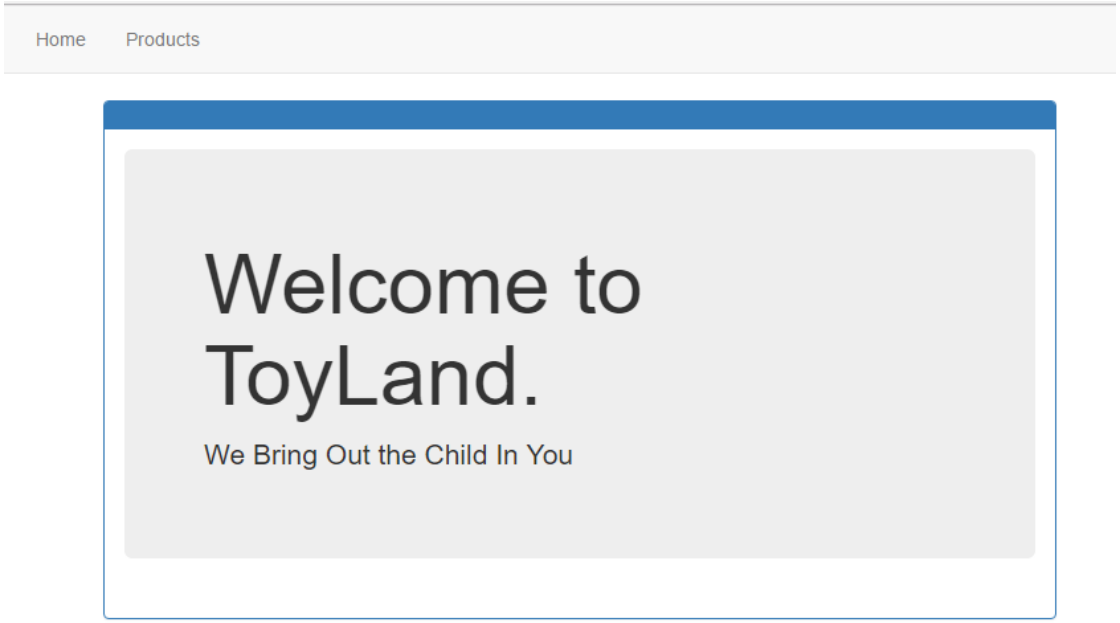
Inline Template multiline using es2015 template strings (back ticks)

```
templateUrl:'./product.component.html'
```

Linked Template

Sprint 1 Creation of the Home Component

- For the first sprint , you task is to develop the Home Component for Product Catalog System. It will be the initial page that will be shown when a user visits the site.



Module 4: Data Binding

Binding

- Coordinates communication between the component's class and its template, often involves passing data.

One way binding (Interpolation)

```
<h1>Hello {{name}}</h1>
```

Template expression
{{ exp }}

```
export class AppComponent {
  name = 'Angular2';
}
```

Property Binding

```
toys:any[]=[
    {
    name:"GI Joe",
    code:'toy101',
    instock:'March 2017',
    price:33,
    rating:5,
    imgUrl:'images/gi.png'

    },
```

```
<img [src]='toy.imgUrl'/>
```

Element property Binding target [ ]

Template expression Binding source ‘ ’

Event Binding

```
<button (click)='toggleImage()' class='btn btn-primary'>
```

```
toggleImage():void{
    alert('something happend');
}
```

Target event

Component Method

Two-way binding

```
<input [(ngModel)]='filter' placeholder="filter"/>
```

```
export class RecipeListComponent{
    filter:string="filtervalue";
```

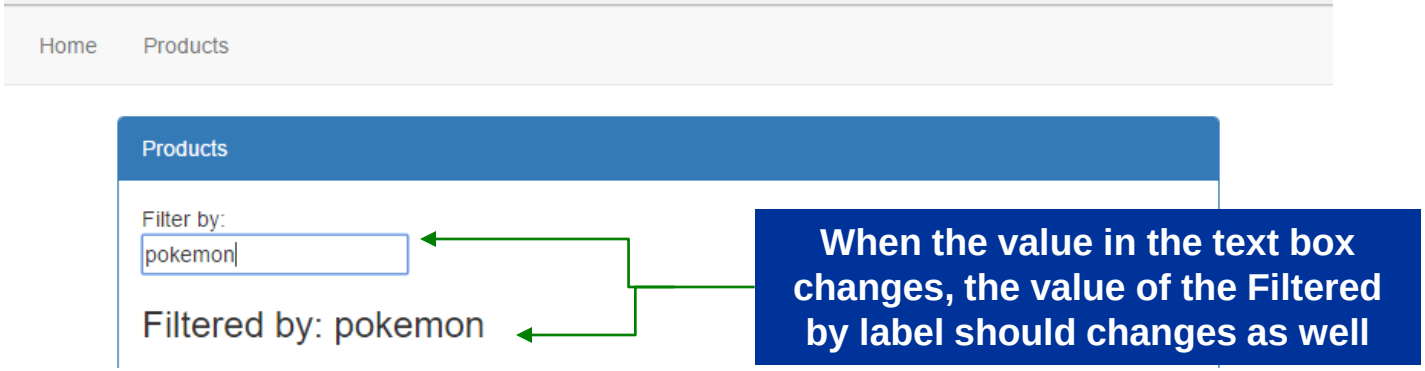
Filter by: pokemon

Filtered by: pokemon

When the value in the text box changes, the value of the Filtered by label changes as well

Sprint 2 Filter Search Binding

- For the second sprint , your tasks are
  - Create the ProductList Component
  - Create of the initial Filter Search feature that when a user types in the search box, it will also show in the page header



Module 5:Directives and Pipes

Directives

- Custom HTML element or attribute used to power up and extend our HTML
  - Custom Directives
  - Angular2 Built-in Directives
    - *ngIf
    - *ngFor

Structural Directives

*ngIf Built-In Directive

```
<div *ngIf='recipes'>
    ...
</div>
```

Contents in the div will show up if the 'recipes' property has a value

*ngFor Built-In Directive

```
toys:any[]=[
    {
    name:"GI Joe",
    price:33
    },
   {
    name:"Pokemon",
    price:55
   },
   {
    name:"Basketball",
    price:45
    },
]
```

```
<tr *ngFor='let toy of toys' >
    <td>{{toy.name}}</td>
    <td>{{toy.price}}</td>
</tr>
```

Transforming Data with Pipes

- Transform bound properties before display
- Built-In pipes include
  - Date
  - Number,decimal,percent,currency
  - Uppercase,lowercase

```
<td>{{toy.name | lowercase}}</td>
```

Before the toy.name is displayed, it is converted to lowercase

```
<td>{{toy.price | currency:'USD':true:'1.2-2'}}</td>
```

Currency has 3 parameters.
1. Currency code
2.Boolean value to display currency symbol
3. Digit info

Building Custom Pipes Cont.

```
import {Pipe,PipeTransform} from '@angular/core';

@Pipe({
    name:'recipeFilter'
})

export class RecipeFilterPipe implements PipeTransform{

    transform(value:any[],args:string[]):any[]{
        let filter:string = args[0] ? args[0].toLocaleLowerCase():null;
        return filter?value.filter(
            (recipe:any)=> recipe.name.toLocaleLowerCase().indexOf(filter)!=-1):value;
    }
}
```

Step 1: import Pipe and PipeTransform interfaces

Step 2: Declare the @Pipe decorator. This will be the pipe used in the html file.

Step 3: Implement the PipeTransform interface and override the transform method

```
import {RecipeFilterPipe} from './recipes/recipesfilter.pipe';

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

Step 4: Register the Custom pipe in the declarations array of the root AppModule

```
<tr *ngFor='let toy of toys | recipeFilter:listFilter' >
```

Step 5: Use the pipe in the template

Encapsulating Component Styles

```
@Component({
  selector:'recipe-list',
  templateUrl:'app/recipes/recipelist.component.html',
  styles:[' thead {color:green}'],
})
```

Inline styles using 'styles' property

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

External files using 'styleUrl' property

Sprint 3 Product List and Filter Search

- For sprint 3, you task are
  - Display a list of products in the Product List component. For this sprint we would just use a mockup list.
  - Filter Search should update the product list based on the search criteria
  - A button that shows and hides the product image when clicked.

Products

Filter by:

Filtered by:

Show

Product	Code	Available	Price
GI Joe	toy101	March 2017	$33.00
Pokemon	toy102	March 2017	$55.00
Basketball	toy103	March 2017	$33.00

Display a list of products with the following properties

Products

Filter by:

Filtered by:

Hide

Product	Code	Available	Price
GI Joe	toy101	March 2017	$33.00

Product image should show /hide when the button is clicked

Products

Filter by:
po

Filtered by: po

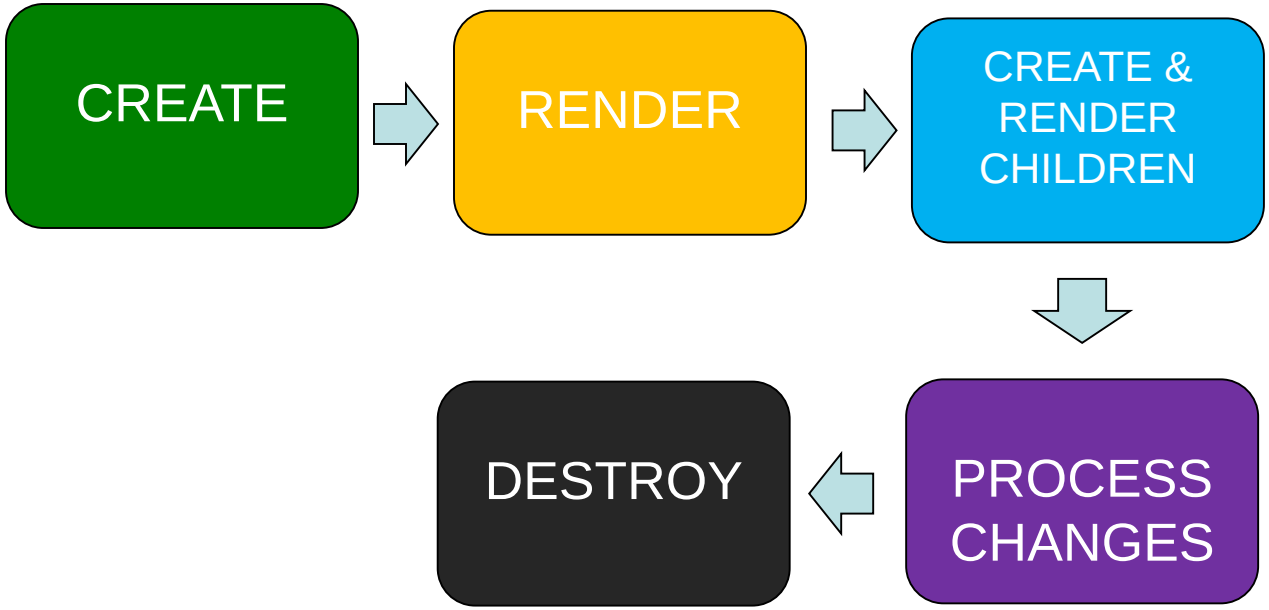
Show

Product	Code	Available	Price
Pokemon	toy102	March 2017	$55.00

List should update base on the search filter

Module 6:
Component Lifecycle

Component Lifecycle



Commonly Used Lifecycle Hooks

OnInit: Perform component initialization, retrieve data

OnChanges: Perform action after change to input properties

OnDestroy: Perform cleanup

Using a Lifecycle Hook

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

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

export class RecipeListComponent implements OnInit{

    ngOnInit():void{
        console.log("OnInit method called");
    }
```

Step 1: import the OnInit interface from angular/core

Step 2: Implement the OnInit interface

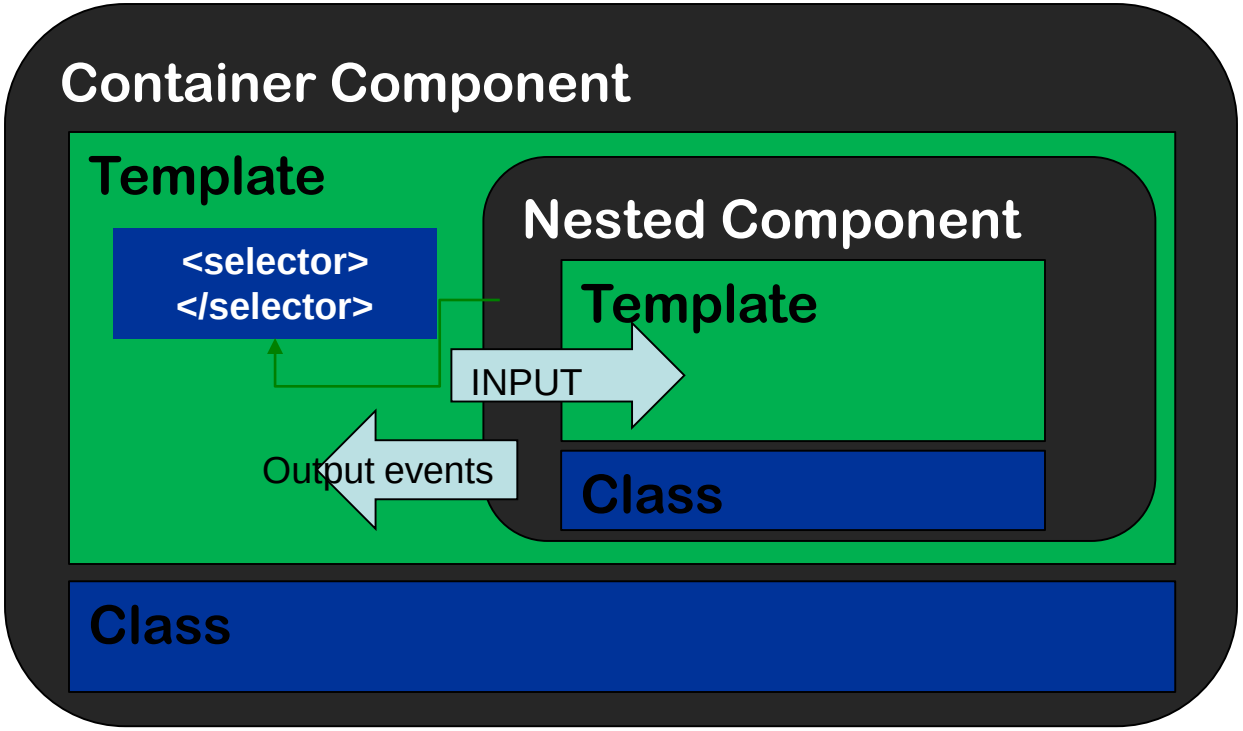
Step 3: Implement the ngOnInit() method

Sprint 4: Implement Lifecycle Hooks

- For sprint 4, your task is to implement the OnInit lifecycle method for both the Home and Product List Component. For now just place a simple console.log that displays “OnInit was called”.

Module 7: Nested Components

Building Nested Components



Nested component receive info from the container through INPUT properties and send messages to the container by raising OUTPUT events

Building Nested Components(Cont)

Container Component

```
export class RecipeListComponent implements OnInit{
```

Nested Component

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

export class StarComponent implements OnChanges {
```

Container Component Template

```
<td>
    <ratings-star></ratings-star>
</td>
```

```
import {StarComponent} from './shared/star.component';

@NgModule({
  imports:      [ BrowserModule,FormsModule ],
  declarations: [ AppComponent,RecipeListComponent,RecipeFilterPipe,StarComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Passing Data to a Nested Component using the @Input

Nested Component

```
export class StarComponent implements OnChanges {
    @Input() rating:number=4;
    starWidth:number;
```

Nested component receives data using the @Input() decorator

Container Component Template

```
<td>
    <ratings-star [rating]='toy.rating'></ratings-star>
</td>
```

Container passed info to the nested component through property bindings

Raising an event to the Container Component using the @Output

```
export class StarComponent implements OnChanges {
    @Input() rating:number;
    @Output() ratingClicked:EventEmitter<string> =
                        new EventEmitter<string>();

    onClick():void{
        this.ratingClicked.emit(`The rating ${this.rating} was clicked`);
    }
```

Step 1. Create a property and decorate it with the @Output() decorator

Step2. Create an method that will send the message to the container component

```
<div class="crop"
    [style.width.px]="starWidth"
    [title]="rating"
    (click)='onClick()'>
```

Step3. On the Container template, set the event binding property

```
<td>
   <ratings-star [rating]='toy.rating'
           (ratingClicked)='onRatingClicked($event)'></ratings-star>
</td>
```

Step4. Handle the message passed in the Container component

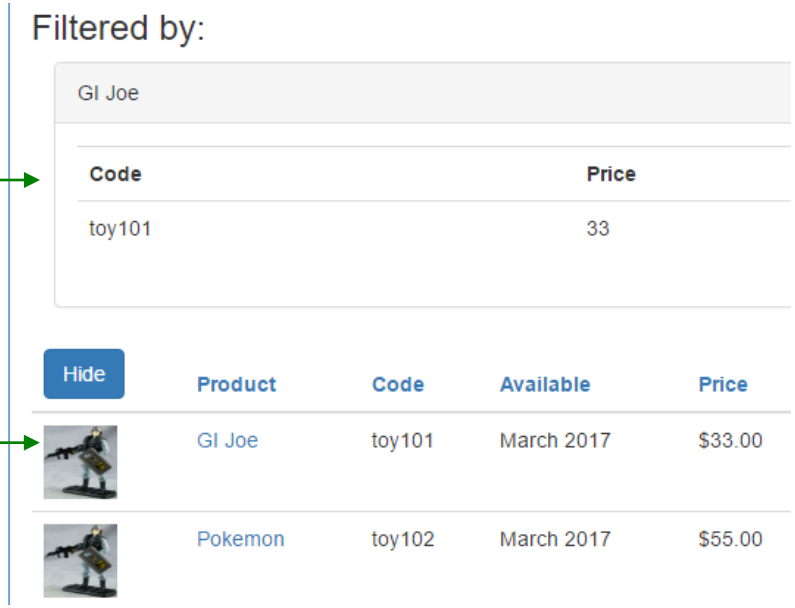
```
export class RecipeListComponent implements OnInit{

  onRatingClicked(message:string):void{
        this.pageTitle=' Message from nested component: ' + message;
    }
```

Sprint 5: Nested Display Details

- For sprint 5, you task is to implement the DetailsComponent. When a user clicks on an product image, it should display the product details as shown.

When an image is clicked, product details should be displayed on the top container



Module 8:
Services and Dependency Injection

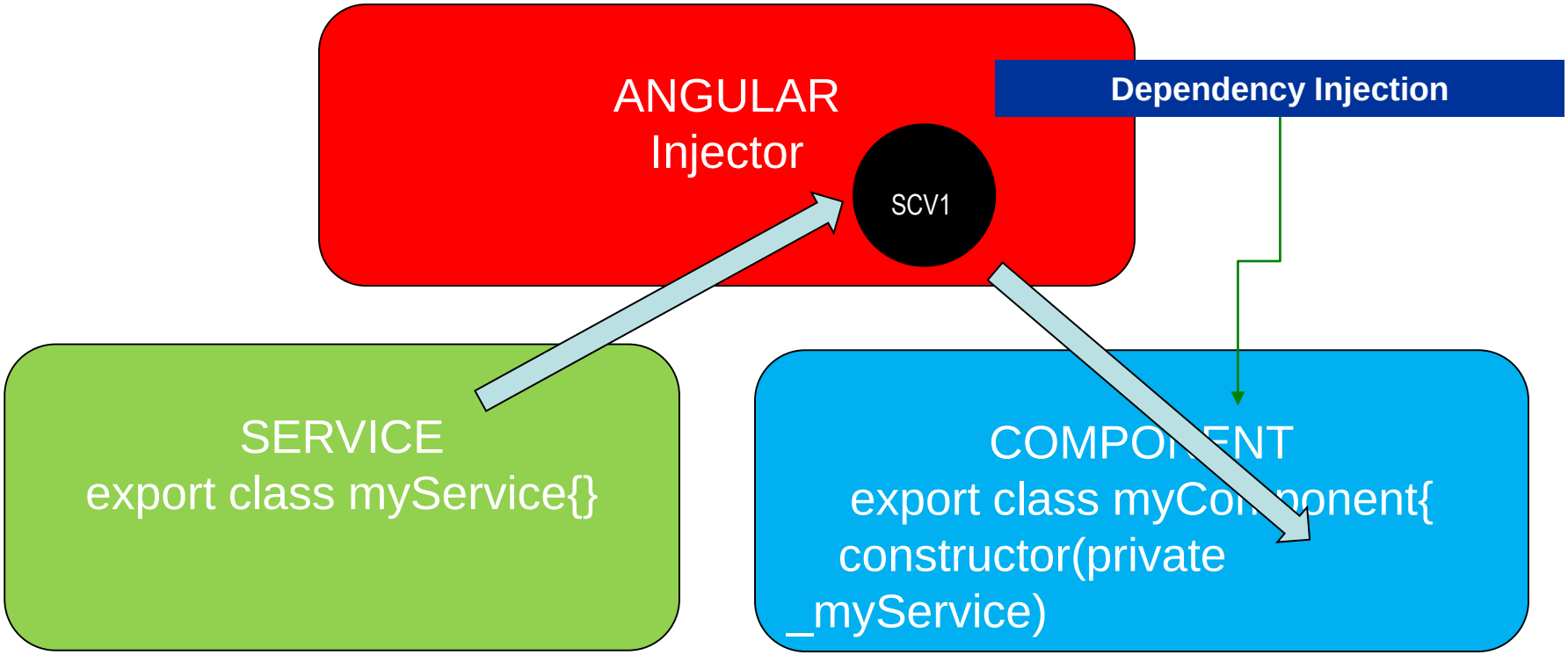
What are Services?

- A Service is a class with a focused purposed
- Independent from any particular component
- Provide shared data or logic across components
- Encapsulate external interactions

What is Dependency Injection

- It is a coding pattern in which a class receives the instances of objects it needs (dependencies) from an external source rather than creating them itself.

How does is work?



Building a Service

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

@Injectable()
export class ProductService{
    getProducts():IToy[]{
      return;
    }
}
```

Step1. Import the Injectable Decorator

Step2. Decorate the class with @Injectable()

Step3. Register the Service in the root Component by supplying a "providers" Metadata

```
import {ProductService} from './recipes/products.services';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>
      <recipe-list></recipe-list>`,
  providers:[ProductService]
})
export class AppComponent  {
```

Injecting the Service

```
import {ProductService} from './products.services';

@Component({
  selector:'recipe-list',
  templateUrl:'app/recipes/recipelist.component.html',
  styleUrls:['app/recipes/recipelist.component.css'],

})

export class RecipeListComponent implements OnInit{
    private _productService:ProductService;
    constructor(productService:ProductService){
        this._productService=productService;
    }
```

Step1. Import the Service

Step2. In the constructor, provide a placeholder for the Service type. The Angular Injector will use this to inject the Dependency.

```
constructor(private _productService:ProductService){}
```

Shortcut version of defining a dependency

Sprint 6: Product Data Service

- For sprint 6, your task is to convert the mock product list into a service. This service should be a dependency of the ProductList component.

Module 9:
Retrieving Data Using Http

Retrieving Data Using Http

- Angular2 has built in support for retrieving data via web apis.
- Support for Observables and Reactive Extensions for retrieving data asynchronously

Observables and Reactive Extensions

- Observables is an array whose items arrive asynchronously over time.
- Helps as manage asynchronous data.
- Angular2 uses Reactive Extension (RxJS), a third party library for implementing Observables.

Observables

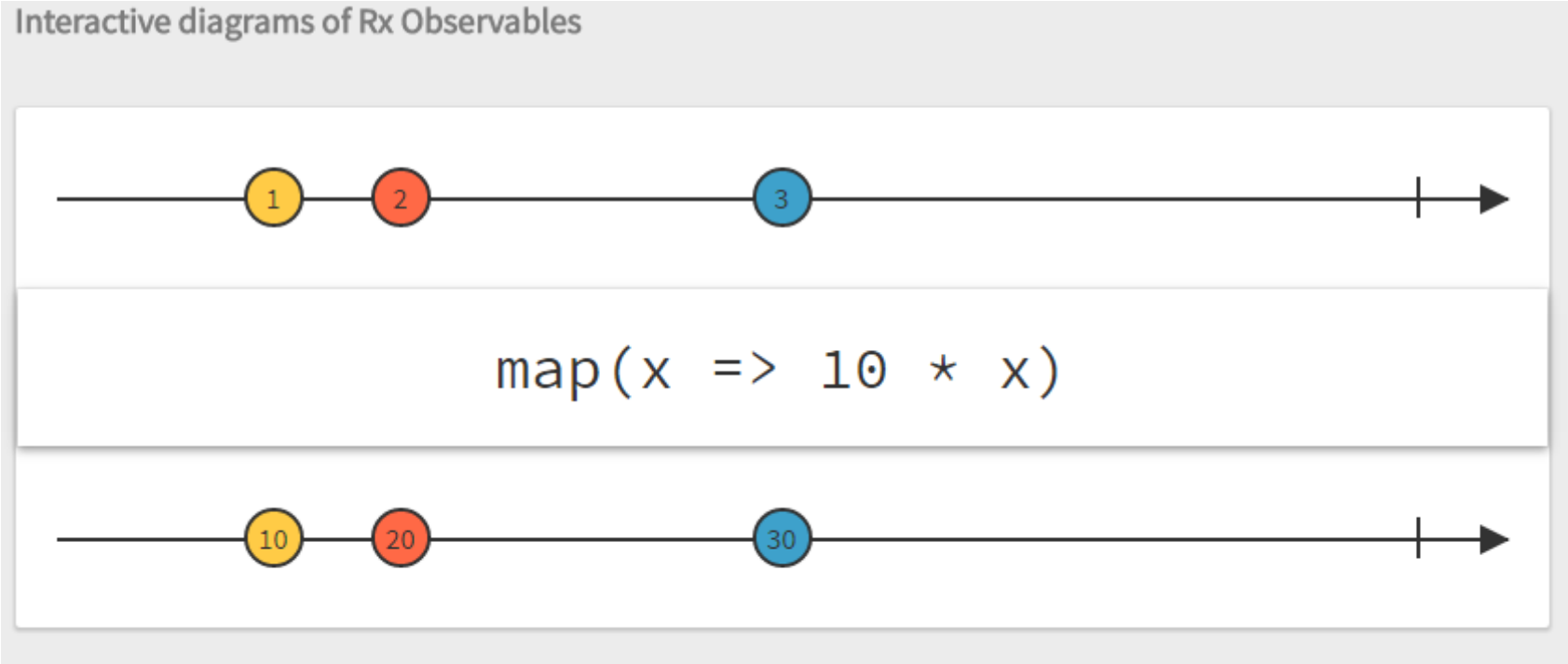
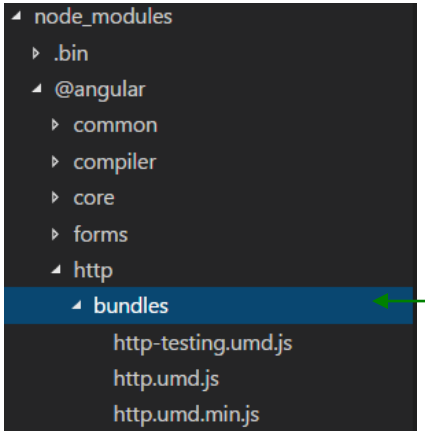


Diagram is from http://www.rxmarbles.com

Promise vs Observables

- Promise
  - Returns a single value
  - Not cancellable
- Observables
  - Works with multiple values
  - Cancellable
  - Supports array operators such as map,filter, reduce and many more

Setting Up HTTP



Step 1. Include the Angular2 Http scripts

```
import {HttpModule} from '@angular/http';
import 'rxjs/Rx';

@NgModule({
  imports:      [ BrowserModule,FormsModule,HttpModule ],
  declarations: [ AppComponent,RecipeListComponent,Recipe
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Step 2. Register the HttpModules and import the RxJS library

Subscribing to an Observable

```
export class RecipeListComponent implements OnInit{
    constructor(private _productService:ProductService){}

   ngOnInit():void{
        console.log("OnInit method called");
        this._productService.getProducts()
        .subscribe(
            toys=> this.toys = toys,
            error=> this.errorMessage = <any>error
        );

    }
```

Action to take when the observable emits data

Action to take when something goes wrong

Request to receive notifications from the Observable

Sprint 7: Using Http

- For Sprint 7, your task is to refactor you Product Data Service to retrieve products via http. To simulate a web api call, place your list in a products.json file.

Module 10:
Routing and Navigation

Routing

```
<body>
    <my-app>
    Loading AppComponent content here ...
    </my-app>
</body>
```

Angular apps are Single Page Apps
or SPA as they are commonly referred to.

All views are displayed in one page.

Views take turn to display on one page

How does Angular manage which view
To display when?

Routing is the one responsible for this
With routing you can

- Configure a route for each component
- Define options/actions
- Tie a route to each option/action
- Activate the route based on user action

Angular2 Routing Setup

- @angular
  - common
  - compiler
  - core
  - forms
  - http
  - platform-browser
  - platform-browser-dynamic
  - router
    - bundles
      - router-testing.umd.js
      - router-upgrade.umd.js
      - router-upgrade.umd.min.js
      - router.umd.js
      - router.umd.min.js

Step 1. Include the Angular2 Routing scripts

Step 2. Register the RouterModule in the main AppModule

```
import {RouterModule} from '@angular/router';

@NgModule({
  imports:      [ BrowserModule,FormsModule,HttpModule,RouterModule ],
  declarations: [ AppComponent,RecipeListComponent,RecipeFilterPipe,StarComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Configure Routes

```
import {RouterModule,Routes} from '@angular/router';
//configure routes
const routes:Routes=[
                    {path:'',redirectTo:'/welcome', pathMatch: 'full'},
                    {path:'welcome',component:WelcomeComponent},
                    {path:"list", component:RecipeListComponent}
                  ];

@NgModule({
  imports:      [ BrowserModule,FormsModule,HttpModule, RouterModule.forRoot(routes)],
  declarations: [ AppComponent,RecipeListComponent,RecipeFilterPipe,
                  StarComponent,ProductDetailComponent,WelcomeComponent],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Setup routes in an array of type Routes

Remember to include all the Components being used in the declarations

Set the routes to the RouterModule

Setting Up The Main View

```
import {RouterModule} from '@angular/router';

@Component({
  selector: 'my-app',
  template:
  `<div>
      <nav class="navbar navbar-default">
         <div class="container-fluid">
            <ul class="nav navbar-nav">
               <li><a routerLink="/welcome">Home</a></li>
               <li><a routerLink="/list">Products</a></li>
          </ul>
        </div>
      </nav>
  </div>
  <div class="container">
      <router-outlet></router-outlet>
  </div>`,
  providers:[ProductService]
})


export class AppComponent  {
   name = 'RecipesRUs'; }
```

Setup links to routes using the routerLink directive

The <router-outlet> selector is where the app templates will be displayed

Passing Parameters

```
//configure routes
const routes:Routes=[
                    {path:'',redirectTo:'/welcome', pathMatch: 'full'},
                    {path:'welcome',component:WelcomeComponent},
                    {path:"list", component:RecipeListComponent},
                    {path:"details/:code",component:ProductDetailComponent}
                  ];
```

Configure a route with the path format "path/:parameter"

```
<td><a [routerLink]="['/details',toy.code]">{{toy.name}}</a></td>
```

Using the routerLink directive, you can specify the path and the parameter to be passed

Passing Parameters Cont

```
import { ActivatedRoute, Params }   from '@angular/router';

@Component({
    templateUrl:'app/recipes/product-details.component.html'
})

export class ProductDetailComponent implements OnInit{
    constructor(private productService:ProductService,
            private route:ActivatedRoute,
            private location:Location){}
```

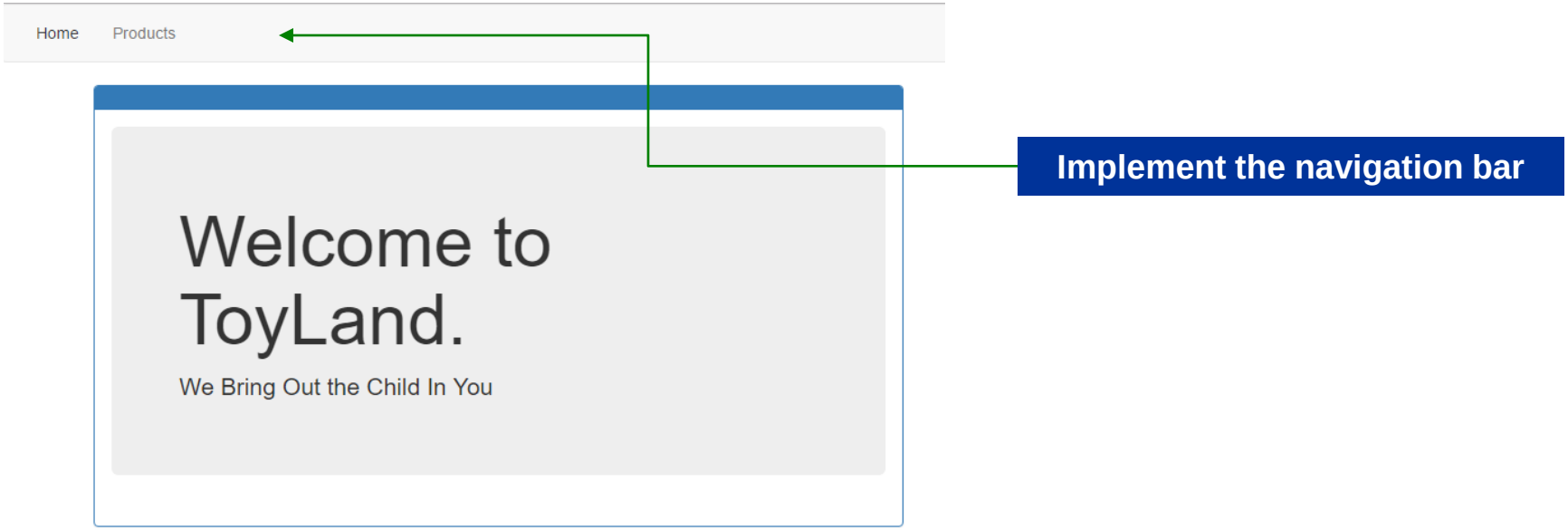
On the page that would receive the parameter, specify the ActivateRoute as a Dependency

```
ngOnInit():void{
    console.log(this.route.params['code']);
    this.productService.getProduct(this.route.params['code'])
    .subscribe(
        toy=>this.toy=toy,
        error=>this.errorMessage=error
        );
}
```

Using the ActivateRoute that was injected by Angular, you can extract the parameter that was passed using format route.params[param]

Sprint 8 Implement Navigation

- For sprint 8, your task is to implement the navigation bar of the web app. When the web first loads, it should default to the Welcome page



Common Setup Files

tsconfig.json × product-details.component.html products.s

```
1   {
2     "compilerOptions": {
3       "target": "es5",
4       "module": "commonjs",
5       "moduleResolution": "node",
6       "sourceMap": true,
7       "emitDecoratorMetadata": true,
8       "experimentalDecorators": true,
9       "lib": [ "es2015", "dom" ],
10      "noImplicitAny": true,
11      "suppressImplicitAnyIndexErrors": true
12    },
13    "exclude": [
14      "node_modules"
15    ]
16  }
17
```

tsconfig.json

This is the typescript configuration file.
The typescript compiler reads this file and uses this settings.

The presence of this file in a folder means that it's the root of the typescript project

Common Setup Files (package.json)

```
{
  "name": "angular-quickstart",
  "version": "1.0.0",
  "description": "QuickStart package.json from the docume
  "scripts": {
    "build": "tsc -p src/",
    "build:watch": "tsc -p src/ -w",
    "build:e2e": "tsc -p e2e/",
    "serve": "lite-server -c=bs-config.json",
    "serve:e2e": "lite-server -c=bs-config.e2e.json",
    "prestart": "npm run build",
    "start": "concurrently \"npm run build:watch\" \"npm
    "pree2e": "npm run build:e2e",
    "e2e": "concurrently \"npm run serve:e2e\" \"npm run
    "preprotractor": "webdriver-manager update",
    "protractor": "protractor protractor.config.js",
    "pretest": "npm run build",
    "test": "concurrently \"npm run build:watch\" \"karma
    "pretest:once": "npm run build",
    "test:once": "karma start karma.conf.js --single-run"
    "lint": "tslint ./src/**/*.ts -t verbose"
  },
```

The scripts section contains the scripts that we can run with the node package manager

```
"dependencies": {
  "@angular/common": "~2.4.0",
  "@angular/compiler": "~2.4.0",
  "@angular/core": "~2.4.0",
  "@angular/forms": "~2.4.0",
  "@angular/http": "~2.4.0",
  "@angular/platform-browser": "~2.4.0",
  "@angular/platform-browser-dynamic": "~2.4.0",
  "@angular/router": "~3.4.0",
  "angular-in-memory-web-api": "~0.2.4",
  "bootstrap": "^3.3.7",
  "core-js": "^2.4.1",
  "rxjs": "5.0.1",
  "systemjs": "0.19.40",
  "zone.js": "^0.7.4"
},
```

The dependencies section contain the libraries that we need to run the application

```
"devDependencies": {
  "concurrently": "^3.2.0",
  "lite-server": "^2.2.2",
  "typescript": "~2.0.10",
  "canonical-path": "0.0.2",
  "tslint": "^3.15.1",
  "lodash": "^4.16.4",
  "jasmine-core": "~2.4.1",
  "karma": "^1.3.0",
  "karma-chrome-launcher": "^2.0.
  "karma-cli": "^1.0.1",
  "karma-jasmine": "^1.0.2",
  "karma-jasmine-html-reporter":
  "protractor": "~4.0.14",
  "rimraf": "^2.5.4",
  "@types/node": "^6.0.46",
  "@types/jasmine": "2.5.36"
},
```

The devdependices contain the additional libraries we need such as support, testing etc.

Thank you!