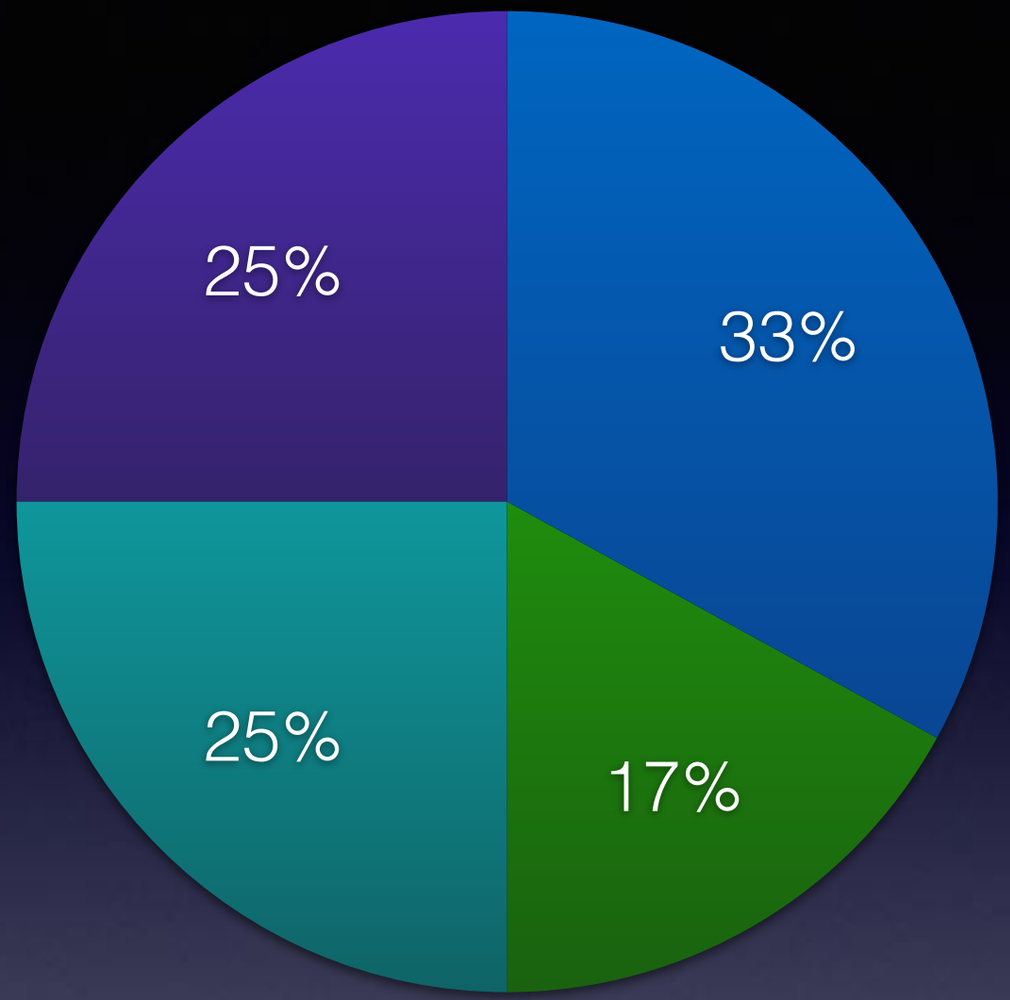
Angular 2

by Artem Koziar

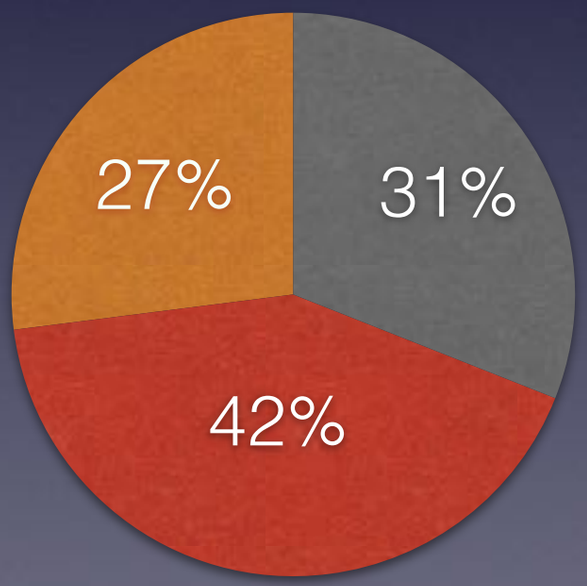
- 1/3 (33%) Planning;
- 1/6 (17%) Writing programs;
- 1/4 (25%) Component testing;
- 1/4 (25%) System testing.



Angular 2

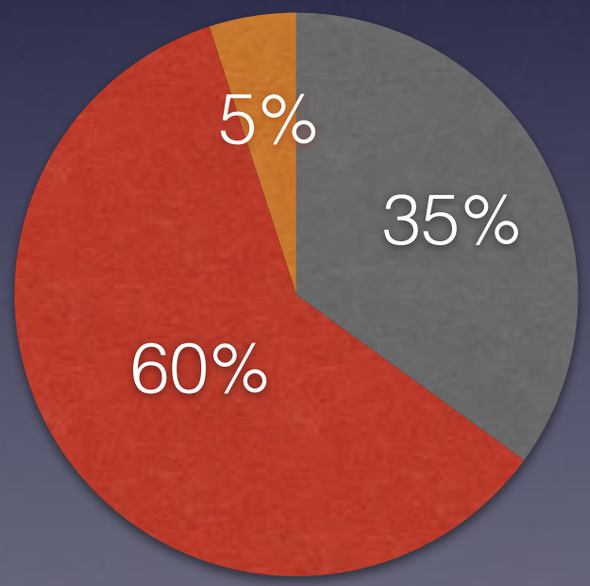
- React
- Angular 2
- Angular 1

2016



- React
- Angular 2
- Angular 1

2017:)



Angular 2

- 15 Sep 2016 — Final Release
- TypeScript (Microsoft)
- https://angular.io/
- https://github.com/angular/angular/
- https://angular.io/styleguide
- https://github.com/angular/angular-cli

Angular 2

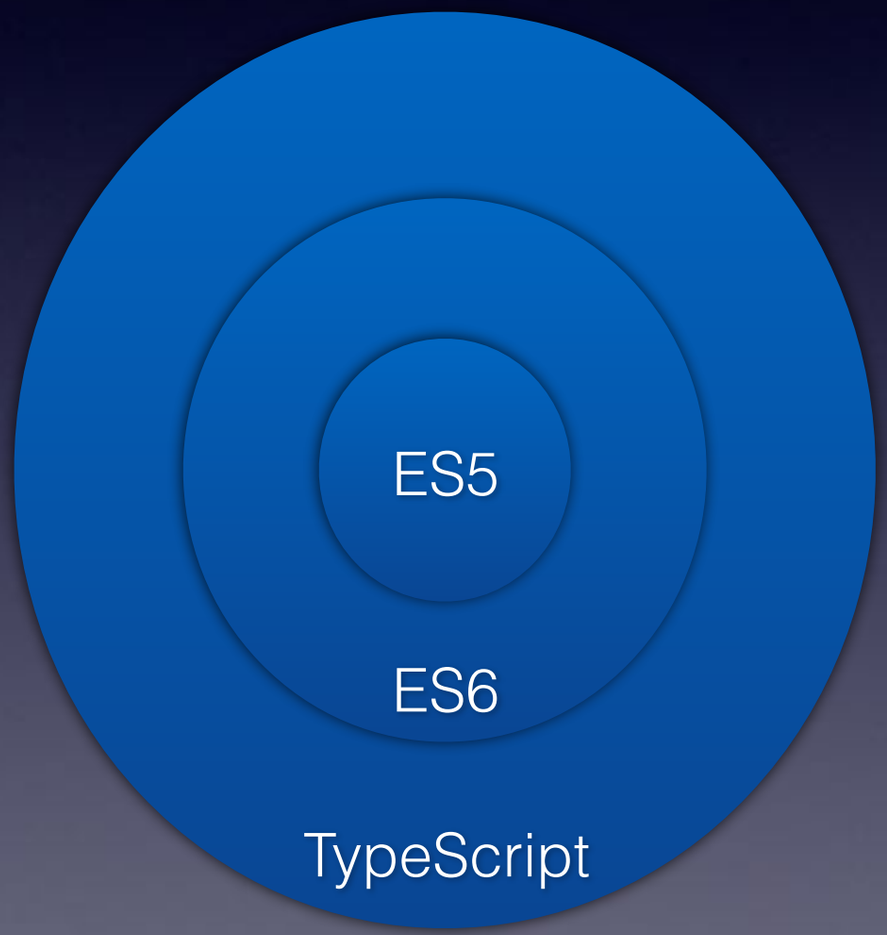
1. TypeScript and Decorators
2. Components
3. Observables
4. Dependency injection
5. Routing
6. Change detection strategies
7. (Forms)

TypeScript

ES6:
— classes
— modules

ES7:
— decorators

TypeScript:
— types
— annotations



TypeScript

- JavaScript's types also exist in TypeSctipt
- TypeScript also adds **enum, any** & **void** (like **undefined**)
- **Interface** allows for custom, abstract types
- Function Signatures can be typed Using **Interfaces**
- **Class**es also define types
- *If it walks like a duck, it is a duck, types with the same shapes are compatible*

TypeScript: Simple types

- let a = 123; // Number
- let b: number = 123; // Number

the same other types (boolean, string and objects etc)

- let a1: string[] = []; // Array of Strings (define empty array)
- let a2: string[]; // Array of Strings (undefined)
- let a3: Array<string> // Array of Strings (undefined)
- let a4 = [“a”, “b”]; // Array of Strings

TypeScript: Interface

```ts
interface User {
  name: string;
}

class UserModel {
  constructor(public name: string, private age?: number) {}
}

let u: User = { name: 'foo' };

u = new UserModel(‘bar');


function useUser(user: UserModel) {
  console.log(user.name);
}
```

TypeScript: Parameters

optional parameter

```
interface User {
  name: string;
}

class UserModel {
  constructor(public name: string,
    private age?: number,
    private city? = 'Kyiv') {}
  }

  let u: User = { name: 'foo' };

  u = new UserModel(‘bar');


function useUser(user: UserModel) {
  console.log(user.name);
}
```

optional parameter
with predefined value

TypeScript: Functions

```ts
interface CallbackForUser {
  (userName: string, age: number): number;
}

class UserModel {
  constructor(public name: string, private age?: number, private city? = 'Kyiv') {}

  doSome(cb: CallbackForUser) {
    cb(this.name, this.age);
  }
}

let u: UserModel = new UserModel('bar');

// 1
u.doSome((name: string, age: number) => {
  console.log(`User ${name} is ${age} years old`);
  return age * 100;
});

// 2

let cb: CallbackForUser = (name: string, age: number) => {
  console.log(`User ${name} is ${age} years old`);
  return age * 100;
};
u.doSome(cb);
```

Decorators

- Decorators — functions that operate on a “target”
- “Target” are classes, methods, properties and parameters
- Decorators invoked with leading **@** like **@Component()**
- Angular 2 decorators always use training brackets, like **@Inject()**
- Decorators do not get follower by **;**

Component

```
// > ng g component hello-world

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

@Component({
  selector: 'app-hello-world',
  template: ‘<p>Hello, {{ title }}</p>’
})
export class HelloWorldComponent {

  title: string;

  constructor() {
    this.title = 'World';
  }

}

// <app-hello-world></app-hello-world>
```

Component

```
<app-hello-world>

  <app-header></app-header>

  <app-user-list>
    <app-user-item></app-user-item>
    <app-user-item></app-user-item>
    <app-user-item></app-user-item>
  </app-user-list>

  <app-add-user-form></app-add-user-form>

  <app-footer></app-footer>

</app-hello-world>
```

Component: Input, OnInit

```
// <app-hello-world title="Some Text"></app-hello-world>
// <app-hello-world [title]="variableName"></app-hello-world>

import { Component, Input, OnInit } from '@angular/core';

@Component({
  selector: 'app-hello-world',
  template: `<p>Hello, {{ title }}!</p>`
})
export class HelloWorldComponent implements OnInit {

  @Input()
  title: string;

  constructor() {
    // if (!this.title) {
    //   this.title = 'World';
    // }
    console.log('constructor', this.title); // undefined
  }

  ngOnInit() {
    if (!this.title) {
      this.title = 'World';
    }
    console.log('ngOnInit', this.title);
  }

}
```

Component: Output

```
// > ng g component counter
// <app-counter
start="12" (result)="onCounterChange($even
t)"></app-counter>

import { Component, Input, Output, OnInit,
EventEmitter } from '@angular/core';

@Component({
  selector: 'app-counter',
  template: `<div>
    <p>{{ counter }}</p>
    <p><a (click)="inc()">plus</a> | <a
(click)="dec()">minus</a></p>
</div>`,
  styleUrls: ['./counter.component.css']
})
export class CounterComponent implements
OnInit {

  @Input()
  start: number;

  counter: number;

  @Output()
  result: EventEmitter<number> = new
EventEmitter();

  constructor() { }

  ngOnInit() {
    this.counter = +this.start || 0;
  }

  inc() {
    this.counter++;
    this.result.emit(this.counter);
  }

  dec() {
    this.counter--;
    this.result.emit(this.counter);
  }

}
```

Two-Way Data Binding

Combines the input and output binding into single notation using the **ngModel** directive.

```
<input [(ngModel)]=“user.name">
```

[()] = BANANA IN A BOX

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

Template directives

```
<ul>
  <li>{{title}}</li>
  <li *ngFor="let menu of menus" [ngClass]="{'active': menu === activeMenu}"><a
routerLink="{{menu.link}}" (click)="onClick(menu)">{{menu.title}}</a></li>
</ul>

<div *ngIf="currentHero">Hello, {{currentHero.firstName}}</div>

<div [class.hidden]="isSpecial">Hide with class</div>
<div [style.display]="isSpecial ? 'block' : 'none'">Show with style</div>
<div [ngClass]="{'first': true, 'second': true, 'third': false}">...</div>

<select [(ngModel)]="employee.manager" (ngModelChange)="change($event)">
  <option *ngFor="let manager of managers" [ngValue]="manager">{{ manager.name }}</option>
</select>

<span [ngSwitch]="toeChoice">
  <span *ngSwitchCase="'Eenie'">Eenie</span>
  <span *ngSwitchCase="'Meanie'">Meanie</span>
  <span *ngSwitchCase="'Miney'">Miney</span>
  <span *ngSwitchCase="'Moe'">Moe</span>
  <span *ngSwitchDefault>other</span>
</span>
```

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

Service

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

export class Todo {
  constructor(public id: number, public title?: string) {}
}

@Injectable()
export class TodoService {

  constructor() { }

  getList(): Promise<Todo[]> {
    return new Promise((resolve, reject) => {
      window.setTimeout(() => {
          resolve([
            new Todo(1, 'Todo 1'),
            new Todo(2, 'Some Todo 2'),
            new Todo(3, 'Op Todo3'),
            new Todo(4, 'Todo 4'),
            new Todo(5, 'Todo 5')
          ]);
      }, Math.random() * 2000 + 1000);
    });
  }

  getTodo(id: number): Promise<Todo> {
    return this.getList()
      .then(todos => todos.find(todo => todo.id === id));
  }

}
```

```ts
// Component
import { Component, OnInit } from '@angular/core';
import { TodoService, Todo } from '../shared/todo.service';

@Component({
  selector: 'app-todo-list',
  templateUrl: './todo-list.component.html',
  styleUrls: ['./todo-list.component.css']
})
export class TodoListComponent implements OnInit {
  public todos: Todo[] = [];
  private isLoading = true;
  private activeTodo: Todo;

  constructor(private todoService: TodoService) { }

  ngOnInit() {
    this.todoService.getList()
      .then((todos) => {
        this.todos = todos;
        this.isLoading = false;
      });
  }

  onClick(todo: Todo) {
    this.activeTodo = todo;
  }

}
```

Observables (ES7)

- Observables open up a continuous channel of communication in which multiple values of data can be emitted over time.
- From this we get a pattern of dealing with data by using array-like operations to parse, modify and maintain data.
- Angular 2 uses observables extensively - you'll see them in the HTTP service and the event system.

http://rxmarbles.com/#filter

Observables: Example

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

import { Http } from '@angular/http';
import './rxjs-operators';
import { Observable }     from 'rxjs/Observable';

@Component({
  selector: 'app-test'
})
export class TestSomponent {
  constructor(private http: Http) {}

  getHeroes (): Observable<Hero[]> {
    return this.http.get(this.heroesUrl)
      .map(res => res.json())
      .filter(data => data.age > 18)
      .subscribe((data) => {
        this.data = data;
      });
  }
}
```

Router

```
import { ModuleWithProviders }  from '@angular/core';
import { Routes, RouterModule } from '@angular/
router';

import { HomeComponent } from './home/
home.component';
import { TodoListComponent } from './todo-list/todo-
list.component';
import { TodoDetailsComponent } from './todo-details/
todo-details.component';

const appRoutes: Routes = [
  {
    path: '',
    component: HomeComponent
  },
  {
    path: 'todo',
    component: TodoListComponent
  },
  {
    path: 'todo/:id',
    component: TodoDetailsComponent
  }
];

export const routing: ModuleWithProviders =
RouterModule.forRoot(appRoutes);
```

```
<a [routerLink]="['/todo/', todo.id]”>{{ todo.title }}</a>
```

```
export interface Route {
    path?: string;
    pathMatch?: string;
    component?: Type<any>;
    redirectTo?: string;
    outlet?: string;
    canActivate?: any[];
    canActivateChild?: any[];
    canDeactivate?: any[];
    canLoad?: any[];
    data?: Data;
    resolve?: ResolveData;
    children?: Route[];
    loadChildren?: LoadChildren;
}
```

Structure

```
/src
    app/
        shared/
            todo.service.ts
            todo.service.spec.ts
            todo.ts
        todo-list/
            todo-list.component.ts
            todo-list.component.html
            todo-list.component.css
            todo-list.component.spec.ts
        todo-details/
        app.component.ts
        app.component.spec.ts
assets/
    imgs/
    fonts/
index.html
main.ts
```

https://angular.io/styleguide

Bootstrap

```
// main.ts
import { platformBrowserDynamic } from '@angular/platform-
browser-dynamic';
import { enableProdMode } from '@angular/core';
import { environment } from './environments/environment';
import { AppModule } from './app/';

if (environment.production) {
  enableProdMode();
}

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

```
// app.module.ts

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule, JsonpModule } from '@angular/http';

import { routing } from './app.routing';

import { AppComponent } from './app.component';
import { TodoListComponent } from './todo-list/todo-
list.component';

import { TodoService } from './shared/todo.service';
import { HomeComponent } from './home/home.component';
import { TodoDetailsComponent } from './todo-details/todo-
details.component';
import { HelloWorldComponent } from './hello-world/hello-
world.component';
```

```
@NgModule({
  declarations: [
    AppComponent,
    TodoListComponent,
    TopMenuComponent,
    HomeComponent,
    TodoDetailsComponent,
    WikipediaSearchComponent,
    HelloWorldComponent,
    // CounterComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    JsonpModule,

    routing

  ],

  providers: [
    TodoService
  ],

  bootstrap: [AppComponent]

})
export class AppModule { }
```

Test Component

https://angular.io/docs/ts/latest/guide/testing.html

:)

Home Work

Pokemon GO:

1. List with filtering and paging

2. Details page

Tech:

Create app component, list component and details component.

Create model, service and routers.

Tests?

Artem Koziar

a@temich.in.ua

artko@ciklum.com

Skype: lsharkichl

TNX