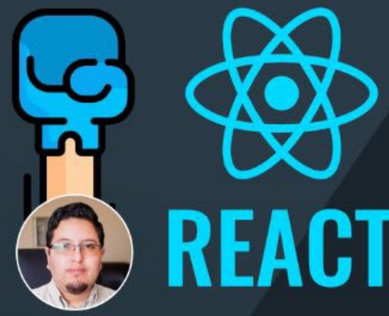
Angular

ANGULAR

Prof. Luis Aviles

VUE

Prof. Josh Ospina



REACT

Prof. Beto Quiroga

Google

- **Framework**
  Implementa todo el modelo MVC.
- **TypeScript**
  Angular usa TypeScript para escribir JavaScript más poderoso.
- **Templates**
  Angular usa HTML y directivas para insertar la lógica.
- **Two way data binding**
  Si cambian los datos en la interfaz cambian en el modelo (estado).

Soportado por la comunidad

- **Framework Progresivo**
  Agrega componentes a medida que tu app necesite.
- **JavaScript**
  Vue usa JavaScript y tiene una curva de aprendizaje muy rápida.
- **Templates y JSX**
  Vue soporta lo mejor de los 2 mundos: Templates en HTML o JSX.
- **Two way data binding**
  La directiva v-model actualiza el modelo si la vista cambia y vicersa.

facebook

- **Librería**
  Es solo la V (vista) del modelo MVC.
- **JavaScript**
  React usa solo JavaScript y aprovecha sus características.
- **JSX**
  React junta la lógica y el marcado en el mismo componente usando JSX.
- **One way data binding**
  Si cambian los datos en el estado cambian en la interfaz pero no al revés.

Angular

- Typescript
- Компонентный подход
- Управляемый Change detection
- Поддерживается любой платформой
- DI
- Структура проекта

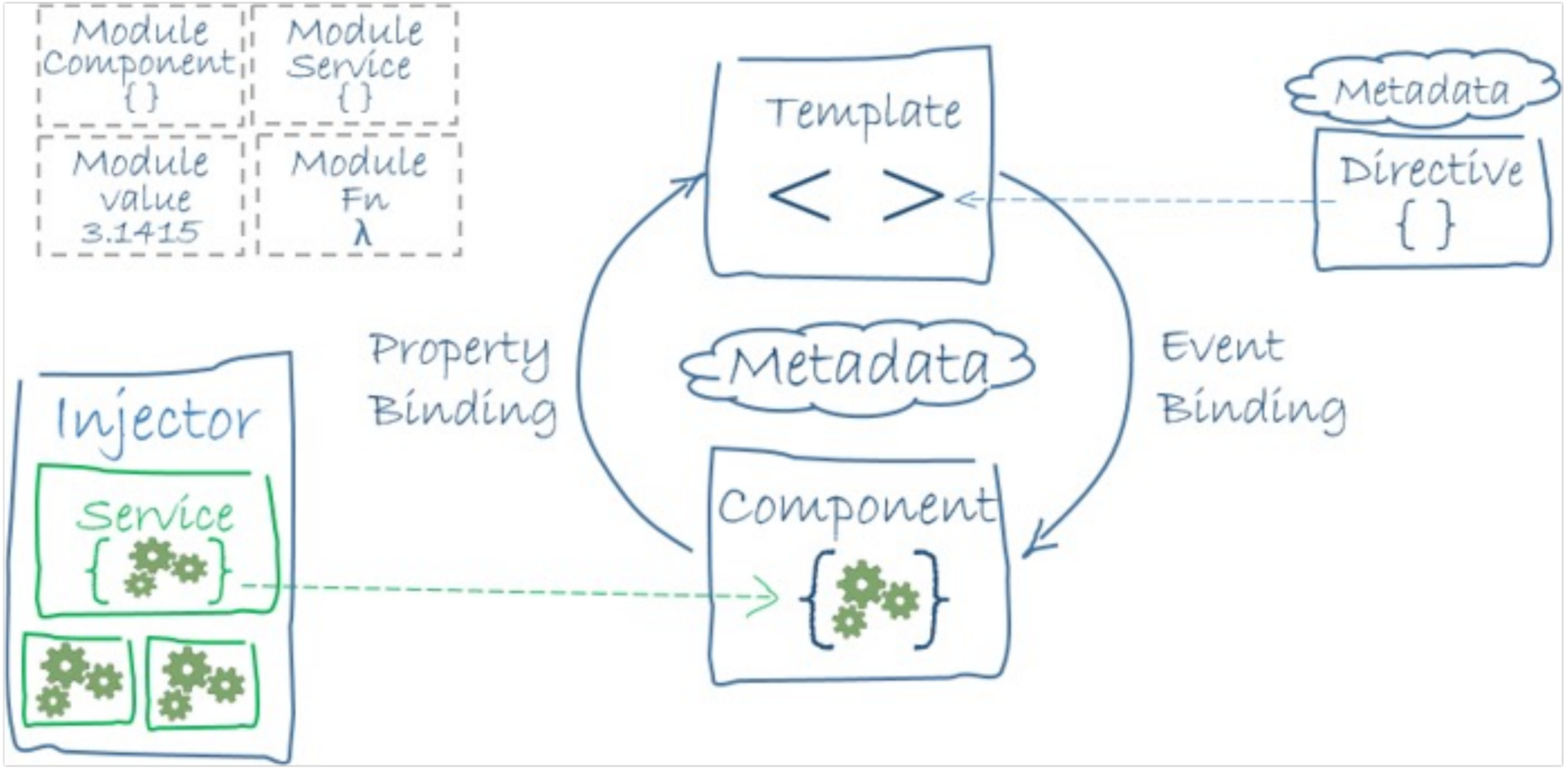
Конфигурация проекта

- Удобный и гибкий CLI
- angular.json — скелет приложения
- Очень гибкая конфигурация проекта
- tsconfig.json
- tslint.json
- Karma, Jasmine для тестов
- Кастомизация сборки

Как создать проект

1. Install node js (https://nodejs.org/en/download/)
2. npm install -g @angular/cli
3. ng new project-name
4. cd my-app
5. ng serve --open

Архитектура Angular приложения



Модули

- **Модуль** - это класс с декоратором `@NgModule()`, который служит изолирующей логической объединяющей структурой для компонентов, директив, пайпов и сервисов.

- Каждое приложение обязательно включает в себя корневой модуль (root module) под названием AppModule (файл app.module.ts).

- Могут ссылаться друг на друга (т.е. возможны импорт и экспорт модулей).

Структура модуля

Содержит секции:

- declarations -- компоненты, директивы (directives) и фильтры (pipes), содержащиеся в этом модуле.
- exports -- то, что объявлено в этой секции, будет видно и доступно для использования в других модулях.
- imports -- список внешних модулей, содержимое секции exports которых используется в текущем модуле.
- providers -- сервисы, реализованные в этом модуле, видимые в глобальном контексте приложения.
- bootstrap -- главное представление приложения (объявляется только в корневом модуле).

Пример модуля

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

Компонент

Компонент - обособленная часть функционала со своей логикой, HTML-шаблоном и CSS-стилями. Компонент управляет отображение представление на экране.
За объявление компонента отвечает декоратор `@Component()`

Основные свойства объекта, который принимает декоратор:

- `selector` — название компонента;
- `template` (или `templateUrl`) — HTML-разметка в виде строки (или путь к HTML-файлу);
- `providers` — список сервисов, поставляющих данные для компонента;
- `styles` — массив путей к CSS-файлам, содержащим стили для создаваемого компонента.

Декораторы свойств

@Input() - декоратор, используемый для получения данных

@Output() - декоратор, используемый для отправки данных, выставляя их в качестве производителей событий

```
@Component({
    selector: 'contacts-item',
    template: ` <p>{{ name }}</p> `,
})
export class ContactsItemComponent {
    _name: string = null;

    @Input() set name(value: string) {
        this._name = value + '*';
    }

    get name(): string {
        return this._name || 'Unknown';
    }
}
```

```
<contacts-item
    [name]="contactPerson"
    (saveContactPerson)="contactPerson = $event"
></contacts-item>
```

Жизненный цикл компонента

OnChanges - устанавливаются или изменяются значения входных свойств класса компонента;

OnInit - устанавливаются "обычные" свойства; вызывается единожды вслед за первым вызовом OnChanges();

DoCheck - происходит изменения свойства или вызывается какое-либо событие;

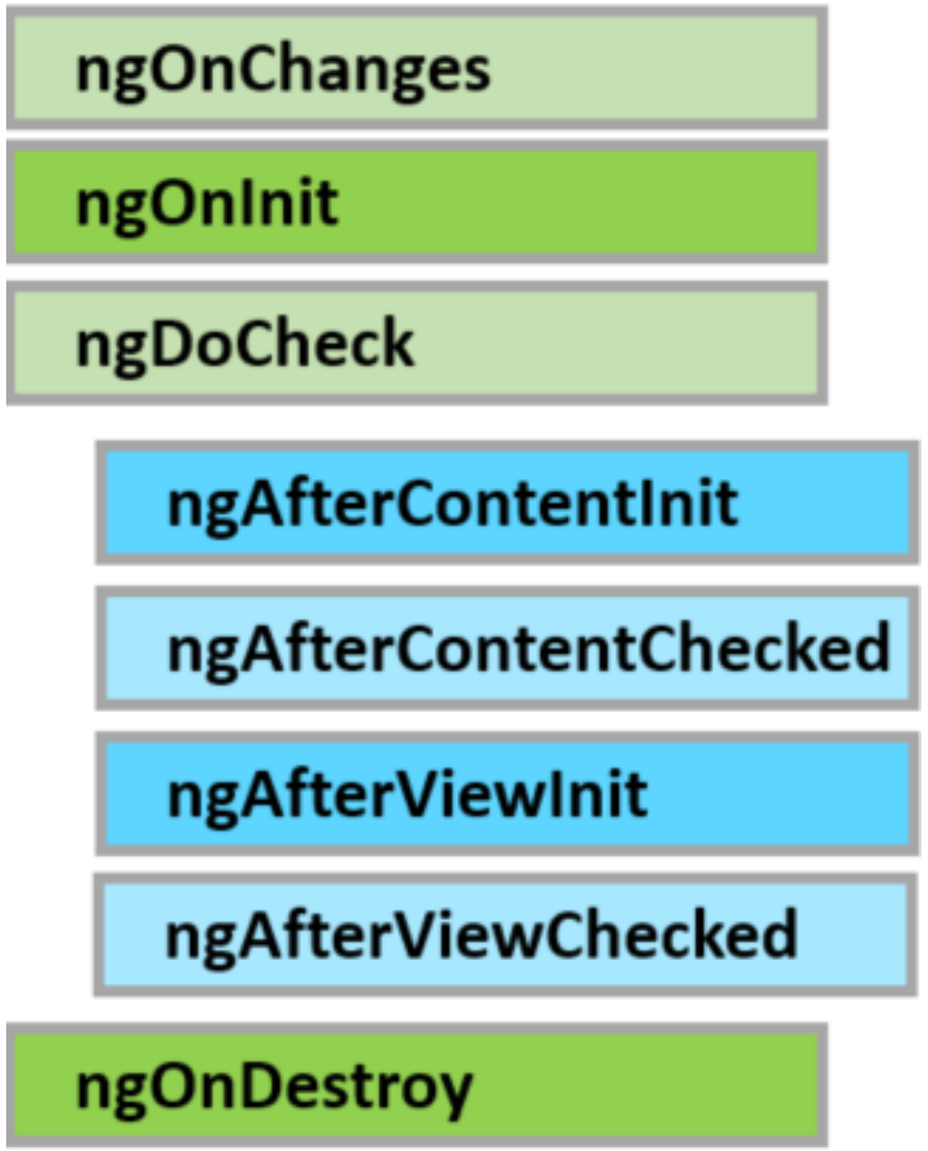
AfterContentInit - в шаблон включается контент, заключенный между тегами компонента;

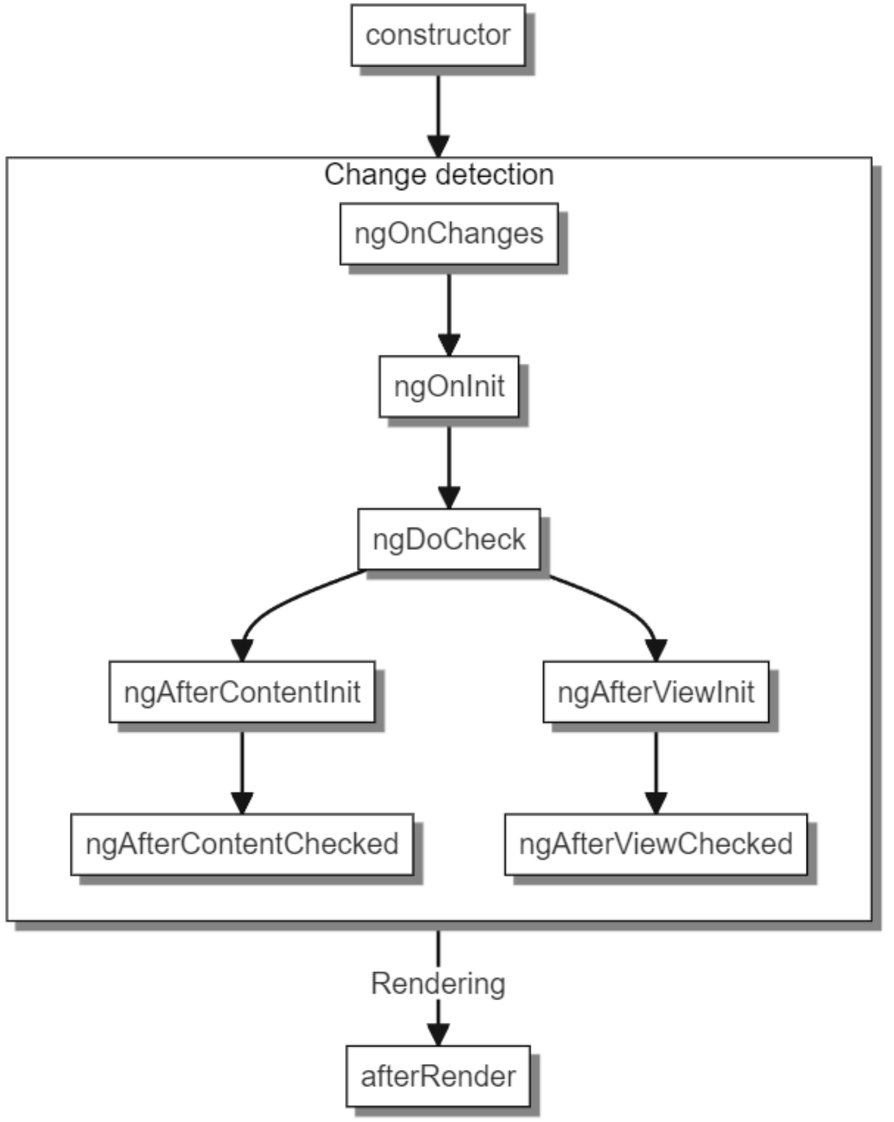
AfterContentChecked - аналогичен DoCheck(), только используется для контента, заключенного между тегами компонента;

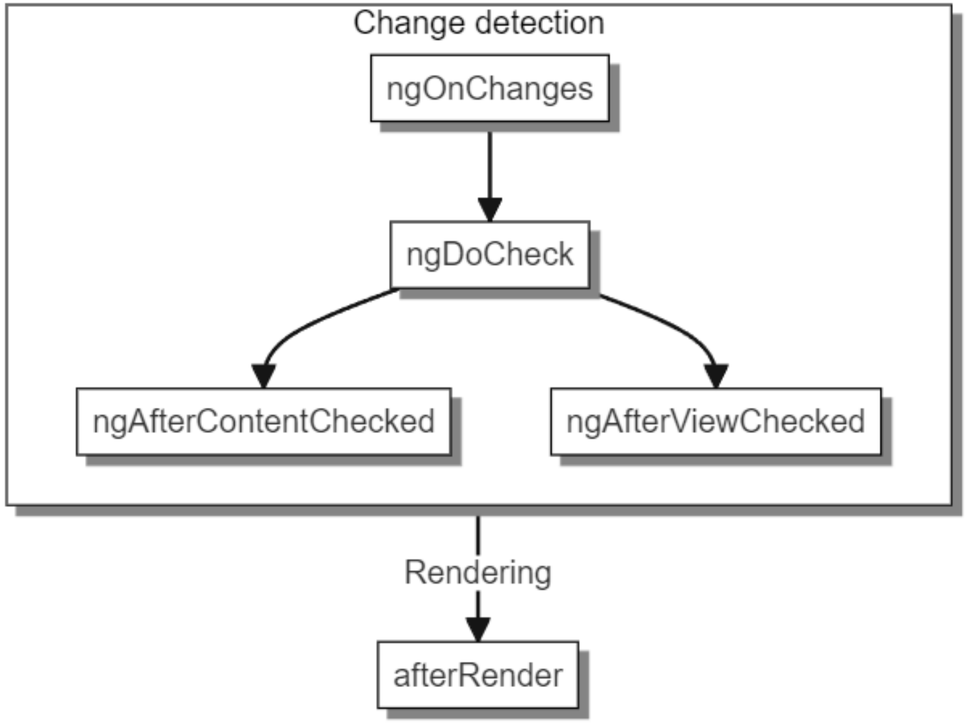
AfterViewInit - инициализируются компоненты, которые входят в шаблон текущего компонента;

AfterViewChecked - аналогичен DoCheck(), только используется для дочерних компонентов;

OnDestroy - компонент "умирает", т.е. удаляется из DOM-дерева




constructor
Change detection
ngOnChanges
ngOnInit
ngDoCheck
ngAfterContentInit
ngAfterViewInit
ngAfterContentChecked
ngAfterViewChecked
Rendering
afterRender


Change detection
ngOnChanges
ngDoCheck
ngAfterContentChecked
ngAfterViewChecked
Rendering
afterRender

Angular hooks реализованы в виде интерфейсов, реализующих функцию, совпадающую по названию с названием интерфейса + префикс ng.

```
export class ContactsItemComponent implements OnInit {
    //
    ngOnInit() {
        console.log('OnInit');
    }
    //
}
```

Директива

Предназначение директив — преобразование DOM заданным образом, наделение элемента поведением.

По своей реализации директивы практически идентичны компонентам, компонент — это директива с HTML-шаблоном, но с концептуальной точки зрения они различны.

Компоненты являются директивами. (Директивы с собственным template)

Есть два вида директив:

- структурные — добавляют, удаляют или заменяют элементы в DOM;
- атрибуты — задают элементу другое поведение.

Они создаются с помощью декоратора @Directive(), с конфигурационным объектом.

В Angular имеется множество встроенных директив (*ngFor, *ngIf), но зачастую их недостаточно для больших приложений, поэтому приходится реализовывать свои.

Примеры директив

- **Структурные встроенные (предваряются символом *)**

*ngIf добавляет или удаляет элемент из DOM-дерева в зависимости от истинности переданного выражения

```
<div *ngIf="true">This will be added to the DOM</div>
<div *ngIf="false">This will be removed from the DOM</div>
```

*ngFor используется для визуализации массива данных. Директива применяется к блоку HTML-кода, определяющего, как должны отображаться данные элемента массива. Далее Angular использует этот HTML как шаблон для всех последующих элементов в массиве.

ngSwitch эмулирует работу оператора switch применительно к шаблонам.

Примеры директив

- **Атрибуты встроенные**

[ngStyle] принимает объект, в котором ключами служат наименования CSS-свойств, а их значениями — возможные значения соответствующих CSS-свойств.

```
<div [ngStyle]="{color: 5 < 10 ? 'green' : 'red' }">
    Some text
</div>
```

[ngClass] также принимает объект, но ключами здесь служат наименования классов, а значениями — выражения типа Boolean. Если выражение истинно, класс будет добавлен к списку уже имеющихся классов.

```
<div [ngClass]="{'label': true}">Some text</div>
```

Пайп (Pipe, фильтр)

- Позволяют осуществлять преобразование формата отображаемых данных (например, дат или денежных сумм) прямо в шаблоне.
- Фильтры можно объединять в последовательности (pipe chains).
- Фильтры могут принимать аргументы.

```
<p>
  The chained hero's uppercase birthday is
  {{ birthday | date | uppercase}}
</p>

<p>
  The chained hero's uppercase birthday in "fullDate" format is
  {{ birthday | date:'fullDate' | uppercase}}
</p>
```

Сервис

Сервис - это класс, который является поставщиком данных. Сервисы инкапсулируют бизнес логику приложения. Сервисы могут предоставлять интерфейс взаимодействия между отдельными не связанными друг с другом компонентами.

- Реализуются в виде отдельных классов в соответствии с принципами ООП.

- Компонент может делегировать какие-либо из своих задач сервисам.

- Доступ компонентов к сервисам реализуется с помощью DI.

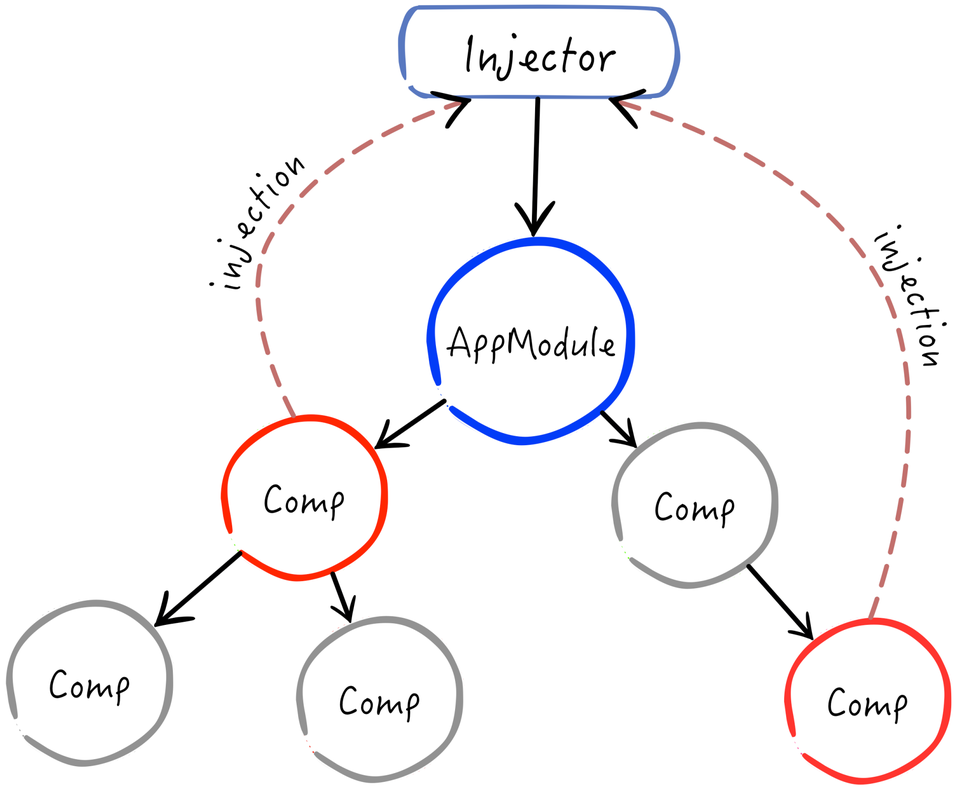
Пример сервиса

```
@Injectable({ providedIn: 'root' })
export class StatesService {
    private _filtersState: any = {
        accounts: {
            all: true,
            opened: false,
        },
        deposits: {
            all: true,
            opened: false,
        },
    };

    getFilters(): any {
        return this._filtersState;
    }
}
```

```
@Injectable({ providedIn: 'root' })
export class AccountsHttpService {
    constructor(private http: HttpClient) {}

    getUsers(): Observable {
        return this.http.get('/api/users');
    }
}
```

Dependency injection

- Компоненты могут использовать сервисы с помощью DI.
- Для того, чтобы класс можно было использовать с помощью DI, он должен содержать декоратор @Injectable().



Основные принципы реализации DI

- Приложение содержит как минимум один глобальный Injector, который занимается DI.
- Injector создаёт зависимости и передаёт их экземпляры контейнеру (container).
- Провайдер (provider) -- это объект, который сообщает Injector’у, как получить или создать экземпляр зависимости.
- Обычно провайдером сервиса является сам его класс.
- Зависимости компонентов указываются в качестве параметров их конструкторов

Провайдеры (providers) для сервисов

Для каждого сервиса должен быть зарегистрирован как минимум один провайдер.

Способы задания провайдера:

- в метаданных самого сервиса

```
@Injectable({providedIn: 'root'})
```

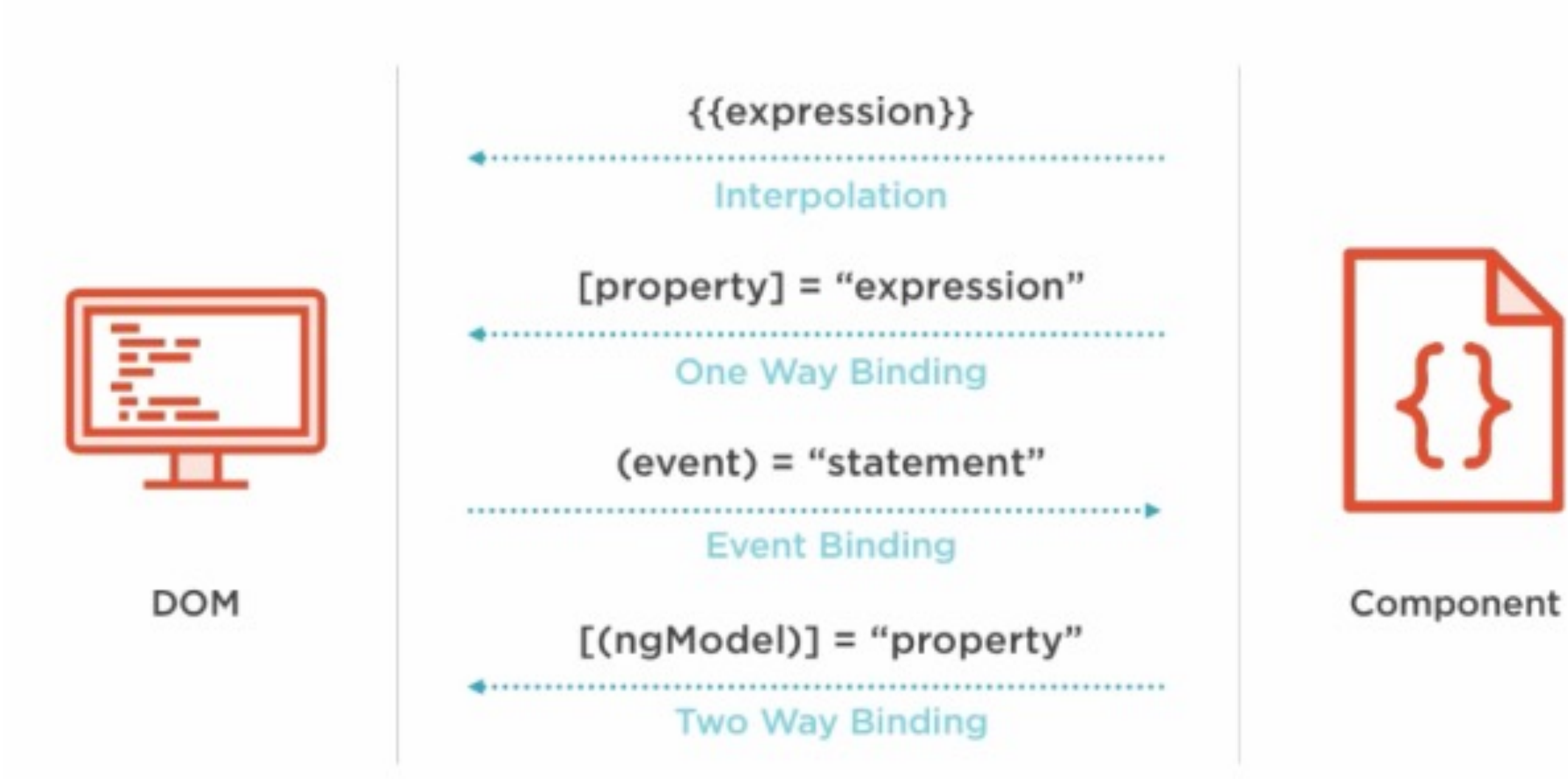
- в метаданных модуля

```
@Injectable({providedIn: AccountsModule})
```

- в метаданных компонента

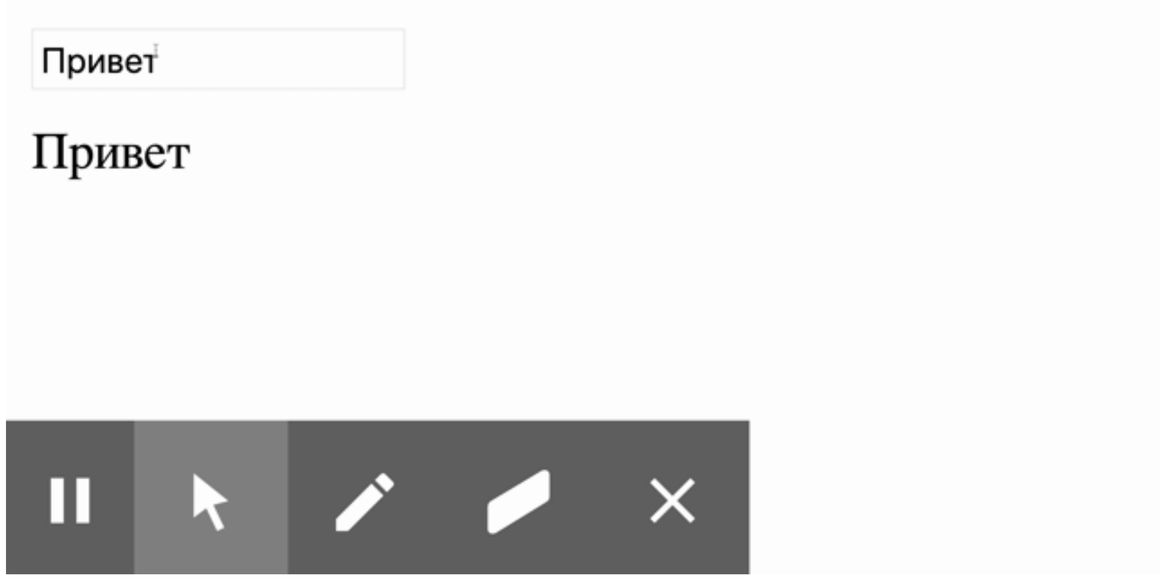
```
@Component({
  selector: 'accounts-list',
  providers: [AccountsHttpService],
  template: `<div>My accounts</div>`
})
```

Angular Bindings / Привязка данных



Two way binding

```
1  <contacts-item [(name)]="contactPerson"></contacts-item>
```



Change detection

ChangeDetection - это механизм в Angular, который отвечает за изменение выражений в шаблонах при их изменении в моделях.

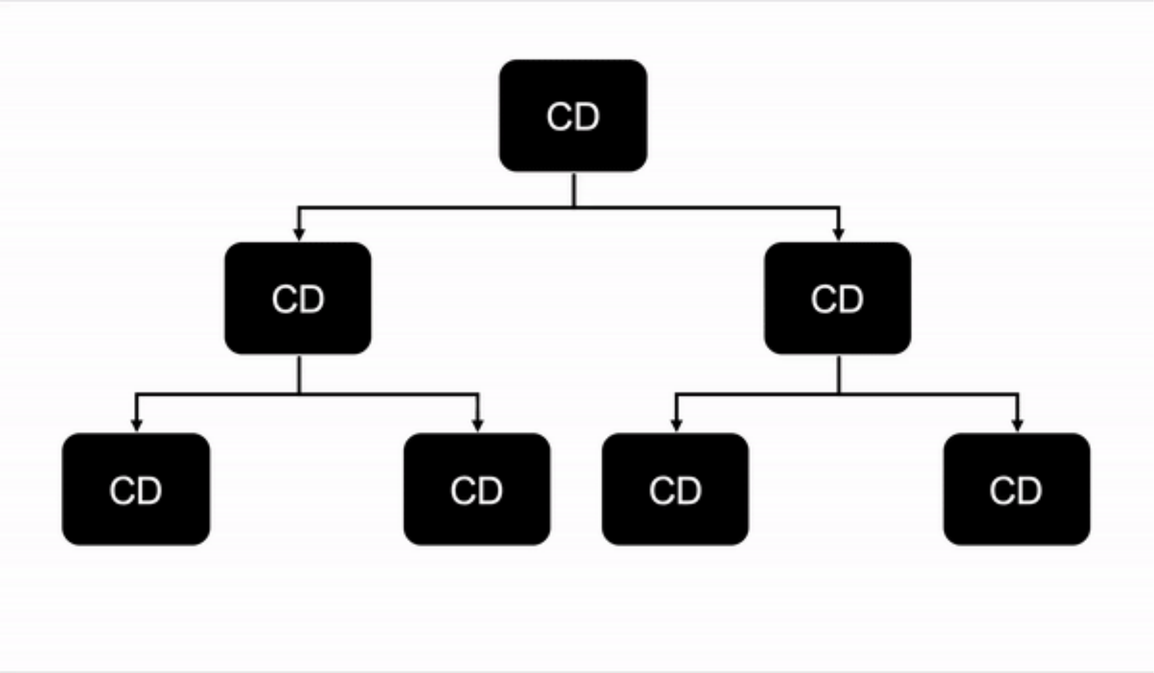
ChangeDetectionStrategy: **OnPush**, **Default**.

Сервис **ChangeDetectorRef** позволяет взять управление механизмом отслеживания изменений полностью под свой контроль.

Основные методы: **detach, detectChanges, reattach**

Change Detection

Проще представлять его в виде дерева компонентов, где на самом верху находится корневой компонент. Похожие механизмы реализованы не только в Angular, но и в других современных инструментах. Важно понимать этот механизм, так как он влияет на производительность, да и в целом дает представление о том, как работает ваше приложение.



Роутинг

ANGULAR
ROUTER

Маршруты

При определении маршрута можно указать ряд свойств:

•path — наименование маршрута;

•component — компонент для отображения при переходе на URL, совпадающий с path;

•children — одно из дополнительных свойств, объединяющее в себе группу маршрутов относительно текущего;

•redirectTo — перенаправляет на указанный URL при попадании на маршрут, указанный в path;

```
const routes: Routes = [
    { path: 'login', component: LoginRouteComponent },
    {
        path: 'home',
        component: HomeRouteComponent,
        children: [
            {
                path: 'profile',
                component: ProfileRouteComponent,
            },
        ],
    },
    {
        path: 'contacts',
        redirectTo: '/home',
        pathMatch: 'full',
        children: [
            {
                path: 'director',
                component: DirectorContactsRouteComponent,
            },
        ],
    },
    { path: '**', component: LoginRouteComponent },
];

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule],
})
export class AppRoutingModule {}
```

Маршруты

```
const routes: Routes = [
    {
        path: 'items',
        loadChildren: () =>
            import('./items/items.module').then(
                (m) => m.ItemsModule
            ),
    },
];
```

```
@NgModule({
    imports: [
        AppRoutingModule
    ],
    declarations: [
        AppComponent,
        LoginRouteComponent,
        HomeRouteComponent,
        ProfileRouteComponent,
        DirectorContactsRouteComponent
    ],
    providers: [],
    bootstrap: [AppComponent],
    exports: []
})
```

Router outlet

```
<div class="wrapper">
    <app-nav></app-nav>

    <main>
        <router-outlet></router-outlet>
    </main>

    <app-footer></app-footer>
</div>
```

Router

navigate(). В качестве первого параметра он принимает массив, где задается URL, а в качестве второго — объект с дополнительными параметрами запрашиваемого маршрута:

```
this.router.navigate(['profile', 3], {
    queryParams: { id: 3 },
    fragment: 'address',
});
```

```
<li>
    <a routerLink="profile" routerLinkActive="active-link"
        >Profile</a
    >
</li>
```

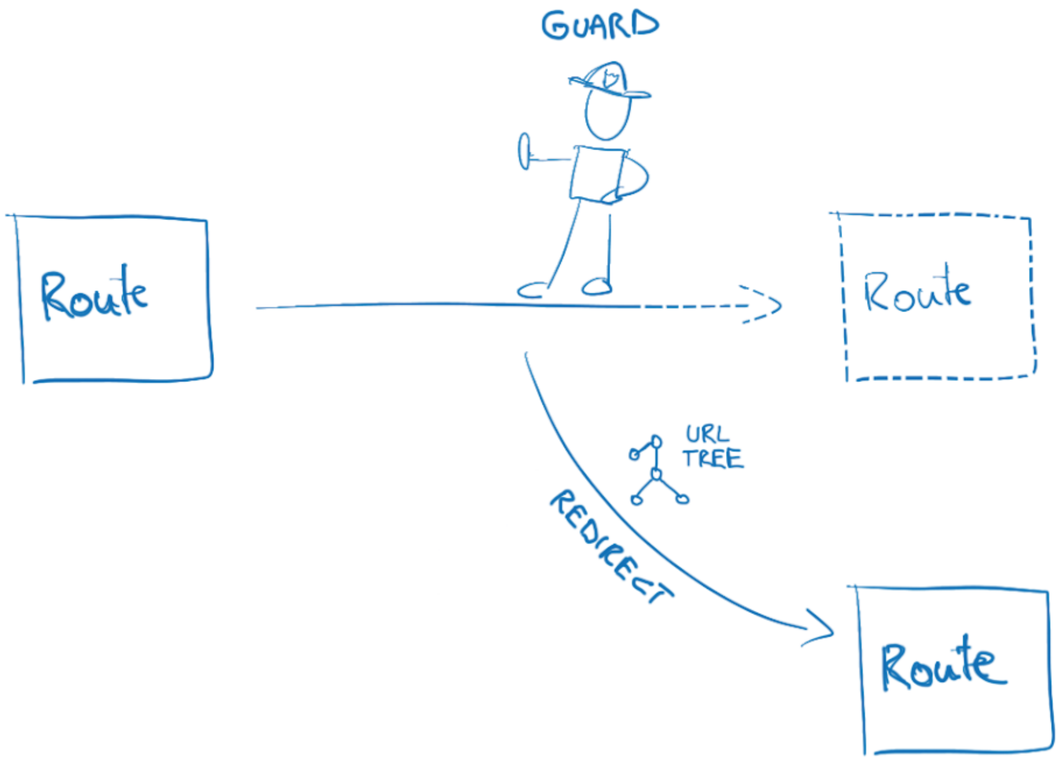
ActivatedRoute

- url — наименование маршрута;
- params — параметры Angular маршрутизации и их значения, указываемые при определении маршрута, например, id в /profile/:id;
- queryParams — параметры строки запроса, например, id в /profile?id=3;
- fragment — значение hash, например, address в /profile#address;
- data — объект одноименного свойства, указываемого при определении маршрута.

```
@Component({
    selector: 'app-profile',
    templateUrl: './profile.component.html',
    styleUrls: ['./profile.component.scss'],
})
export class ProfileComponent {
    constructor(private route: ActivatedRoute) {
        console.log(this.route);
    }
}
```

Ограничение доступов к маршруту

Route Guards позволяют ограничить доступ к маршрутам на основе определенного условия, например, только авторизованные пользователи с определенным набором прав могут просматривать страницу.



Route Guards

Различают следующие виды guard-ов:

- CanActivate — разрешает/запрещает доступ к маршруту;
- CanActivateChild -разрешает/запрещает доступ к дочернему маршруту;
- CanDeactivate — разрешает/запрещает уход с текущего маршрута;
- Resolve — выполняет какое-либо действие перед переходом на маршрут, обычно ожидает данные от сервера;
- CanLoad — разрешает/запрещает загрузку модуля, загружаемого асинхронно.

Создание Guards

```
@Injectable()
export class AuthGuard
    implements CanActivate, CanActivateChild {
    constructor(
        @Inject(AuthService) private auth: AuthService
    ) {}

    canActivate(
        next: ActivatedRouteSnapshot,
        state: RouterStateSnapshot
    ): boolean {
        return this.auth.isLoggedIn;
    }

    canActivateChild(
        next: ActivatedRouteSnapshot,
        state: RouterStateSnapshot
    ): boolean {
        return this.canActivate(next, state);
    }
```

```
const routes: Routes = [
    { path: 'login', component: LoginComponent },
    {
        path: 'pages',
        component: PagesComponent,
        canActivate: [AuthGuard],
        canActivateChild: [AuthGuard],
        children: [
            { path: 'about', component: AboutComponent },
            {
                path: 'contacts',
                component: ContactsComponent,
            },
        ],
    },
];

@NgModule({
    imports: [RouterModule.forChild(routes)],
    exports: [RouterModule],
})
```

Взаимодействие с back-end’ом

- Реализуется с помощью сервиса **HttpClient**.
- Для его использования необходимо импортировать модуль **HttpClientModule** в свой AppModule.
- После импорта модуля можно заинжектить **HttpClient**

```
1  @Injectable()
2  export class DataService {
3      constructor(private http: HttpClient) {}
4  }
```

Пример взаимодействие с back-end’ом

```
1  //GET-запрос на получение списка счетов
2  getAccounts(){
3      return this.http.get('http://example.com/api/accounts');
4  }
5
6  //GET-запрос на получение счета по id, id передается как GET-параметр
7  getAccountByID(id: number | string){
8      return this.http.get('http://example.com/api/accounts', {
9          params: new HttpParams().set('id', id)
10     });
11 }
```

Где почитать

Официальная документация

https://angular.io/

На русском

https://metanit.com/web/angular2/

Zone js in angular

https://medium.com/@lukaonik/what-is-ngzone-in-angular-762580ae37f6

Angular architecture

https://angular-academy.com/angular-architecture-best-practices/

Angular platform

https://medium.com/nuances-of-programming/%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D1%8B-angular-%D0%B2-%D0%B4%D0%B5%D1%82%D0%B0%D0%BB%D1%8F%D1%85-%D1%87%D0%B0%D1%81%D1%82%D1%8C-1-%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D1%8B-angular-bc5002b46e52

Doc на русском

https://webdraftt.com/

Doc RxJs

https://rxjs-dev.firebaseapp.com/

Ngrx – redux for angular

https://ngrx.io/