Angular

- вопросы для подготовки к собесу
- черновик - /frontend/framework/angular2.md

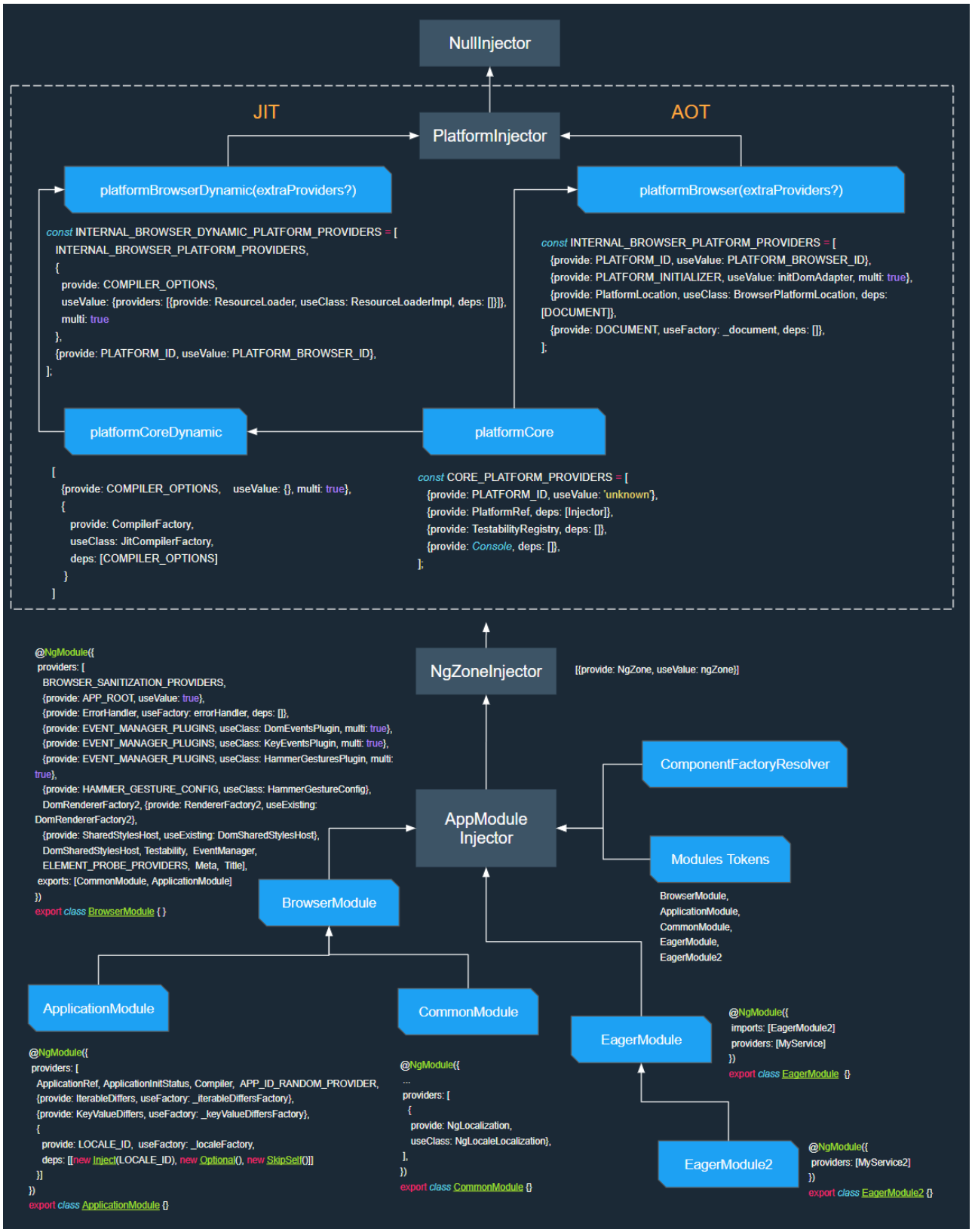
Angular

1. сравнение версий 2-10
    - https://github.com/angular/angular/blob/main/CHANGELOG.md
    - общие для большинства версий изменения: версии NS/RxJS, производительность, Material компоненты
    - 2016 - 2 - typescript, dart
        - changelogmd#200-proprioception-reinforcement-2016-09-14
    - 2017 - 4 - renderer v2
        - changelog
    - 2017 - 5 - HttpClientModule заменил http
        - changelog
    - 2018 - 6 - angular elements(web custom elements), tree-shakable providers(providedIn:root), rxjs 6, Ivy
        - changelog
    - 2018 - 7 - drag and drop, node v10, service worker
        - changelog
    - 2019 - 8 - webworker, ES6 импорт модулей `()=>`
        - changelog
    - 2020 - 9 - Ivy по-умолчанию, fullTemplateTypeCheck, strictTemplates в tsc
        - changelog
    - 2020 - 10 - исправили 700 и посмотрели 2000 ошибок, компонент диапазона дат
        - changelog
    - 2020 - 11 - harnesses for all of the components, webpack 5 experimental, ng serve --hmr, tslint deprecated, IE9/10 deprecated
        - changelog
    - 2021 - 12 - nullish coalescing, tailwind css, webpack 5 prod, IE11 deprecated
        - changelog
    - 2021 - 13 - new API removes the need for ComponentFactoryResolver being injected into the constructor. Ivy creates the opportunity to instantiate the component with ViewContainerRef.createComponent without creating an associated factory
        - changelog
    - 2022 - 14 - Strictly Typed Reactive Forms; standalone components, directives and pipes: add imports directly in your @Component() without an @NgModule()
        - changelog
        - standalone components dev preview
    - 2022 - 15 - ngmodule --> standalone migration
        - changelog
        - deprecating providedIn: NgModule. If you should truly scope providers to a specific NgModule, use NgModule.providers instead

        - default formatting configuration for DatePipe
        - ng g component --standalone
        - experimental esbuild support
        - Component Dev Kit - listbox
        - refactoring of the Angular material components based on Material Design Components for Web (MDC) is now done
        - Better stack traces
2. Архитектура
    - MVVM
    - зачем - потому что автоматическое связывание data binding(единственное отличие от MVP)
    - достоинства
    - недостатки
3. Различие между AngularJS и Angular
    - rxjs, DI
4. преимущества Angular
    - поскольку вместо virtualDOM у нас shadowDOM и декларативный стиль HTML вместо императивного JS/аннотации
    - не надо делать обёртки для сторонних UI компонентов
    - Много библиотек с поддержкой вендора(router/ngModule), покрывающих все типовые нужды SPA
    - чёткие подробные наставления
    - релиз каждые 6 мес с полной обратной совместимостью
    - не нужно думать о совместимости версий внешних библиотек, всё включено
    - DI - контроль количества экземпляров, изоляция, оптимизация
    - мнения https://gist.github.com/irustm/375a9db35be6273368ac16be9e844cfa
5. недостатки
    - больший размер из-за Angular CLI, но после компиляции его убирают. В старом режиме JIT компилятор на борту также увеличивает объём.
    - Приложение может быть сложным, Angular - это и полный framework, и platform с набором инструментов. Из-за этого его сложнее учить. В отличие от React тут включены по-умолчанию управление состоянием, сервисы, HTTP библиотека, DI, валидация форм, маршрутизация URL.
6. ngModule
    - declarations - только компоненты, конвейеры и директивы
        - нельзя декларировать импортированные модули или уже декларированные компоненты
    - imports - импортировать декларации из другого модуля
    - exports - сделать видимыми для импорта свои декларации
    - providers - сделать видимыми для экспорта сервисы и другие внедряемые зависимости
7. веб-компоненты, custom elements
    - элементы - реализации обёртки над веб-компонентами, расширение DOM API
    - стандартный подход к компонентам без применения ангуляр специфичного синтаксиса
    - можно написать кастомный элемент формы, например, кастомный select.
    - Статические компоненты рисуются без участия ангуляр, на стороне браузера

    - динамические компоненты без применения dynamic components `componentFactoryResolver.resolveComponentFactory`, т.е. меньше кода для встраивания логики в отрисовку.
    - https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_custom_elements `customElements.define('word-count', WordCount, { extends: 'p' });`
    - https://angular.io/guide/elements
    - Elements in v6 and Beyond - Rob Wormald
8. JIT Deep Dive into the Angular Compiler | Alex Rickabaugh | 2019
    - перед чтением полей
    - в декораторах
    - простой тайпскрипт
    - Больше объём приложения с Angular compiler.
9. AOT
    - компилирует код за декораторами `@`
    - Лучше отрисовка с AOT. Браузер рисует без компиляции.
    - Меньше XHR запросов. Компилятор сразу добавляет в код HTML+CSS.
    - Меньше объём приложения без Angular compiler.
    - Определяет ошибки шаблонов/связывания во время компилирования.
    - Лучше ИБ из-за уменьшения загрузок и выполнения кода на клиенте
    - нельзя использовать стрелочные функции и функциональные выражения внутри метаданных за декораторами @
10. ivy
    - Ivy - новый компилятор и отрисовщик. По-умолчанию с 9 версии
    - локальная компиляция классов для лучшего деревотрясения. Компонент может быть вне модуля
    - https://github.com/angular/angular/blob/master/packages/compiler/design/architecture.md
    - готовность ivy
    - генерирует меньше кода, он более понятный, его легче отлаживать - ссылки в консоли на шаблон
    - angular compiler
    - lazyLoad ivy
    - Angular In Depth workshop on dynamic rendering with Ivy
    - https://medium.com/ngx/angular-ivy-renderer-%D0%B3%D0%BE%D1%82%D0%BE%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C-93-20e0aff2e861

DI

1. 

2. Иерархия Injector'ов, какие бывают, сколько их может быть?
    - https://angular.io/guide/hierarchical-dependency-injection
    - https://medium.com/angular-in-depth/angular-dependency-injection-and-tree-shakeable-tokens-4588a8f70d5d
    - Что вы знаете о функции inject? - можно внедрить компонент без модулей и передать туда данные
    - две иерархии, чтобы не двоились lazy load зависимости

        - первая: ElementInjector иерархия для каждого DOM элемента. ElementInjector пустой пока нет providedIn в @Directive() или @Component()
        - вторая: ModuleInjector иерархия через @NgModule() или @Injectable()
        - разрешаются через mergeInjector `constructor(private injector: Injector) {}`
    - иерархия NullInjector()-->platformModuleInjector(Renderer, Sanitizer)-->rootModuleInjector(services)-->componentInjector(ElementRef, ChangeDetectorRef)
    - @Injectable() предпочтительнее @NgModule() providers, т.к. позволяют tree-shaking
    - `platformBrowserDynamic()` создаёт `platformModuleInjector` для platform/browser специфичных зависимостей как domsanitizer. Настраивается через platformBrowser().extraProviders
    - можно создать дочерние ModuleInjectors при ленивой загрузке
    - стратегии разрешения зависимостей: @Optional(), @Self(), @SkipSelf(), @Host() https://angular.io/guide/dependency-injection-in-action#make-a-dependency-optional-and-limit-search-with-host
        - @Optional() присваивает null не найденным провайдерам сервисов. Нужен для игнорирования ошибок
        - @Self() ищет в текущем ElementInjector component или directive
        - @SkipSelf() ищет в родительском ElementInjector
        - @Host() ищет в component и ниже по вложенному дереву

```
    constructor(
        @Host()     // limit search for logger; hides the
application-wide logger
        @Optional() // ok if the logger doesn't exist
        private loggerService?: LoggerService
    ) {
```

3. Управление экземплярами https://angular.io/guide/dependency-injection-in-action
    - песочница, отдельные экземпляры: регистрируем в providers аннотациях компонентов https://angular.io/guide/dependency-injection-in-action#multiple-service-instances-sandboxing
    -
4. Сколько у нас инжекторов, у одного модуля и трёх компонентов?
5. Dependency injection
    - провайдер
        - Объект, который реализует один из интерфейсов Provider
        - предоставляет инжектору порядок разрешения зависимости, связанной с токеном/идентификатором
        - может предоставлять разные реализации одной и той же зависимости

    - multi для расширения токена новыми зависимостями
        - https://blog.thoughtram.io/angular2/2015/11/23/multi-providers-in-angular-2.html

```
var injector = Injector.create([
{ provide: Engine, deps: []},
{ provide: Engine, useClass: TurboEngine, deps: [] }
]);
```

- токен
    - ключ в связке с провайдером `constructor(token: Type)`
    - объект, который реализует интерфейс InjectionToken
- инжектор
    - Объект(абстрактный класс), который находит именованную зависимость в своём кэше, либо создаёт её используя провайдер
    - Предоставляет и внедряет синглтон
    - Создаются автоматом для модулей в ходе bootstrap и наследуется в иерархии компонентов
- зачем
    - для работы с сервисами и модулями
    - меньше кода в конструкторах
    - облегчение рефакторинга, автоматическая инъекция зависимостей по всей цепочке
    - облегчение юнит-тестирования сервисов
    - переиспользование сервисов
- в версии 15 появился standalone component - обёртка компонентов без модулей

```
// импорт внутрь автономного
@Component({
    standalone: true,
    selector: 'photo-gallery',
    imports: [RegularComponent, RegularModule],
    template: `
        ... <image-grid [images]="imageList"></image-grid>
    `,
})
export class StandaloneComponent {
    // component logic
}

// импорт автономного в обычный модуль
@NgModule({
```

```
    declarations: [AlbumComponent],
    exports: [AlbumComponent],
    imports: [StandaloneComponent],
})
export class AlbumModule {}

// можно стартовать в main.ts из автономного компонента
bootstrapApplication(StandaloneComponent);
```

- синглтон https://angular.io/guide/architecture-services#providing-services
  - для всего приложения: в аннотации компонента `@Injectable({providedIn: 'root'})` https://angular.io/api/core/Injectable#injectable
    - root: для приложения
    - platform - для всех приложений
    - в любой модуль
  - для модуля: в модуле `@NgModule({providers: [...]`
  - для компонента: в аннотации компонента `@Component({...,providers: [ HeroService ],...})`
  - используется
    - в маршрутизаторе для хранения по одной копии состояния маршрута
- не синглтон когда сервис регистрируется в компоненте

```
@Component({
    selector:    'app-hero-list',
    templateUrl: './hero-list.component.html',
    providers:  [ HeroService ]
})
```

- RouterModule.forRoot: модуль с providers
- RouterModule.forChild: отдельный экземпляр модуля без providers для ленивой загрузки
- нельзя внедрять интерфейсы https://angular.io/guide/dependency-injection-providers#interfaces-and-dependency-injection , но можно абстрактные классы https://angular.io/guide/dependency-injection-in-action#class-interface
- внедрение лёгких токенов для уменьшения размера библиотек

```
@Component({selector: 'lib-header',...,})
class LibHeaderComponent {}

@Component({selector: 'lib-card',...,})
class LibCardComponent {
    @ContentChild(LibHeaderComponent)
    header: LibHeaderComponent|null = null;
}
```

- внедрение классов

```
    // более многословный вариант стандартного внедрения
    { provide: HeroService,   useClass:    HeroService },
    // подмена класса
    { provide: LoggerService, useClass:    DateLoggerService }
```

- внедрение псевдонимов

```
    providers: [
        NewLogger,
        // Alias OldLogger w/ reference to NewLogger
        { provide: OldLogger, useExisting: NewLogger}
    ]
```

- внедрение/разрыв циклической зависимости

```
    providers: [{ provide: Parent, useExisting: forwardRef(() =>
AlexComponent) }],
```

- внедрение не классов

```
    // src/app/app.config.ts
    import { InjectionToken } from '@angular/core';
    export const APP_CONFIG = new InjectionToken<AppConfig>
('app.config');

    // src/app/providers.component.ts
    providers: [{ provide: APP_CONFIG, useValue: HERO_DI_CONFIG
}]

    // src/app/app.component.ts
    constructor(@Inject(APP_CONFIG) config: AppConfig) {
        this.title = config.title;
    }
```

- внедрение по условию

```
    // src/app/heroes/hero.service.ts
    constructor(
        private logger: Logger,
        private isAuthorized: boolean
    ) { }
```

```
    getHeroes() {
        const auth = this.isAuthorized ? 'authorized ' :
'unauthorized';
        this.logger.log(`Getting heroes for ${auth} user.`);
        return HEROES.filter(hero => this.isAuthorized ||
!hero.isSecret);
    }

    // src/app/heroes/hero.service.provider.ts
    const heroServiceFactory = (logger: Logger, userService:
UserService) => {
        return new HeroService(logger,
userService.user.isAuthorized);
    };

    // src/app/heroes/hero.service.provider.ts
    export let heroServiceProvider = {
        provide: HeroService,
        useFactory: heroServiceFactory,
        deps: [Logger, UserService]
    };
```

- внедрение в DOM через ElementRef

```
@Directive({selector: '[appHighlight]'})
export class HighlightDirective {

    @Input('appHighlight') highlightColor: string;

    private el: HTMLElement;

    constructor(public el: ElementRef, public control: NgModel) {
        this.el = el.nativeElement;
        @HostBinding('class.valid') get valid() { return
this.control.valid; }
        @HostBinding('class.invalid') get invalid() { return
this.control.invalid; }
    }

    @HostListener('mouseenter') onMouseEnter() {
        this.highlight(this.highlightColor || 'cyan');
    }

    @HostListener('mouseleave') onMouseLeave() {
        this.highlight(null);
    }

    private highlight(color: string) {
        this.el.style.backgroundColor = color;
    }
}
```

```
// HTML
<div appHighlight="yellow">
<input [(ngModel)]="prop">
```

- hostbinding
  - перехват изменения свойств DOM
- hostlistener
  - перехват событий DOM

Конвейеры, директивы и компоненты

1. связывание данных

```
    <!-- значение переменной, присвоение в свойство -->
    <img [src]="itemImageUrl">
    <!-- просто строка, присвоение в атрибут -->
    <app-item-detail childItem="parentItem"></app-item-detail>
    <!-- значение переменной, присвоение в атрибут -->
    <app-item-detail childItem="{{parentItem}}"></app-item-detail>
    <!-- верблюжий регистр -->
    <tr><td [colSpan]="1 + 1">2</td></tr>
    <!-- нижний регистр -->
    <tr><td colspan="{{1 + 1}}">2</td></tr>
    <p><img src="{{itemImageUrl}}"> is the <i>interpolated</i> image.
</p>
    <!-- присвоение в атрибут интерполированного значения, некоторые
атрибуты этого требуют-->
    <p><img attr.src="{{itemImageUrl}}"> is the <i>interpolated</i>
image.</p>
    <p><img alt="Interpolated item" src="{{itemImageUrl}}"> is the
<i>interpolated</i> image.</p>
    <p><img [src]="itemImageUrl"> is the <i>property bound</i> image.
</p>
    <p><span>"{{interpolationTitle}}" is the <i>interpolated</i>
title.</span></p>
    <p>"<span [innerHTML]="propertyTitle"></span>" is the <i>property
bound</i> title.</p>
```

2. https://angular.io/guide/user-input

```
    <button type="button" (click)="onClickMe()">Click me!</button>
    <!-- passing $event breaks the separation of concerns between the
template (what the user sees) and the component (how the application
processes user data). -->
    <input (keyup)="onKey($event)">
    <!-- correct -->
    <input #box (keyup)="onKey2(box.value)">
    <input #box (keyup.enter)="onEnter(box.value)">
```

```
    <input #box
        (keyup.enter)="update(box.value)"
        (blur)="update(box.value)">
```

```
    onKey(event: KeyboardEvent) { // with type info
        this.values += (event.target as HTMLInputElement).value + ' |
';
    }

    onKey2(value: string) {
        this.values += value + ' | ';
    }
```

3. защита от null/undefined https://angular.io/guide/template-expression-operators

```
    <!-- Assert color is defined, even if according to the `Item` type
it could be undefined. -->
    <p>The item's color is: {{item.color!.toUpperCase()}}</p>
    <p>The item's undeclared best by date is:
{{$any(item).bestByDate}}</p>
```

4. Какие обязательные props для Component
    - template
    - style
5. Разница поведения между ng-if и visibility: hidden
    - `ngif` удаляет элемент из DOM
    - можно прятать структурные директивы в ng-container
6. В чем разница между Directive и Component.
    - компонент - частный случай директивы, с шаблоном
    - Структурные директивы — манипуляции с DOM: `ngif, ngfor`
        - Обязательно импортируют: Input, TemplateRef, ViewContainerRef
        - this.viewContainer.createEmbeddedView(this.templateRef);
    - Атрибутные директивы — манипулирует: element, component, directive. `ngStyle, ngClass`

      ```
      @Directive({
          selector: '[appHighlight]'
      })
      ```

Директивы

- https://angular.io/guide/built-in-directives
- ngClass
  - https://angular.io/api/common/NgClass#description
  - `ngClass: string | string[] | Set<string> | { [klass: string]: any; }`

```
    <div [ngClass]="isSpecial ? 'special' : ''">
    <div [ngClass]="currentClasses">  <!-- Лучше вместо функции ввести
переменную-->
    <some-element [ngClass]="'first second'">...</some-element>
    <some-element [ngClass]="['first', 'second']">...</some-element>
    <some-element [ngClass]="{'first': true, 'second': true, 'third':
false}">...</some-element>
    <some-element [ngClass]="stringExp|arrayExp|objExp">...</some-
element>
    <some-element [ngClass]="{'class1 class2 class3' : true}">...
</some-element>

    <div [class.active]="isActive">  <!-- так классы видно в
отладчике, иначе 'class=[Object object]' -->
    <div [class]="{'odd-row': odd, 'second': true}">
```

```
    currentClasses: {};
    // src/app/app.component.ts
    setCurrentClasses() {
        // CSS classes: added/removed per current state of component
properties
        this.currentClasses =  {
        saveable: this.canSave,
        modified: !this.isUnchanged,
        special:  this.isSpecial
        };
    }
```

- ngStyle

```
    <div [ngStyle]="currentStyles">
```

```
    currentStyles: {};
    /* . . . */
    setCurrentStyles() {
```

```
        // CSS styles: set per current state of component properties
        this.currentStyles = {
        'font-style':  this.canSave      ? 'italic' : 'normal',
        'font-weight': !this.isUnchanged ? 'bold'   : 'normal',
        'font-size':   this.isSpecial    ? '24px'   : '12px'
        };
    }
```

- NG-FOR. TrackBy зачем нужен, преимущества.
    - для более быстрого вычисления признака замены значений вложенного объекта в ячейке массива, быстрота перерисовки

Стили Material

- cdk-virtual-scroll-viewport - появился в 7 версии для бесконечной прокрутки, использует событие scroll
- https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API
- scss variables лучше через миксины, не требуется делать лишние @include

```
@mixin variables() {
      --color-main: #121212;
  }
// ...
:root {
  .theme-light {
      @include variables();
      @include light-style();
      //@include angular-material-theme(theme-light);
  }
  .theme-dark {
      @include variables();
      @include dark-style();
      //@include angular-material-theme(theme-dark);
  }
}
// ...
color: var(--color-main);
```

LifeCycle Hooks. Назовите основные, которые используете в приложении.

```
* ngOnChanges()

    ```ts
        ngOnChanges(changes: SimpleChanges) {
        for (let propName in changes) {
            let chng = changes[propName];
```
```

```
            let cur  = JSON.stringify(chng.currentValue);
            let prev = JSON.stringify(chng.previousValue);
        }}
    ```
* ngOnInit()
* ngDoCheck()
* ngAfterContentInit()
* ngAfterContentChecked()
* ngAfterViewInit()
* ngAfterViewChecked()
* ngOnDestroy() - Отписка от Observables, DOM events, остановка таймеров, Unregister all callbacks that the directive registered with global or application services.
```

1. Template variables. Как объявить. Зачем нужны?
    - # или ref- в html, для доступа к свойствам элемента внутри html шаблона
    - ссылается на: DOM element, directive (which contains a component), an element, TemplateRef, or a web component.
    - Область видимости - весь шаблон, потому одинаковые переменные будут содержать непредсказуемые ссылки.
2. TemplateRef, ElementRef, в чем разница?
    - ссылка на ng-template или любой элемент
    - шаблон можно передать в директиву ngTemplateOutlet а элемент в componentOutlet
        - <ng-container *ngTemplateOutlet='someVar'>
        - <ng-container *ngComponentOutlet='someVar'>
3. template input variable
    - https://angular.io/guide/template-reference-variables#template-input-variable
    - у let-* и #* переменных разные пространства имён

```
<!--            объявление и присвоение  -->
<ng-template #hero let-hero let-i="index" let-odd="isOdd">
    <div [class]="{'odd-row': odd}">{{i}}:{{hero.name}}</div>
</ng-template>
```

    - охват входящих переменных ограничен текущим экземпляром шаблона а не всеми в виде наследования
4. В чем отличие ContentChild vs ViewChild
    - одинаковые по способностям
    - ContentChild
        - ng-content внешний HTML, transclusion, content projection
```

- доступ после ngContentInit
- https://angular.io/api/core/ContentChildren
- https://angular.io/api/core/ContentChild

```
    // Query for a CONTENT child of type `ChildComponent`
    @ContentChild(ChildComponent) contentChild!: ChildComponent;

    ngAfterContentInit() {
        // contentChild is set after the content has been
initialized
    }

    ngAfterContentChecked() {
        // contentChild is updated after the content has been
checked
        this.prevHero = this.contentChild.hero;
        }
    }
```

- viewchild

- для доступа к DOM после рендеринга afterViewInit
- shadowDom/JS
- https://angular.io/api/core/ViewChild
- https://angular.io/guide/glossary#view-hierarchy
- https://angular.io/guide/lifecycle-hooks#responding-to-view-changes
- доступ после ngAfterViewInit
- прямой доступ в DOM нежелательно использовать при рендеринге на сервере из-за ИБ
- можно ссылаться на директивы по имени и типу, на шаблонные переменные #
- можно читать/писать в @Input компонента
- https://angular.io/api/core/ViewChildren
- https://angular.io/api/core/AfterViewChecked вызывается очень часто
- view query. The change detector looks for the first element or the directive matching the selector in the view DOM. The following selectors are supported:
  - класс @Component or @Directive decorator
  - A template reference variable as a string (e.g. query <my-component #cmp> with @ViewChild('cmp'))
  - Any provider defined in the child component tree of the current component (e.g. @ViewChild(SomeService) someService: SomeService)
  - Any provider defined through a string token (e.g. @ViewChild('someToken') someTokenVal: any)
  - A TemplateRef (e.g. query with @ViewChild(TemplateRef) template;)
-

```
    // Accessing DOM element with JavaScript
    let domReference = document.getElementById("someElement");
```

```
    // Access DOM element using Angular @ViewChild
    @ViewChild("someElement") domReference;
    @ViewChild(NgModel) userNameReference: NgModel;
    @ViewChild("userInformation") childComponentReference: any;

    ngAfterViewInit(): void {
        this.domReference.nativeElement.focus();
        this.userNameReference.valueChanges.subscribe(() => {
 this.executeOtherFunction() })
        // Accessing Property of Child Component
        this.childComponentReference.userName = "Updated Name";

        // Accessing Functions of Child Component
        this.childComponentReference.updateUserName();
    }
```

- viewChildren - Директива @ViewChild отличается от @ViewChildren тем, что первая всегда вернет вам только один элемент, в то время как вторая позволяет вам находить несколько элементов, возвращая вам объект типа QueryList.

```
    @ViewChildren("NgModel") domReference: QueryList<NgModel>;
    @ViewChildren("UserDetailComponent") userDetailReferences:
QueryList<NgModel>;
    @ViewChildren("userName, userAge, userDesignation")
userInfoReference: QueryList<NgModel>;

    ngAfterViewInit(): void {
        console.log("Element List: " + this.domReference.length);
        console.log("Element List: " +
this.userDetailReferences.length);
    }
```

5. динамическое создание компонентов - как создать динамически компонент, который лежит во внешнем файле, а также вставлять его в DOM из нашего сервиса
   - content projection https://angular.io/guide/content-projection
   -

```
    //ссылки через DI на себя:
    constructor(
        private templateRef: TemplateRef<any>,
        private el: ElementRef
    ){}{}

    //viewChild
    @ViewChild('input') input;

    ngAfterContentInit() {
```

```
        this.input.nativeElement.focus();
    }

    //renderer2 - сделан для SSR nodejs
    let inputElement = this.renderer.createElement('input');
    this.renderer.appendChild(parent, inputElement);
    this.renderer.setProperty(inputElement, 'checked', true);

    //DI
    constructor(
        private renderer: Renderer2,
        private elementRef: ElementRef
    ) {}

    @Input() set content(value: string) {
        let buttonElement = this.renderer.createElement('button');
        this.renderer.appendChild(this.elementRef.nativeElement,
buttonElement);
    }
```

- viewContainer - createEmbeddedView

```
    constructor(
        private viewContainerRef: ViewContainerRef
    ){}

    // viewContainer - можно создать
Host(component)/Embedded(template) view
    //ангуляр не вставляет View-элемент внутрь указанного
контейнера, а добавляет его сразу после контейнера
    //динамически добавляемые компоненты не поддерживают Input- и
Output-декораторы
    this._contentViewRef = this.popover.createEmbeddedView();
    const componentFactory =
this._cfResolver.resolveComponentFactory(Popover);
    this._componentRef = this.viewContainerRef.createComponent(
        componentFactory,
        this._injector,
        0,
        [this._contentViewRef.rootNodes]
    );

    this._componentRef.instance.title = this.title;
    this._contentViewRef.detectChanges();
```

- templateOutlet

```
    <ng-container *ngTemplateOutlet="svk; context: myContext">
</ng-container>
```

```
    <ng-template #svk><span>Hello</span></ng-template>
```

```
    <!-- content-projection/src/app/example-zippy.template.html -
->
    <ng-container [ngTemplateOutlet]="content.templateRef"></ng-
container>
    <!-- content-projection/src/app/app.component.html -->
    <ng-template appExampleZippyContent>
        It depends on what you do with it.
    </ng-template>
```

```
    //content-projection/src/app/example-zippy.component.ts
    @Directive({
        selector: '[appExampleZippyContent]'
    })
    export class ZippyContentDirective {
        constructor(public templateRef: TemplateRef<unknown>) {}
    }

    @ContentChild(ZippyContentDirective) content!:
ZippyContentDirective;
```

```
    <p question>
        Is content projection cool?
    </p>
    <ng-container ngProjectAs="[question]">
        <p>Is content projection cool?</p>
    </ng-container>
```

- componentOutlet

```
    <ng-container *ngComponentOutlet="HelloWorld"></ng-container>
```

```
    class ComponentOutletExample {
        public HelloWorld = HelloWorldComponent
```

6. Как получить доступ к HTML Element из компонента.
   - https://angular.io/guide/component-interaction
   - шаблонные переменные в родителе дают доступ в HTML к свойствам компонента

```
    <button (click)="timer.start()">Start</button>
    <button (click)="timer.stop()">Stop</button>
    <div class="seconds">{{timer.seconds}}</div>
    <app-countdown-timer #timer></app-countdown-timer>
```

- @viewChild даёт доступ к экземпляру другого компонента

```
    @Component({
    selector: 'app-countdown-parent-vc',
    template: `
        <button (click)="start()">Start</button>
        <button (click)="stop()">Stop</button>
        <div class="seconds">{{ seconds() }}</div>
        <app-countdown-timer></app-countdown-timer>
    `,
    styleUrls: ['../assets/demo.css']
    })
    export class CountdownViewChildParentComponent implements
AfterViewInit {

    @ViewChild(CountdownTimerComponent)
    private timerComponent: CountdownTimerComponent;

    seconds() { return 0; }

    ngAfterViewInit() {
        // Redefine `seconds()` to get from the
`CountdownTimerComponent.seconds` ...
        // but wait a tick first to avoid one-time devMode
        // unidirectional-data-flow-violation error
        setTimeout(() => this.seconds = () =>
this.timerComponent.seconds, 0);
    }

    start() { this.timerComponent.start(); }
    stop() { this.timerComponent.stop(); }
    }
```

- ElementRef даёт доступ к собственному DOM
  - https://angular.io/guide/testing-components-basics#nativeelement
  - https://angular.io/guide/attribute-directives
  - https://angular.io/guide/dependency-injection-in-action#inject-the-components-dom-element

```
@Directive({selector: '[appHighlight]'})
export class HighlightDirective {
    constructor(el: ElementRef) {
    el.nativeElement.style.backgroundColor = 'yellow';
    }
}
```

```
```

- ViewContainerRef - ещё позволяет создавать дочерние элементы createEmbeddedView

7. ViewEncapsulation. Какая бывает, зачем нужна?
    - Emulated - CSS обёртка для эмуляции стандартного поведения. если не объявлены templates/templateUrls переключается в None.
    - None - для наследования общих стилей
    - shadowDom - для прямого доступа к изолированным shadow DOM узлам
    - https://angular.io/guide/view-encapsulation
    - https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM
8. Change detection. Какие есть стратегии, для чего используются? Какие есть методы, чтобы запустить детектор?
    - https://angular.io/guide/zone#fundamentals-of-change-detection
    - https://angular.io/guide/glossary#change-detection
    - https://web.dev/faster-angular-change-detection/
    - https://angular.io/api/core/ChangeDetectorRef#usage-notes
    - ангуляр манкипатчит browser API - mouse/input/keyb events, xhr, promise, async/await, webworkers,
    - распространяется на вложенные компоненты
    - default ("CheckAlways") - the change detector goes through the view hierarchy on each VM turn to check every data-bound property in the template. In the first phase, it compares the current state of the dependent data with the previous state, and collects changes. In the second phase, it updates the page DOM to reflect any new data values.
    - OnPush ("CheckOnce") -
        - https://angular.io/guide/lifecycle-hooks#using-change-detection-hooks
        - https://angular.io/api/core/ChangeDetectorRef#usage-notes
        - влияет на хук ngOnChanges
        - ручная проверка
        - поменялась @Input ссылка(не значение)
        - DOM event(input) для связанных свойств

- async pipe(rxjs, promise)

```
    @Component({
        changeDetection: ChangeDetectionStrategy.OnPush
    })

    @Input() set live(value: boolean) {
        if (value) {
            this.changeDetectorRef.reattach(); // вернуть в
дерево
        } else {
            this.changeDetectorRef.detach();
        }
    }

    constructor(private ref: ChangeDetectorRef) {
        this.changeDetectorRef.markForCheck(); // помечает как
diry

        changeDetectorRef.detach(); // отсоединяет от change-
detection дерева
        setInterval(() => {
            this.changeDetectorRef.detectChanges(); // помечает
для проверки в отсоединённом локальном дереве
        }, 5000);
    }
```

- detectChanges - используется вместе с detach для локальной обработаки изменений
- https://angular.io/api/core/ChangeDetectorRef#detectchanges

9. Что такое zone.js, как он работает.

- https://medium.com/@overthesanity/zone-js-от-а-до-я-fdb995917968
- https://angular.io/guide/zone#zones-and-execution-contexts
- https://github.com/angular/angular/blob/master/packages/zone.js/MODULE.md
- https://github.com/angular/angular/blob/master/packages/zone.js/README.md
- https://youtu.be/3IqtmUscE_U?t=116
- портирован из Dart
- zone предоставляет контекст исполнения для асинхронных задач
- манкипатчинг и освобождение ресурсов
- профилирование отладка(трассировка) связывания HTML-JS
- заглушки для тестирования Jasmine/Mocha
- принудительный патчинг асинхронных API, не перехватываемых по-умолчанию

```
this.ngZone.run(() => {
    someNewAsyncAPI(() => {
        // update the data of the component
    });
});
```

10. Как запустить код за пределами Angular.

    - https://angular.io/guide/zone#ngzone-run-and-runoutsideofangular

```
    export class AppComponent implements OnInit {
    constructor(private ngZone: NgZone) {}
    ngOnInit() {
        // You know no data will be updated,
        // so you don't want to trigger change detection in this
        // specified operation. Instead, call
ngZone.runOutsideAngular()
        this.ngZone.runOutsideAngular(() => {
        setTimeout(() => {
            // update component data
            // but don't trigger change detection.
        });
        });
    }
    }
```

    - принудительное выключение перехватчиков по типу

```
    // disable patching requestAnimationFrame
    (window as any).__Zone_disable_requestAnimationFrame = true;

    // disable patching specified eventNames
    (window as any).__zone_symbol__UNPATCHED_EVENTS = ['scroll',
'mousemove'];
```

    - принудительное выключение перехватчиков

```
    // import 'zone.js/dist/zone';  // Included with Angular CLI.
    platformBrowserDynamic().bootstrapModule(AppModule, { ngZone:
'noop' })
    .catch(err => console.error(err));
```

11. pipe конвеер

    - `<p>Message: {{ message$ | async }}</p>` - rxjs/promise без подписок
    - `{{ amount | currency:'EUR':'Euros '}}`
    -

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

```
    * Raise the value exponentially
    * Takes an exponent argument that defaults to 1.
    * Usage:
    *   value | exponentialStrength:exponent
    * Example:
    *   {{ 2 | exponentialStrength:10 }}
    *   formats to: 1024
    */
    @Pipe({
        name: 'exponentialStrength',
        pure: true // default
    })
    export class ExponentialStrengthPipe implements PipeTransform {
        transform(value: number, exponent?: number): number {
            return Math.pow(value, isNaN(exponent) ? 1 : exponent);
        }
    }
```

```
    <p>Super power boost: {{2 | exponentialStrength: 10}}</p>
```

12. Сколько может быть router-outlet в компоненте.
    - несколько именованных https://angular.io/api/router/RouterOutlet#description
13. Зачем нужны resolvers.
    - https://angular.io/guide/lazy-loading-ngmodules#preloading-component-data
    - https://angular.io/guide/router-tutorial-toh#preloading-background-loading-of-feature-areas
    - https://codeburst.io/understanding-resolvers-in-angular-736e9db71267
    - https://medium.com/first-byte/resolvers-in-angular-simplified-85becdd6932b
    - блокируют загрузку страницы до окончания отрисовки компонента

```
    // service
    export class CrisisDetailResolverService implements Resolve<> {
        resolve(route: ActivatedRouteSnapshot, state:
RouterStateSnapshot): Observable<> {
            // your logic goes here
        }
    }

    // route
    {
        path: '/your-path',
        component: YourComponent,
        resolve: {
            crisis: YourResolverService
        }
    }
```

```
// component
ngOnInit() {
    this.route.data
        .subscribe((your-parameters) => {
        // your data-specific code goes here
        });
}
```

14. Зачем нужен OnInit если есть Constructor?
    - для работы с начальными @Input значениями
15. Pure pipes / pipes
    - По-умолчанию чистые(pure) - запуск только при изменении объекта целиком. Необходимо передавать чистые функции без побочных эффектов
    - Грязные конвейеры запускаются при каждом нажатии или движении мышки.

```
@Pipe({
    name: 'flyingHeroesImpure',
    pure: false
})
```

16. ng-content
    - точка сборки для вложенных компонентов
    - transclusion или content projection
    - https://angular.io/guide/content-projection
    - Single-slot content projection

```
<ng-content></ng-content>
```

```
<app-zippy-basic>
<p>Is content projection cool?</p>
</app-zippy-basic>
```

    - Multi-slot content projection

```
<!-- Default: -->
<ng-content></ng-content>

<!-- Question: -->
<ng-content select="[question]"></ng-content>
```

```html
    <app-zippy-multislot>
        <p question>
            Is content projection cool?
        </p>
        <p>Let's learn about content projection!</p>
    </app-zippy-multislot>
```

Router

1. Как подгрузить отдельную библиотеку по требованию, не используя роутер?
2. Как загрузить по требованию какую-то часть приложения. Lazy-loading.
    - To lazy load Angular modules, use loadchildren (instead of component) in your AppRoutingModule routes configuration as follows.

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

3. Разница между root и forChild routes
    - forChild создаётся отдельный экземпляр для lazyLoad импортов, нет providers
4. Query params vs matrix params, в чем преимущества и недостатки.
5. Текущий маршрут

```
    constructor(private route: ActivatedRoute){
        this.id = route.snapshot.paramMap.get('id')
        this.id$ = route.paramMap.map(params=>params.get('id'))
    }
```

- router debug

```
    RouterModule.forRoot(
    appRoutes,
    { enableTracing: true } // <-- debugging purposes only
    )
```

Формы

1. NgForm, преимущество реактивных форм над template-driven.
    - удобные валидаторы
    - меньше кода в HTML
2. template vs reactive
    - в шаблонных хуже покрытие линтерами
    - state
        - value: pristine, dirty
        - validity: valid, errors: errorName:any[]
        - visited: touched, untouched
    - для обоих одинаковые модели форм, но они по-разному создаются
    - fromControl
        - `<input>`
        -
        -
    - formGroup
        - `<form>`
        - .reset()
        - .setControl('formControlName', any[])
    - шаблонные
        - в шаблон: form, input, связывание со свойствами, правила валидации, ошибки валидации
        - в класс: свойства, методы
        - модель генерируется автоматом
        - доступ к formGroup возможен через шаблонные переменные
        - директивы: ngForm, ngModel, ngModelGroup
        - не подходят для:
            - отложить валидацию до конца ввода значений
            - анализа вводимого текста на лету
            - динамического добавления полей ввода
            - валидация по условию
            - иммутабельные данные
    - реактивные
        - в шаблон: form, input, связывание с моделью
        - в класс: правила валидации, ошибки валидации, свойства, методы, модель
        - директивы: formGroup, formControl, formControlName, formGroupName, formArrayName
        - доступ:

        - formGroup.controls.controlName
        - formGroup.get('controlName)
    - запись:
        - formGroup.setValues(valuesObj) - полный список полей
        - formGroup.patchValues(valuesObj) - неполный список полей
    - типизированные(14+)

```
interface LoginForm {
    email: FormControl<string>;
    password?: FormControl<string>;
}

const login = new FormGroup<LoginForm>({
    email: new FormControl('', {nonNullable: true}),
    password: new FormControl('', {nonNullable: true}),
});
```

3. валидаторы
    - formControl.clearValidators();
    - formControl.updateValueAndValidity();
    - для валидации нескольких полей можно сделать валидатор для вложенной группы

```
    myVal(c:AbstractControl): {[key:string]: boolean}|null{
        if (c.firstInput.dirty && c.secondInput.dirty &&
c.firstInput.value !== c.secondInput.value) { return
{'notSame':true}}
        return null;
    }
    this.formGroup = this.fb.group({
        g: this.fb.group({
            firstInput: '',
            secondInput: '',
        }, {validator: myVal})
    })
```

    -
    -
4.
    -
    -
    -
    -
    -

5.
6.

Angular Material и SDK.

Сеть

- angular in memory web api https://angular.io/tutorial/toh-pt6#simulate-a-data-server
- interceptors перехватчики https://javascript.plainenglish.io/angular-interceptors-a-complete-guide-7294e2317ecf

```
import { HTTP_INTERCEPTORS } from '@angular/common/http';
// Rest imports...

@NgModule({
declarations: [AppComponent],
imports: [
 // ...
],
providers: [
 //
 /*
 the order of processing for requests is: LogInterceptor--
>CacheInterceptor-->AuthInterceptor-->MockInterceptor, while the order of
processing for responses is the opposite: MockInterceptor--
>AuthInterceptor-->CacheInterceptor-->LogInterceptor
 */
 { provide: HTTP_INTERCEPTORS, useClass: LogInterceptor, multi: true },
 { provide: HTTP_INTERCEPTORS, useClass: CacheInterceptor, multi: true
},
 { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true },
 { provide: HTTP_INTERCEPTORS, useClass: MockInterceptor, multi: true },
],
bootstrap: [AppComponent],
})
export class AppModule {}
```

-

RxJS

- public/kbo/kb/frontend/angular/rxjs.md

NGRX

- public/kbo/kb/frontend/angular/ngrx.md

angular CSS

- shadow dom
- https://angular.io/guide/component-styles
- Как работает ViewEncapsulation и ng-deep в Angular
- селекторы со скобками - функциональная форма
- псевдо-класс - специальное состояние элемента
  - `:host(tag)|:host` - добавляет генерируемый префикс к стилям
  - `:root` - синоним `<html>`
  - `:default, :checked, :nth-of-type, :hover, :active`
- псевдо-элементы - выбор элемента
  - `::before, ::after`
  - `::ng-deep` - отключает инкапсуляцию
    - общее правило - добавлять :host или связывать с селектором компонента для предотвращения расползания охвата `:host ::ng-deep h3 {`
  - `::part()` - дополнительный идентификатор

```
//<div part="tab">Tab 3</div>
tabbed-custom-element::part(tab):focus {
}
```

- css variables, css custom properties

```
<button *ngFor="let item of items; index as index"
        [style.--animation-delay]="index * 0.2 + 's'">
{{ item }}
</button>
```

```
:button {
    ...
```

```
            animation-delay: var(--animation-delay);
        }
```