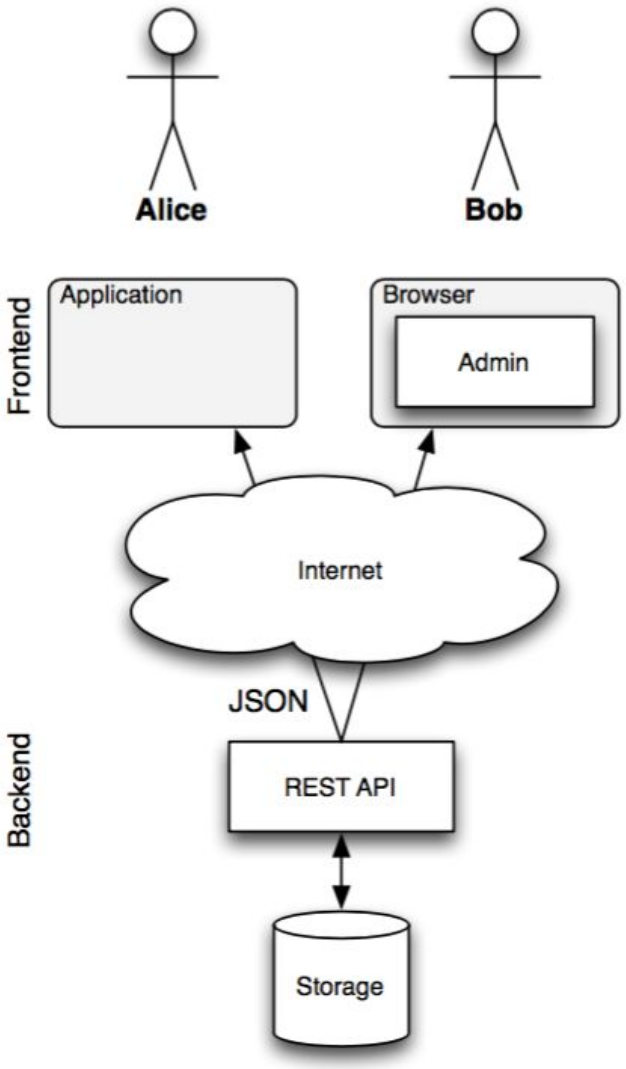
Front-end на базе Angular

Цопа Е.А.

2018/19 уч. год

ИТМО ВТ

Современные веб-приложения



Плюсы подхода:

- Можно выбирать любые технологии для front-end'а и back-end'а.
- Удобно добавлять альтернативные интерфейсы (например, Android App).
- Удобно "нарезать" на микросервисы.

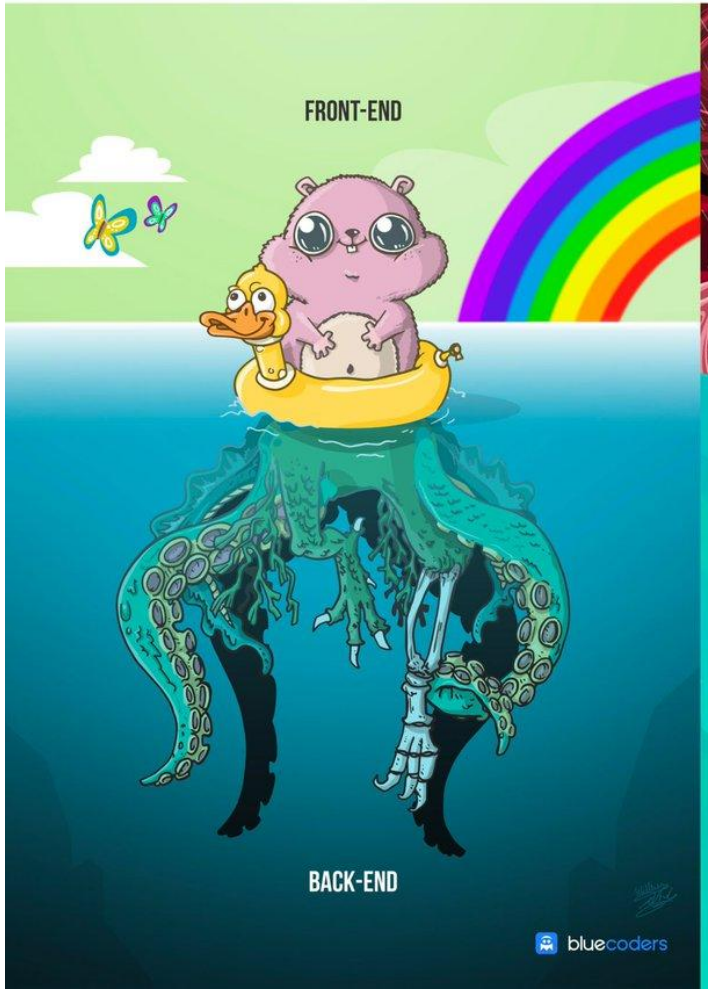
Минусы подхода:

- Логику синхронизации между front-end и back-end реализует программист.
- Необходимость управления состоянием на стороне клиента.

ИТМО ВТ

Front-end и Back-end

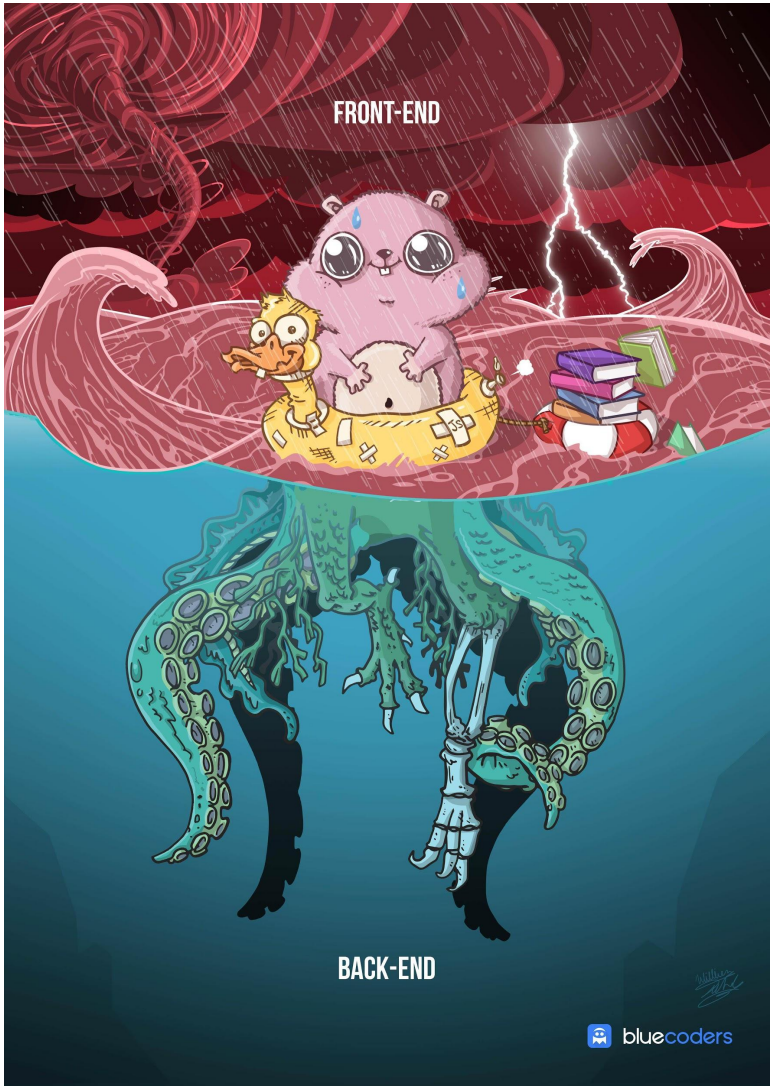
С точки зрения Front-end разработчика



С точки зрения Back-end разработчика

ИТМО ВТ

...на самом деле



ИТМО ВТ

Angular и AngularJS

(Две большие разницы).

Angular -- открытая и свободная платформа от Google для разработки веб-приложений.

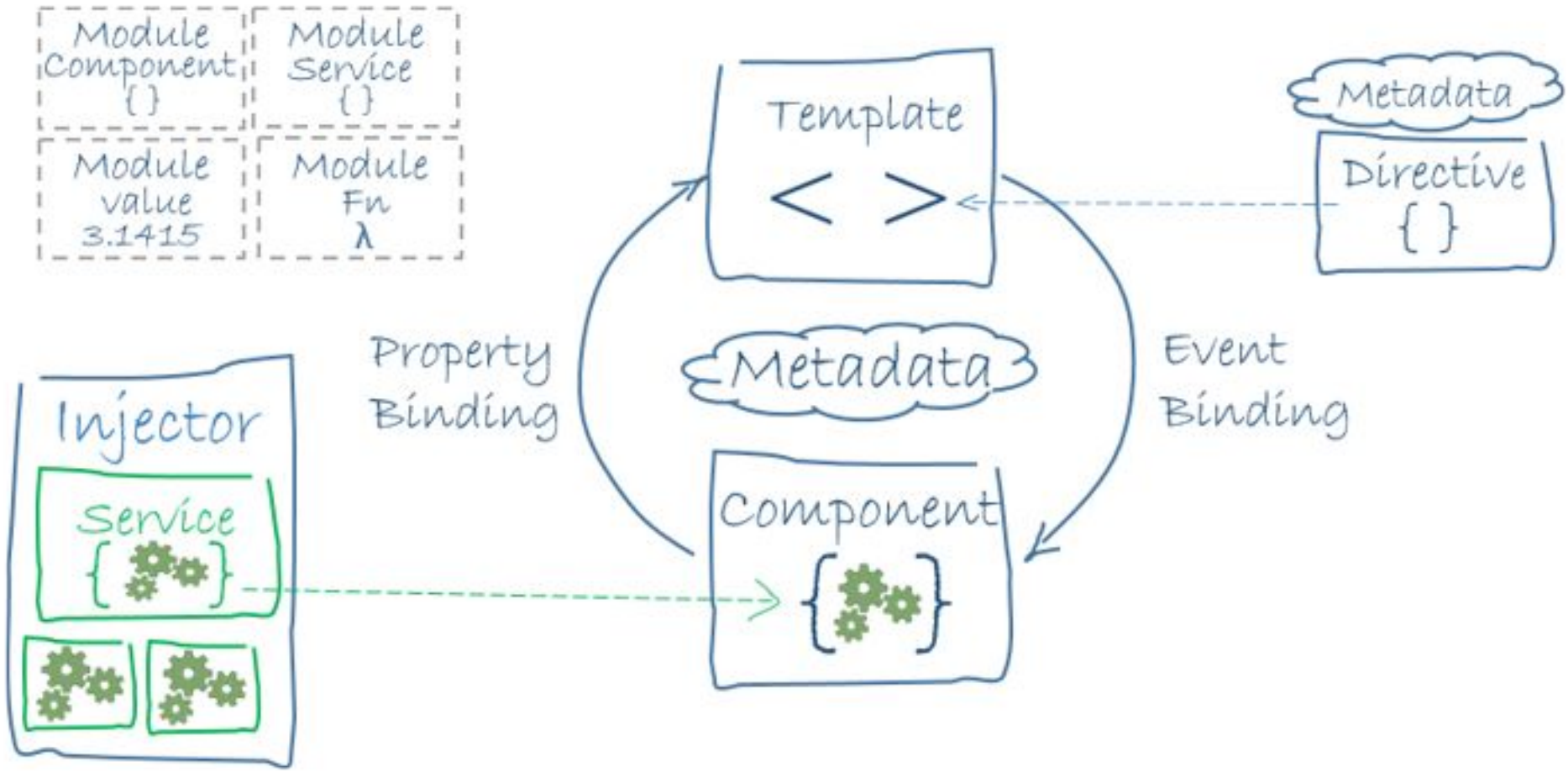
- Написана на языке TypeScript.
- Первый релиз (2.0) -- сентябрь 2016 г. Актуальная версия -- 7.0 (октябрь 2018 г.)
- Является развитием проекта AngularJS от той же команды (разрабатывается с 2009 г.)

Особенности Angular

- Для разработки нужно настроить сборочное окружение (на базе node.js и npm).
- Приложения состоят из *модулей (NgModules)*.
- Модули обеспечивают контекст для *компонентов (components)*.
- Из компонентов строятся *представления (views)*.
- Компоненты взаимодействуют с *сервисами (services)* с помощью DI.

ИТМО ВТ

Архитектура приложения на базе Angular



Модули (NgModules)

- Не совсем то же самое, что модули в ES6 (хотя и похожи).
- Каждый модуль обеспечивает *контекст компиляции* для одного или группы компонентов.
- Могут связывать компоненты с нужными для их работы сервисами.
- Группировка компонентов по модулям -- на усмотрение программиста.
- 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).
- Могут использоваться для реализации *загрузки по требованию* (*lazy loading*).

Структура модуля

- Каждый модуль -- класс TS с декоратором `@NgModule()`.
- 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).

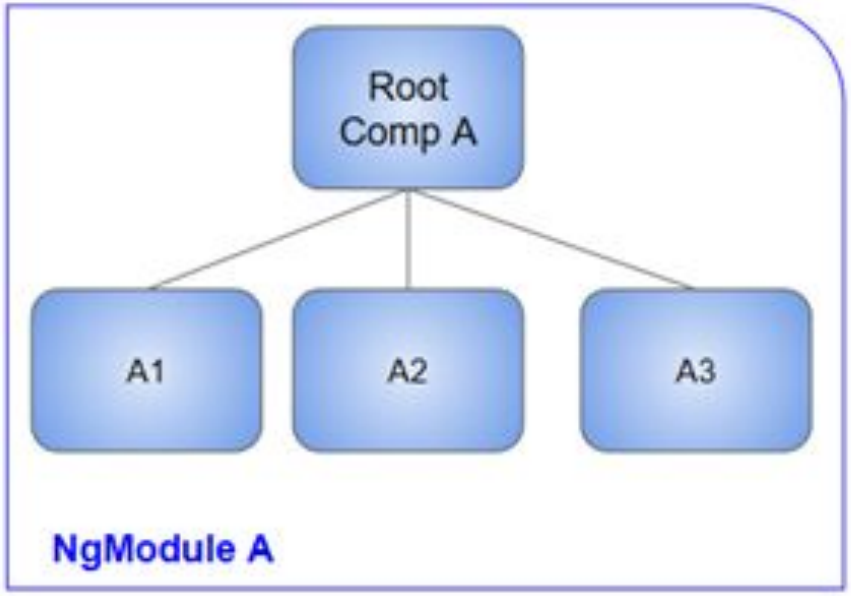
Пример модуля

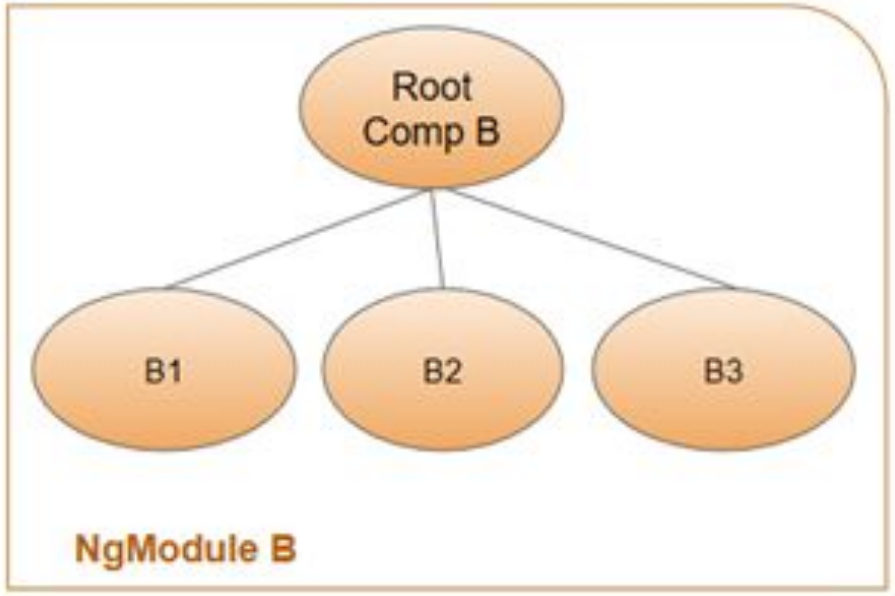
src/app/app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Модули и компоненты

У всех компонентов внутри модуля общий *контекст компиляции (compilation context)*.

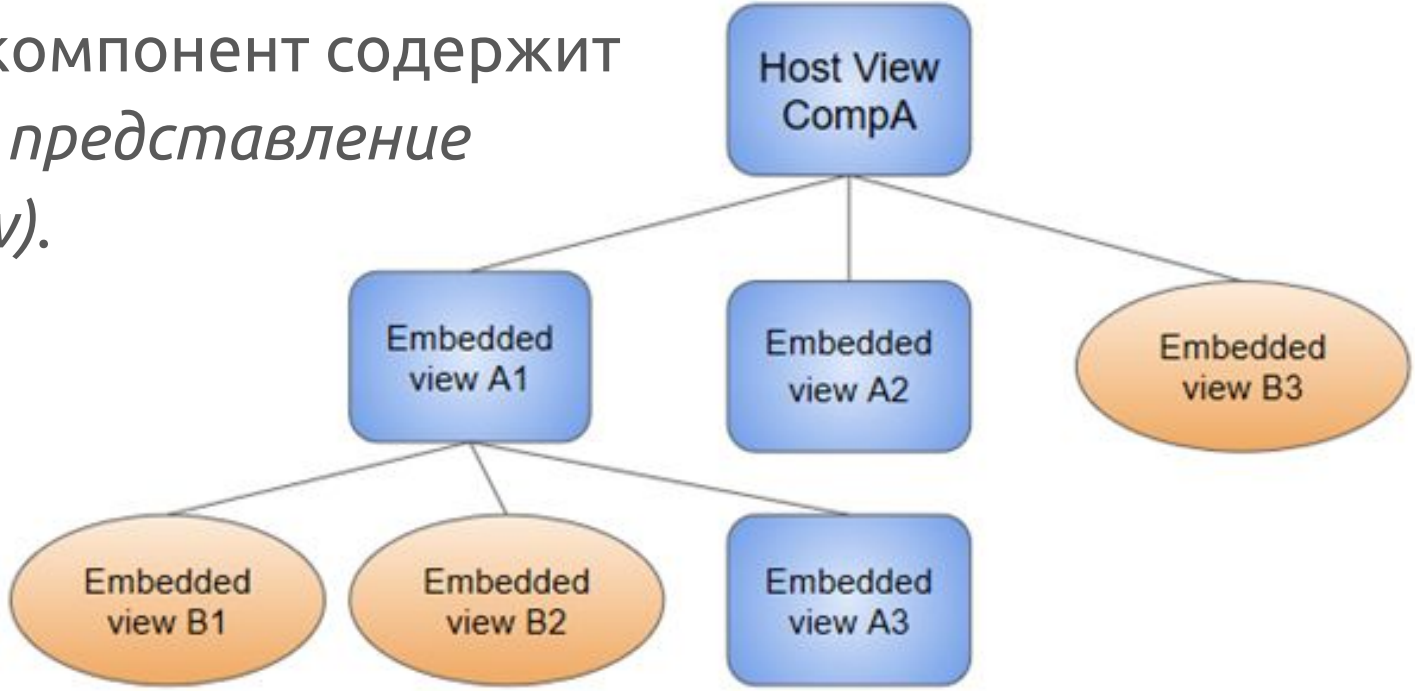




ИТМО ВТ

Представления (views)

- Компоненты и их шаблоны формируют *представления (views).*
- Компонент может содержать *иерархию представлений (view hierarchy).*
- Каждый компонент содержит *корневое представление (host view).*



ИТМО ВТ

Компоненты

- Каждый компонент -- отдельный класс.
- Контролирует область экрана, называемую *представлением (view)*.
- Angular управляет жизненным циклом компонентов.

src/app/hero-list.component.ts (class)

```
export class HeroListComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor(private service: HeroService) { }

  ngOnInit() {
    this.heroes = this.service.getHeroes();
  }

  selectHero(hero: Hero) { this.selectedHero = hero; }
}
```

ИТМО ВТ

Метаданные компонента

- Задаются с помощью декоратора `@Component`.
- Сообщают рантайму о том, что это за компонент, и где искать его составляющие.

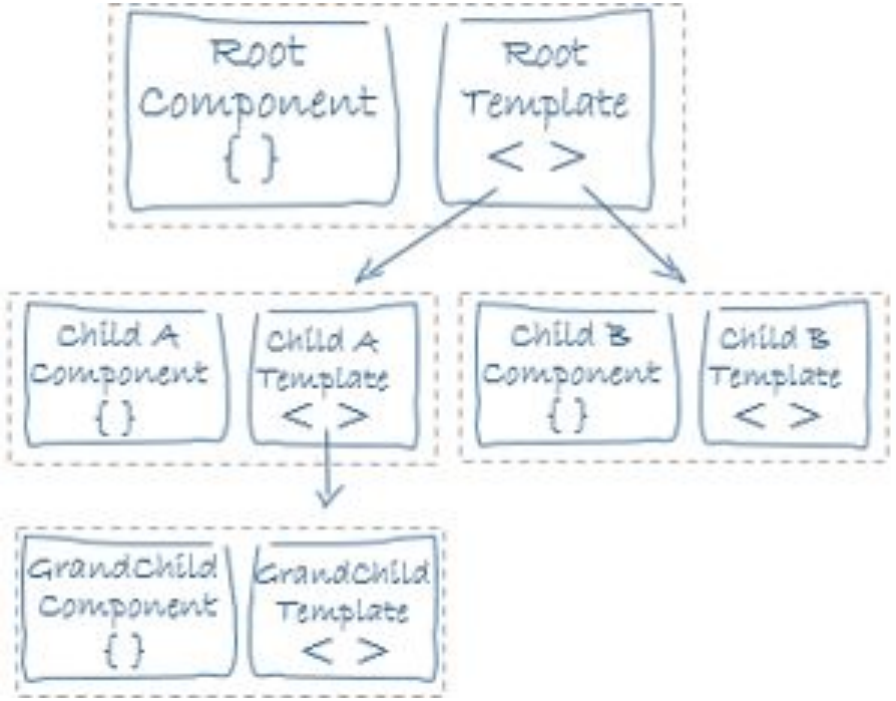
src/app/hero-list.component.ts (metadata)

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

ИТМО ВТ

Шаблоны и представления

- *Представление (view)* компонента задаётся с помощью *шаблона (template)*.
- Представления часто группируются иерархически.
- Компонент может содержать *иерархию представлений (view hierarchy)*, которая содержит *встроенные представления (embedded views)* из других компонентов.



ИТМО ВТ

Синтаксис шаблонов

- Похож на обычный HTML.
- Взаимодействие с классом компонента осуществляется с помощью ссылок на его свойства *(data binding)*.
- Также можно использовать *фильтры (pipes)* и *директивы (directives)*.

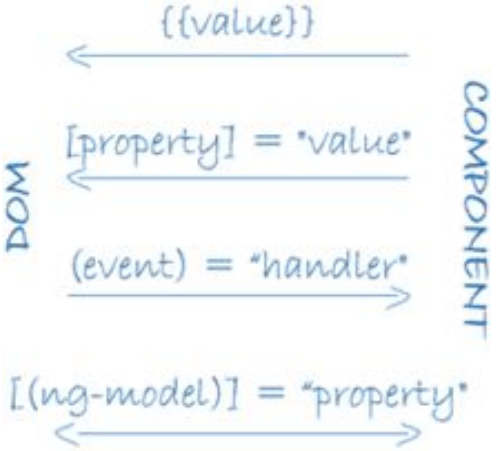
src/app/hero-list.component.html

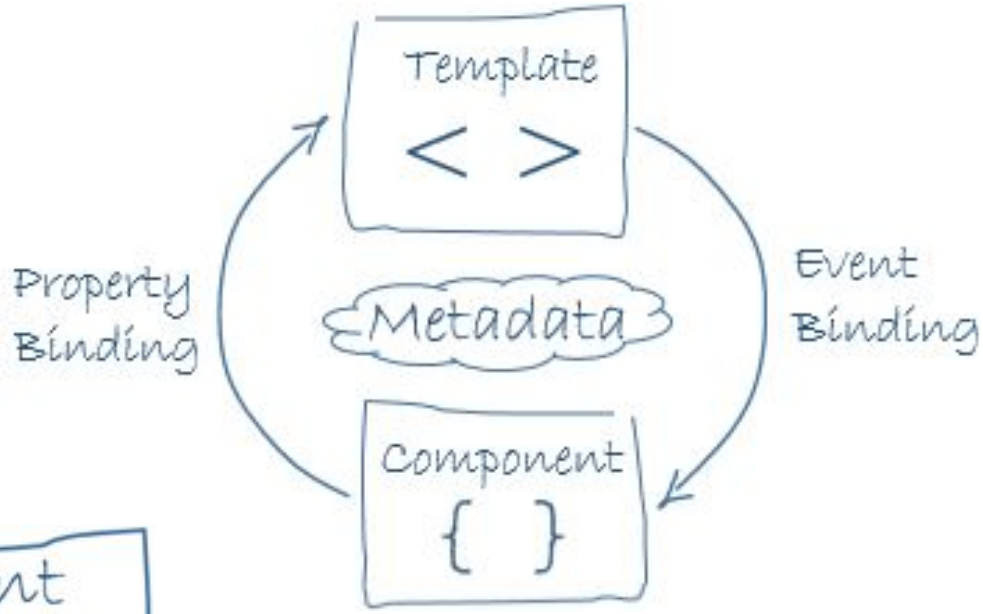
```
<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>
<ul>
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-
hero-detail>
```

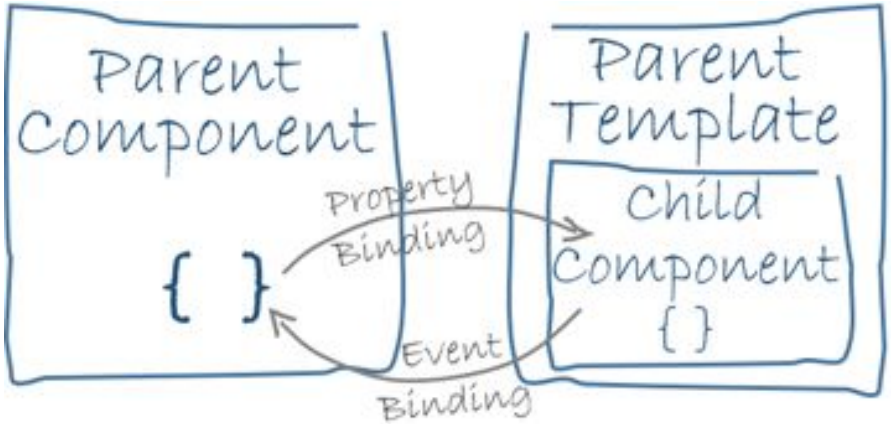
ИТМО ВТ

Двунаправленное связывание (Two-way Data Binding)

Связь между шаблоном и свойствами компонента -- двунаправленная.







ИТМО ВТ

Три вида связей

src/app/hero-list.component.html (binding)

```
<li>{{hero.name}}</li>
<app-hero-detail [hero]="selectedHero"></app-hero-detail>
<li (click)="selectHero(hero)"></li>
```

- `{{hero.name}}` -- *отображение (interpolation)*. Показывает значение свойства в HTML-разметке.
- `[hero]` -- *связывание свойства (property binding)*. Передаёт значение свойства `selectedHero` родительского компонента `HeroListComponent` в качестве свойства `hero` дочернего компонента `HeroDetailComponent`.
- `(click)` -- *связывание по событию (event binding)*. Вызывает метод, когда пользователь кликает по элементу.

ИТМО ВТ

Фильтры (pipes)

- 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.

```
<!-- Default format: output 'Jun 15, 2015'-->
 <p>Today is {{today | date}}</p>

<!-- fullDate format: output 'Monday, June 15, 2015'-->
<p>The date is {{today | date:'fullDate'}}</p>

 <!-- shortTime format: output '9:43 AM'-->
 <p>The time is {{today | date:'shortTime'}}</p>
```

ИТМО ВТ

Директивы (directives)

- Инструкции по преобразованию DOM.
- Создаются с помощью декоратора `@Directive()`.
- Все компоненты, технически -- директивы.
- Два вида:
  - Структурные директивы (structural directives):

src/app/hero-list.component.html (structural)

```
<li *ngFor="let hero of heroes"></li>
<app-hero-detail *ngIf="selectedHero"></app-hero-detail>
```

  - Директивы-атрибуты (attribute directives):

src/app/hero-detail.component.html (ngModel)

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

Сервисы и DI

- *Сервисы (services)* реализуют какие-либо действия, не формируя представление.
- 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.

ИТМО ВТ

Примеры сервисов

src/app/logger.service.ts (class)

```
export class Logger {
  log(msg: any)   { console.log(msg); }
  error(msg: any) { console.error(msg); }
  warn(msg: any)  { console.warn(msg); }
}
```

src/app/hero.service.ts (class)

```
export class HeroService {
  private heroes: Hero[] = [];

  constructor(
    private backend: BackendService,
    private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {
      this.logger.log(`Fetched ${heroes.length} heroes.`);
      this.heroes.push(...heroes); // fill cache
    });
    return this.heroes;
  }
}
```

ИТМО ВТ

Dependency Injection

- Компоненты могут использовать сервисы с помощью DI.

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

ИТМО ВТ

Основные принципы реализации DI

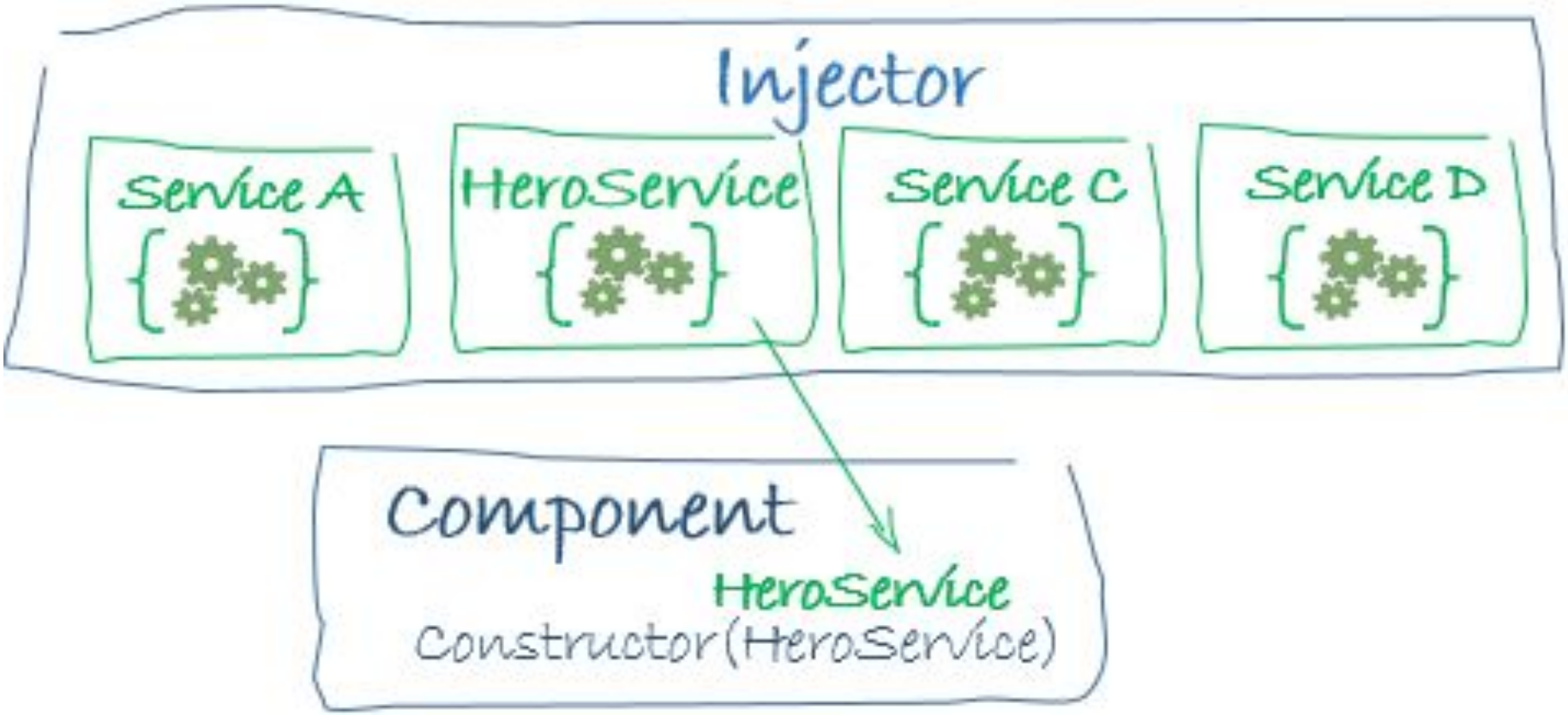
- Приложение содержит как минимум один глобальный *Injector*, который занимается DI.
- Injector создаёт зависимости и передаёт их экземпляры *контейнеру (container)*.
- *Провайдер (provider)* -- это объект, который сообщает Injector'у, как получить или создать экземпляр зависимости.
- Обычно провайдером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

src/app/hero-list.component.ts (constructor)

```
constructor(private service: HeroService) { }
```

ИТМО ВТ

Injector



ИТМО ВТ

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 ---------->

```
@NgModule({
  providers: [
  BackendService,
  Logger
 ],
 ...
})
```

  - в метаданных компонента -------->

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

ИТМО ВТ

Взаимодействие с back-end'ом

- Реализуется с помощью сервиса `HttpClient`.
- Для его использования необходимо импортировать модуль `HttpClientModule` в свой `AppModule`:

app/app.module.ts (excerpt)

```
import { NgModule }         from '@angular/core';
import { BrowserModule }    from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';

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

ИТМО ВТ

Взаимодействие с back-end'ом

- Реализуется с помощью сервиса `HttpClient`.
- Для его использования необходимо импортировать модуль `HttpClientModule` в свой `AppModule`:

app/app.module.ts (excerpt)

```
import { NgModule }         from '@angular/core';
import { BrowserModule }    from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';

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

ИТМО ВТ

Использование `HttpClient`

- После импорта модуля можно заинжектить `HttpClient`:

app/config/config.service.ts (excerpt)

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable()
export class ConfigService {
  constructor(private http: HttpClient) { }
}
```

ИТМО ВТ

Пример получения JSON с сервера

- Файл на сервере:

assets/config.json

```
{
  "heroesUrl": "api/heroes",
  "textfile": "assets/textfile.txt"
}
```

- Сервис:

app/config/config.service.ts (getConfig v.1)

```
configUrl = 'assets/config.json';

getConfig() {
  return this.http.get(this.configUrl);
}
```

ИТМО ВТ

Пример получения JSON с сервера (продолжение)

- Компонент:

app/config/config.component.ts (showConfig v.1)

```
showConfig() {
  this.configService.getConfig()
    .subscribe((data: Config) => this.config = {
        heroesUrl: data['heroesUrl'],
        textfile:  data['textfile']
    });
}
```