Микрофронтенд

BIV Group

0.0.1,

Содержание

1 Определения и сокращения

Сокращение	Наименование
Microfrontend	Микрофронтенд
Web components	Веб компоненты
single spa	Библиотека single spa
JWT	JWT
Angular	Angular JS Framework
Angular Elements	Расширение для создание веб компонентов на Angular
Angular CLI	Утилита создания приложений Angular
SPA	Single Page Application

2 Введение

Микрофронтед является прямым продолжением идеи микросервисной архитектуры для фронтенд-разработки. Таким образом, по аналогии с микросервисом, микрофронтенд - это небольшая, логически отделенная часть приложения, а точнее, в контексте разработки frontend, часть пользовательского интерфейса вашего приложения. Часть, которая в первую очередь не зависит от остальных частей системы с точки зрения ее развертывания. Кроме того, Микрофронтед должен инкапсулировать некоторую часть бизнес-логики приложения и должен принадлежать исключительно команде, которая отвечает за эту часть приложения от начала до конца.

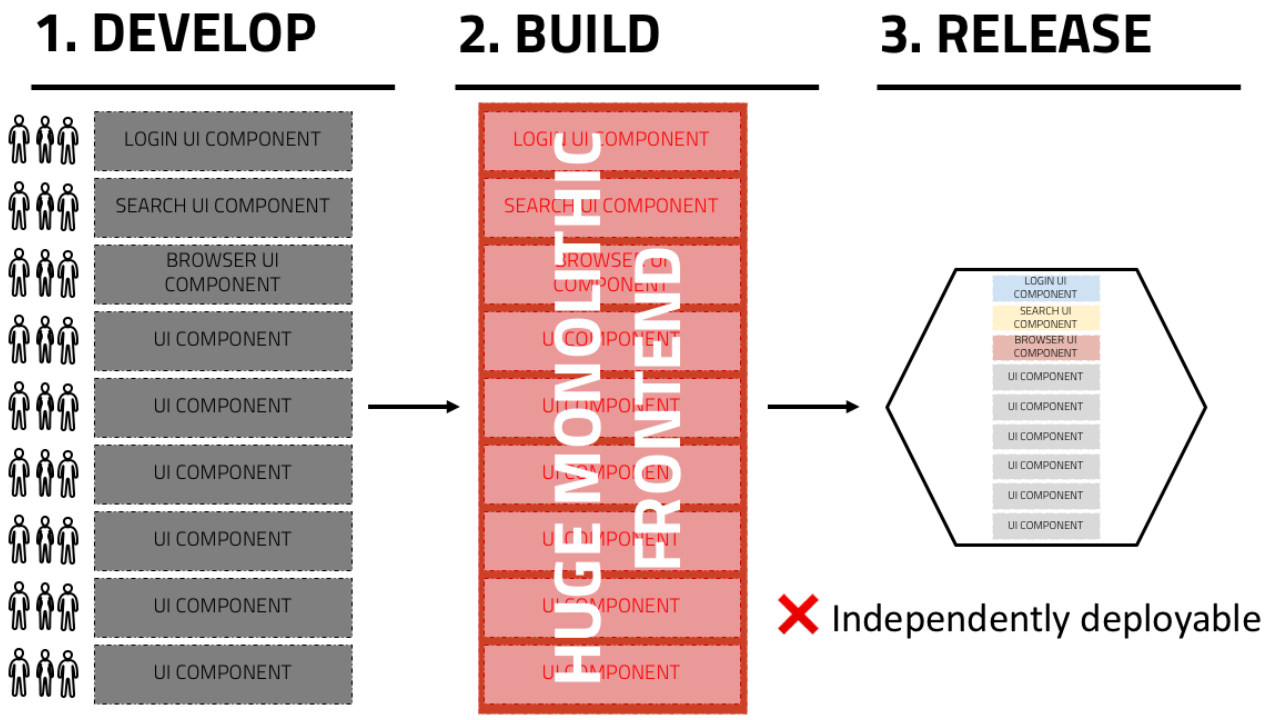
Приложение, созданное с помощью микрофронтенд подхода для конечного пользователю может выглядеть как обычный сайт, в то время как под капотом он может быть построен в виде композиции отдельных микроприложений. Каждое из этих приложений имеет свою собственную базу кода и репозиторий кода, свой собственный жизненный цикл разработки, свою собственную версию и свою собственную команду, ответственную за это приложение от начала до конца.

2.1 Проблематика

Текущие SPA приложения обладают очень большой функциональностью.

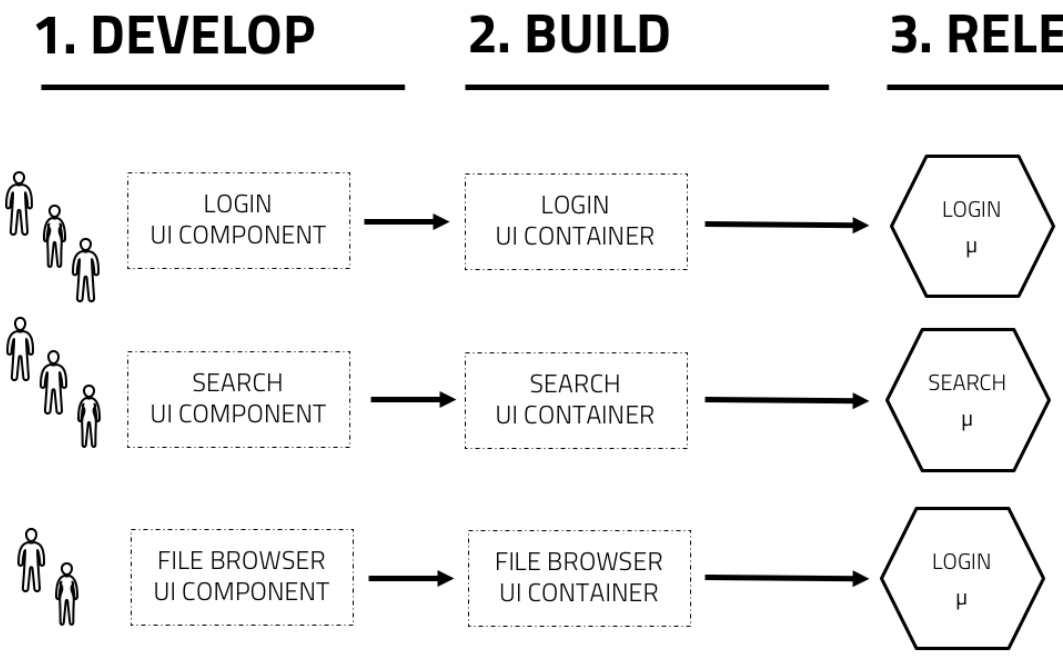
Разработка приводит к созданию большого количества небольших компонентов, которые затем объединяются вместе в итоговый код приложения (js bundle или несколько js bundle).

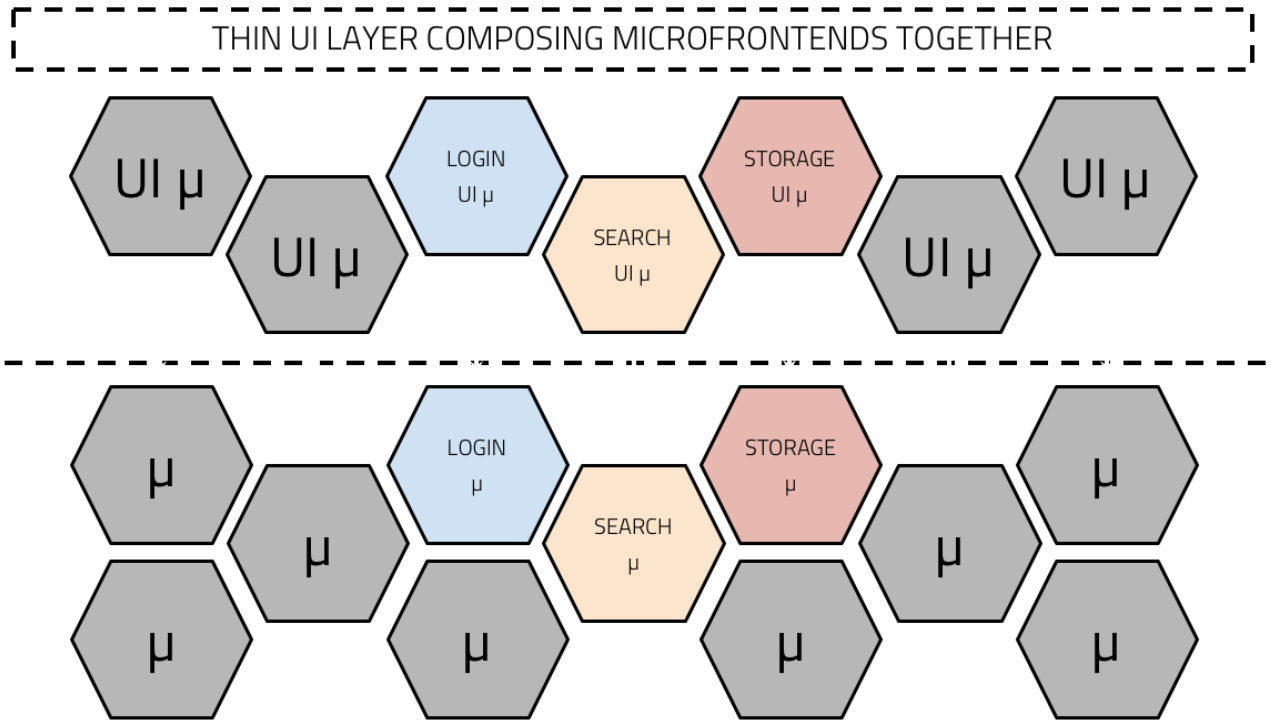
Со временем приложение, часто разрабатываемое несколькими командами, растет и его становится все труднее поддерживать. Как результат получается монолит.



3 Архитектура

Микрофронтенд предлагает подход избежать получения монолита и приводит к созданию слабосвязанных микроприложений, которые могут иметь различные жизненные циклы разработки.





Микрофронтенд состоит из элементов:

1. Слой управления микроприложениями
2. Микроприложения

3. Веб компоненты

4 Микроприложение

Как уже было упомянуто выше, микрофронтендный подход позволяет каждому микроприложению иметь свою собственную базу кода, это означает, что библиотека single spa позволяет:

- Использовать несколько фреймворков на одной странице без обновления страницы (React, AngularJS, Angular, Ember или все, что вы используете)
- Разворачивать свои микроприложения самостоятельно.
- Написать код, используя новый фреймворк, не переписывая существующее приложение.

4.1 Single-spa микроприложения

Single-spa микроприложения состоят из следующего:

1. Микроприложения, каждое из которых представляет собой целый SPA (своего рода). Каждое микроприложение может отвечать на события маршрутизации URL и должно знать, как загружать, монтировать и размонтировать себя из DOM. Основное различие между традиционными приложениями SPA и микроприложениями состоит в том, что они должны иметь возможность сосуществовать с другими приложениями, и у каждого из них нет собственной HTML-страницы.
2. Single-spa-config, представляет собой html-страницу и JavaScript, который регистрирует приложения в single-spa. Каждое приложение зарегистрировано тремя правилами:
    - Наименование
    - Функция, которая загружает код микроприложения
    - Функция, которая определяет, когда приложение активно / неактивно

4.2 Пример развертывания single-spa микроприложения

Процесс сборки микроприложения на фреймворке Angular:

Если у вас нет angular приложения, то вам необходимо создать каталог, открыть терминал и прописать:

```
ng new my-app --routing --prefix my-app
cd my-app
```

Теперь вы имеете базовое приложение от фреймворка Angular, далее будет обновление просто приложения в single-spa микрпоприложение.

- Установка

Для того чтобы ваше фреймворк приложение обернулось в single-spa микроприложение, необходимо в корневом каталоге вашего проекта в консоли прописать:

```
ng add single-spa-angular
```

важно! после выполнения данной команды будет предложено 2 пункта:

1. "**? Does your application use Angular routing**?" - необходимо согласиться, выбрать "**Yes**". Angular routing необходим для маршрутизации между микроприложениями.
2. "**? Does your application use BrowserAnimationsModule?**" - необходимо отказаться, ввести "**No**". BrowserAnimationsModule по неизвестным причинам, мешает работе загрузки, монтирования и размонтирования.

- Схема приложения

После того как вы провели установку single-spa, схема вашего приложения изменилась, а именно:

1. Сгенерировалось "**main.single-spa.ts**" в "**src/**"

   Исходный код в `main.single-spa.ts`, необходим для того чтобы обычное spa приложение, проинициализировала в себя жизненный цикл, маршрутизацию и наименование для дальнейшей работы с микроприложениями и корневой конфигурацией всех микроприложений.

2. Сгенерировалось "**single-spa-props.ts**" в "**src/single-spa/**"

   В `main.single-spa.ts`, вы можете прописать свои собственные пропсы, для общения между микроприложением и корневой конфигурацией.

3. Сгенерировалось "**asset-url.ts**" в "**src/single-spa/**"

   Single-spa assets имеет свою особенность, она позволяет загружать статические файлы так, чтобы они работали в обоих направлениях.

4. Сгенерировалось "**empty-route.component.ts**" в "**src/app/empty-route/**"

   Данный файл необходим для настроек маршрутизации в `app-routing.module.ts`.

5. Добавилась настройка запуска приложения "**serve:single-spa": "ng serve --disable-host-check --port 4200 --deploy-url** http://localhost:4200/ **--live-reload false**" в "**package.json**"

```
Данные настройки необходимы для указания порта и пути по умолчанию для статических файлов.
```

6. Добавилась настройка сборки приложения "**build:single-spa": "ng build --prod --deploy -url http://localhost:4200/**," в "**package.json**"

```
Процесс сборки, выполняет те же функции, что и в пункте 5.
```

Далее необходимо настроить файлы по маршутизации

- Марщутизация

Для того чтобы мы смогли перемещаться между микроприложениями нам необходимо:

1. Добавить в файле "**app-routing.module.ts**":

```
providers: [{ provide: APP_BASE_HREF, useValue: '/' }]
```

```
Больше информации о APP_BASE_HREF, находится в - https://angular.io/api/common/APP_BASE_HREF.
```

1. Добавить в файле "**app-routing.module.ts**":

```
const routes: Routes = [ { path: '**', component: EmptyRouteComponent }];
```

EmptyRouteComponent является частью single-spa-angular схемы. Этот route удостоверяет нас о том что single-spa переходы по маршрутам не вернет 404 или другие ошибки. Больше информации об маршутизации вы можете прочесть тут - https://angular.io/guide/router#configuration

Пример перехода по маршруту:

```
<a routerLink="/other-app">
  Путь к другому микроприложению
</a>
```

4.3 Пример создания и настройки single-spa config

Перед тем как начать создавать корневой файл конфигурации single-spa микроприложений, вы должны иметь несколько микроприложений (сделанных по примеру из пункта 4.2, с разными портами и наименованиями).

- Создание каталога для single-spa config

создайте каталог (назовите его - "root"), создайте в нем файлы:

1. "**package.json**", внутри этого файла нужно прописать настройки и зависимости:

```
{
  "name": "root",
  "description": "The single-spa root config for angular-microfrontends",
  "main": "index.js",
  "scripts": {
    "start": "serve -s -l 4200"
  },
  "author": "Test",
  "license": "MIT",
  "devDependencies": {
    "jspm": "^0.16.55",
    "serve": "^11.1.0"
  },
  "dependencies": {
    "sofe": "npm:sofe@^3.0.0"
  }
}
```

```
далее открыть терминал в данном каталоге и прописать: npm install
```

2. "**index.html**", внутри этого файла для начала нужно иметь стандартную верстку (<html>, <head>, <body> и т.д.).

```
2.1 Далее в теге "*<head>*" нужно импортировать single-spa библиотеку и
микроприложения:
```

```
<script type="systemjs-importmap">
    {
    "imports": {
        "app1": "http://localhost:4201/main.js",
        "app2": "http://localhost:4202/main.js",
        "main-page": "http://localhost:4203/main.js",
        "single-spa": "https://cdnjs.cloudflare.com/ajax/libs/single-
spa/4.3.5/system/single-spa.min.js"
    }
    }
</script>
```

где "app1", "app2" и "main-page" - являются микроприложениями, где их значения это путь с их личным портом.

После чего нужно в этом же теге "*<head>*" импортировать зависимости:

```
<link rel="preload" href="https://cdnjs.cloudflare.com/ajax/libs/single-spa/4.3.5/system/single-spa.min.js" as="script" crossorigin="anonymous" />
<script src='https://unpkg.com/core-js-bundle@3.1.4/minified.js'></script>
<script src="https://unpkg.com/zone.js"></script>
<script src="https://unpkg.com/import-map-overrides@1.6.0/dist/import-map-overrides.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/4.0.0/system.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/4.0.0/extras/amd.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/4.0.0/extras/named-exports.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/4.0.0/extras/named-register.min.js"></script>
```

2.2 Последним этом нужно в теге "*<body>*" зарегистрировать микроприложения через библиотеку single-spa:

```
<script>
System.import('single-spa').then(function (singleSpa) {
    singleSpa.registerApplication(
        'main-page',
        function () {
            return System.import('main-page');
        },
        function (location) {
            return true;
        }
    );
    singleSpa.unloadApplication('main-page', {waitForUnmount: true});
    singleSpa.registerApplication(
        'app1',
        function () {
            return System.import('app1');
        },
        function (location) {
            return true;
        }
    );
    singleSpa.unloadApplication('app1', {waitForUnmount: true});
    singleSpa.registerApplication(
        'app2',
        function () {
            return System.import('auth');
        },
        function (location) {
            return location.pathname.startsWith('/app2');
        }
    );
    singleSpa.unloadApplication('app2', {waitForUnmount: true});
    singleSpa.start();
})
</script>
```

Как было описано в заголовке 4.1 в пунтке 2, single-spa регестрирует микроприложение, а именно его наименование, его исходный код и путь по которому пользователь сможет попать в данное микроприложение.

Если путь указан так:

```
function (location) {
    return location.pathname.startsWith('/app2');
}
```

то по переходу по ссылке "http://localhost:4200/app2", отобразится микроприложение "app2".

Если же путь указан так:

```
function (location) {
    return true
}
```

то микроприложение всегда будет активно в рамке страницы. Например если у микроприложения "main-page", путь будет указан "return true", то перейдя к любому микроприложению, микроприложение "main-page", не будет размонтирован (то в одной странице будут два микроприложения)

2.3 Далее вам необходимо открыть терминал в каталоге "/root" и прописать:

```
npm start
```

После чего вы можете перейти по ссылке "http://localhost:4200" и увидеть результат.

5 Веб компонент

Веб компоненты – набор стандартных API, позволяющих нативно создавать кастомные HTML теги со своей функциональностью и жизненным циклом компонентов. API включают в себя спецификацию по кастомным элементам, теневой DOM, который позволяет изолировать внутренности компонента, а также импорты HTML, где описывается, как необходимо загружать компоненты, и как их использовать в веб-приложении. Основная задача веб-компонентов – инкапсуляция кода компонентов в хорошие, повторно используемые пакеты для достижения максимальной совместимости.

Angular 2+ делает много тяжелой работы за вас, компилируя шаблон компонента в рендерер JS и синхронизируя данные между ним и объектом компонента. Делается это через однонаправленный поток данных и менеджер жизненных циклов, который определяет изменения в свойствах компонента и указывает, какой шаблон необходимо заново отрендерить. Язык шаблонов Angular – это в основном HTML с синтаксическим сахаром, поэтому компилятор уже знает, как создавать узлы DOM по тегам в шаблоне. По умолчанию компилятор понимает только стандартные теги HTML и компоненты Angular, зарегистрированные в приложении. Как только компилятор Angular узнает о веб-компонентах, в приложении сразу можно использовать любой веб-компонент, зарегистрированный в браузере, как родной HTML элемент.

5.1 Пример создания веб компонента

Веб компонент можно завернуть в практически любой фреймворк (React, Vue, Angular и т.д.)

Процесс создания веб компонента на примере фреймворка Angular:

Создаем новый ангуляр проект, открываем терминал и прописываем:

```
ng new webcomp
```

Когда терминал попросит вас:

```
 - Would you like to add Angular routing?
```

Необходимо отказаться - выбираем "**No**"

Далее необходимо добавить в проект библиотеки "**@angular/elements**" и "**ngx-build-plus**":

```
ng add @angular/elements
ng add ngx-build-plus
```

Чтобы убедиться, что установка прошла успешно, перейдите к файлу "**angular.json**" и проверьте свойство **«builder»** в "**Architect → build**", должно быть **«builder»**: **«ngx-build-plus: browser»**.

Давайте прокомментируем свойство "**bootstrap**" в объявлении "**@NgModule**", добавим "**AppComponent**" к "**entryComponents**" и добавим пользовательский элемент в "**webcomp** / **src** / **app** / **app.module.ts**":

import { BrowserModule } from '@angular/platform-browser'; import {DoBootstrap, Injector, NgModule} from '@angular/core';

import { AppComponent } from './app.component'; import {createCustomElement} from '@angular/elements';

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  // bootstrap: [AppComponent],
  entryComponents: [
    AppComponent
  ]
})
export class AppModule implements DoBootstrap {
  constructor(private injector: Injector) {
  }
  ngDoBootstrap() {
    const el = createCustomElement(AppComponent, { injector: this.injector });
    customElements.define('web-comp', el);
  }
}
```

Чтобы запустить проект в режиме разработки ("**ng serve**"), вы должны раскомментировать свойство "**bootstrap**" в объявлении "**@NgModule**" в "**webcomp** / **src** / **app** / **app.module.ts**".

В файле "**webcomp** / **src** / **app** / **app.component.html**" удалите все содержимое, и например вставьте туда кнопку:

```
<button>i'm button webcomp</button>
```

Далее необходимо сгенерировать проект:

```
ng build --prod --output-hashing none --single-bundle true
```

5.2 Пример использования веб компонента

- Развертывание сервера

Необходимо создать сервер, для расположения нашего веб компонента.

Создайте новый католг и файл "**package.json**", в этом файле пропишите:

```
{
  "name": "external",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "express-cache-controller": "^1.1.0",
    "memory-cache": "^0.2.0"
  }
}
```

Далее откройте терминал и установите все зависимости:

```
npm install
```

После чего необходимо создать файл "**server.js**" и поднять сервер, который будет выдавать веб компонент:

```
const express = require('express');
const server = express();
server.get('/webcomp', (req,res) => {
    res.set('Cache-Control', 'public, max-age=31557600');
    res.sendFile('/main-es2015.js', { root : __dirname})
});
server.listen(3000, () => console.log('server started!'));
```

Из проекта веб компонента необходимо достать основной js файл. Перейдите в католог проекта, далее в папку "**dist**", скопируйте файл "**main-es2015.js**" и вставьте в католг где находится сервер.

Далее необходимо запустить сервер, откройте терминал в катологе с сервером, пропишите:

```
node server
```

Теперь веб компонент будет находится на пути - "**http://localhost:3000/webcomp**"

- Интеграция веб компонента в микроприложение

В вашем микроприложении необходимо установить библиотеку "**@angular-extensions/elements**" ("**npm install --save @angular-extensions/elements**"), и активировать веб-компоненты в "**src/app/app.module.ts**":

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';
import { LazyElementsModule } from '@angular-extensions/elements';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
     AppComponent
  ],
  imports: [
     BrowserModule,
     LazyElementsModule
  ],
  providers: [],
  bootstrap: [AppComponent],
  schemas: [
 CUSTOM_ELEMENTS_SCHEMA
  ]
})
export class AppModule {}
```

Импортировали "**CUSTOM_ELEMENTS_SCHEMA**" из "**@angular/core**" и добавили ее обьявление "**@NgModule**" приложения в свойстве "**schemas**". Так мы разрешаем компилятору шаблонов Angular веб-компоненты и их атрибуты. Помимо этого мы импортировали билиотеку "**@angular-extensions/elements**", для того чтобы веб компонент подгружался лениво (позже остальных DOM елементов).

Далее обьявляем ссылку на наш веб компонент в файле "**app.component.ts**":

```
import { Component, Inject } from '@angular/core';
@Component({
  selector: 'auth-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent{
  // Ссылка на веб компонент
  linkWebcomp = "http://localhost:3000/webcomp"
}
```

Последним этапом является использование компонента в шаблоне, откройте файл "**app.component.html**" и обьявите тэг веб компонента:

```
<web-comp *axLazyElement="linkWebcomp"></web-comp>
```

Запустите вашего микроприложение и проверьте веб компонент кнопку.