# Podstawowe definicje

* Moduł – splata elementy Angulara w jedną całość. Każda aplikacja zawiera przynajmniej jeden moduł (app.module.ts). W tym miejscu powinny znajdować się odwołania do komponentów.
* Komponent – Część aplikacji, która komunikuje się z użytkownikiem, w tym miejscu powstaje to co widać na ekranie.
* Serwis – zaplecze naszej aplikacji, dostarcza dane, zajmuje się ich przekształcaniem, wykonuje zapytania http oraz obsługuje komunikację między komponentami.

# Deklaracja zmiennych

Deklaracja zmiennej wygląda następująco: *nazwaZmiennej = wartość;*   
Aby wyświetlić wartość zmiennej należy użyć Interpolacji.

# Deklaracja metod

Metody deklarujemy w sposób: *nazwaMetody(): typZwracany{ instrukcja}* np:   
getFooter(): string{   
return „Tekst”;   
}

Metody deklarowane są w *app.component.ts* i w nagłówku *export class AppComponent.*

# Interpolacja

Używana, aby wyświetlić dane zawarte w zmiennej czy metodach. Należy zadeklarować zmienną w *app.component.ts* a następnie wywołać interpolację poprzez {{ *nazwa\_zmiennej }}*. Jeżeli tekst w zmiennej zmieni się to automatycznie zostaje wyświetlony aktualny w interpolacji. Wyświetla tylko typy proste (booling, string, number). Nie można wypisywać całych obiektów – tylko ich konkretne pola poprzez *nazwaObiektu.pole*

# Change detection

Jest to mechanizm odpowiedzialny za wykrywanie zmian w zmiennych i odświeżaniu HTML. Zostaje wywołany po zapytaniach http lub eventach typu button.

# Operator Pipe

Symbolizowany jest znakiem |. Służy do modyfikowania danych przed wyświetleniem ich przez Interpolację, np. {{*nazwa\_zmiennej* | *uppercase*}} wyświetli zawartość zmiennej dużymi literami.

# Question mark

Symbolizowany znakiem ?. Po jego użyciu wyświetlenie interpolacji będzie wstrzymane aż do momentu aż znajdzie się w zmiennej treść do wyświetlenia. Wykorzystywane np. kiedy pobieramy jakieś dane do zmiennej z serwera i należy na te dane chwile poczekać. Przykład to {{*metoda?.date*}} gdzie wyświetlimy datę z pobieraną w metodzie, dopiero kiedy pole to będzie miało wartość.

# ngFor

Dyrektywa służąca wyświetlaniu (iterowaniu) po listach i obiektach.  
Aby wyświetlić listę należy w dokumencie *.html* użyć polecenia:  
*\*ngFor=”let zmienna of nazwa\_listy/obiektu”*

# Indeks

Dla *ngFor* można użyć wbudowanej funkcji indeksowania, która można nam wypisać liczby porządkowe w punktach z listy. Aby zadeklarować indeksowanie, należy:  
*\*ngFor=”let zmienna of nazwa\_listy/obiektu; let zmienna\_indeksu = index”*Zmienną zadeklarowaną jako indeks, można wyświetlać poprzez interpolację (indeksowanie zaczyna się od wartości 0).

# Interface

Służy do reprezentowania danych. Dzięki temu możemy opisać właściwości obiektów, zdefiniować jego pola i typy.

Aby utworzyć interface należy w katalogu *app* dodać nowy plik *nazwa.ts*. Następnie w tym piku eksportujemy nasz interface:  
*export interface nazwa\_interface {pola i dane}*.   
Po utworzeniu interface’u, należy zaimportować go w pliku *app.component.ts* poprzez polecenie:   
*import {nazwa\_interface} from ‘./nazwa\_pliku’;*

Aby zadeklarować listę o typie utworzonym w interface, należy w dokumencie *app.component.ts* dodać listę:  
*nazwa\_listy: nazwa\_interface[] = [wartości];*

Aby wyświetlić pola z zadeklarowanej listy należy w dokumencie *app.component.html* w odpowiednim znaczniku uzyć dyrektywy:  
\**ngFor = „let nazwa\_zmiennej of nazwa\_listy”*a następnie wyświetlić odpowiednie pola używając interpolacji:  
{{*nazwa\_zmiennej.nazwa\_polaListy}}*Zmienna jest widoczna w obrębie znacznika i nie można jej wywołać poza znacznikiem.

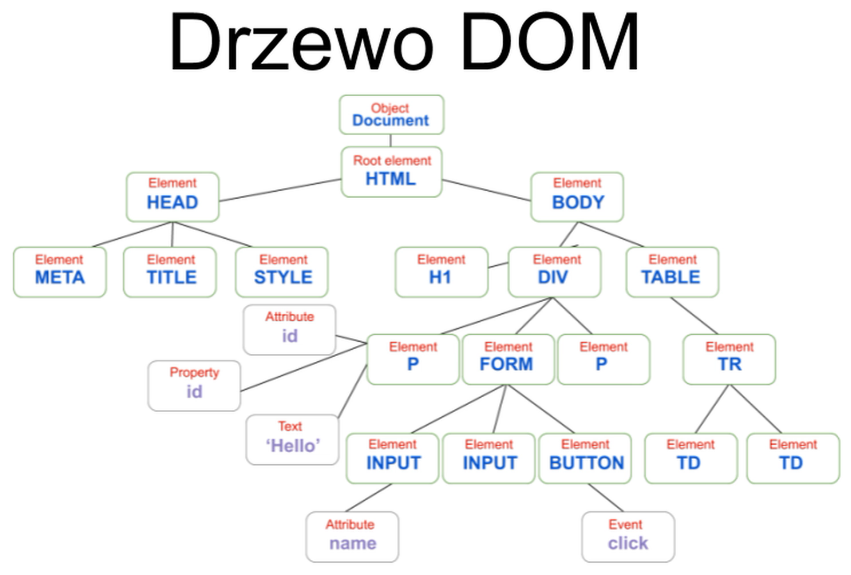
# Data binding

Jest to synchronizacja danych pomiędzy widokiem HTML a komponentem. Może działać w jedną stronę (HTML do komponentu lub odwrotnie) albo w obie.  
Synchronizacja z widoku HTML do komponentu (event binding), np:   
<button *(click)=”cleartList()”></button>*gdzie w okrągłym nawiasie mamy nazwę eventu a następnie metodę z klasy komponentu.

Synchronizacja z komponentu do widoku HTML (Property binding), np.:  
Interpolacja,  
*<input type=”text” [value]=”task.name”>*

Synchronizacja w obu kierunkach (two-way binding), to połączenie nawiasów kwadratowych oraz okrągłych, np.:  
*<input type=”text” [(value)}=”task.name”>*

Plik HTML jest drzewiastą strukturą - DOM, która udostępnia API do pobierania, modyfikowania, dodawania i usuwania elementów:



# $event

Metoda służąca do przechwycenia obiektu wytworzonego przez określoną metodę, np.:  
*(keyup)=”nazwa\_metody($event)”* – pozwala na przechwytywania znaków z pola tekstowego do podanej metody. Metoda nie jest zalecana przez oficjalną dokumentację Angular’a.

# Template reference

Umożliwia odwołanie się bezpośrednio do elemntu HTML i utworzenie referencji do obiektu DOM, np., możemy odwołać się do pola tekstowego i wyciągnąć z niego zapisaną przez użytkownika treść. Aby zadeklarować template reference należy w znaczniku elementu HTML zapisać: *#nazwaReferencji*.

# Bindowanie

Umożliwia deklarację właściwości elementów HTML tak aby np., po wykonaniu jakieś czynności aktywować/dezaktywować przycisk. *<button* ***[disable]****-„counter > 1 „>Przycisk”</button>*

# Dyrektywa NgModel

Pozwala na dwustronne przekazywane danych między komponentem a plikiem HTML. Uzywana z wykorzystaniem ***[(ngModel)****]=”taskName”*.   
Przed rozpoczęciem używanie tej dyrektywny, należy dodać w pliku *app.module.ts* import forms module:  
 *import { FormModule } from ‘@angular/forms’;*   
oraz dopisać *FormsModule* do *imports: [].*Jest to bardziej zalecany sposób synchronizacji danych niż $event oraz Template reference

# Dyrektywa ngIf oraz ngSwitch

Są to dyrektywy strukturalne wykorzystywane do wyświetlania elementów HTML po spełnieniu pewnego warunku. Pozwalają na dodawanie i usuwanie elementów.

## ngIf

Przykład ngIF: *<h1 \*ngIf = „show()”> Hello </h1>*

Różnica między igIf a Visibility jest taka, że igIf usuwa element ze strony co powoduje przesunięcie się pozostałych elementów na miejsce już usuniętego. Visiblity sprawia, że element nadal jest na stronie, ale nie jest on widoczny (zostaje tylko ukryty, ale nadal zajmuje miejsce na stronie oraz w pamięci).  
W przypadku bardziej rozbudowanych opcji, wykorzystać można *else:*  
*<ul class=’collection’ \*ngif=task.length>0;else nic”>  
…  
<ng-template #nic> <h2> Nie ma nic </h2> </ng-template>*Jeżeli warunek będzie spełniony zostanie wyświetlona lista, jeżeli nie zostanie wywołana metoda #nic zadeklarowana w tymczasowej metodzie NG (wykorzystywane przy else dla IF oraz Switch).  
Dla bardziej rozbudowanego przykładu można użyć *<ng-container>  
<div class="row">*

*<ng-container \*ngIf="editMode; then editON; else editOFF"></ng-container>*

*<ng-template #editON>*

*<button class="btn col s2 offset-s10" (click)="switchEditMode()">Dodaj zadanie</button>*

*</ng-template>*

*<ng-template #editOFF>*

*<button class="btn col s2 offset-s10" (click)="switchEditMode()">Zakończ</button>*

*</ng-template>*

*</div>*

## ngSwitch

*<ng-container [ngSwitch] = „day”>  
 <span \*ngSwtchCase=”1”>Poniedziałek</span>  
 <span \*ngSwtchCase=”2”>Wtorek</span>  
 …  
 <span \*ngSwtchDefault>Niedziela</span>  
</ng-container>*