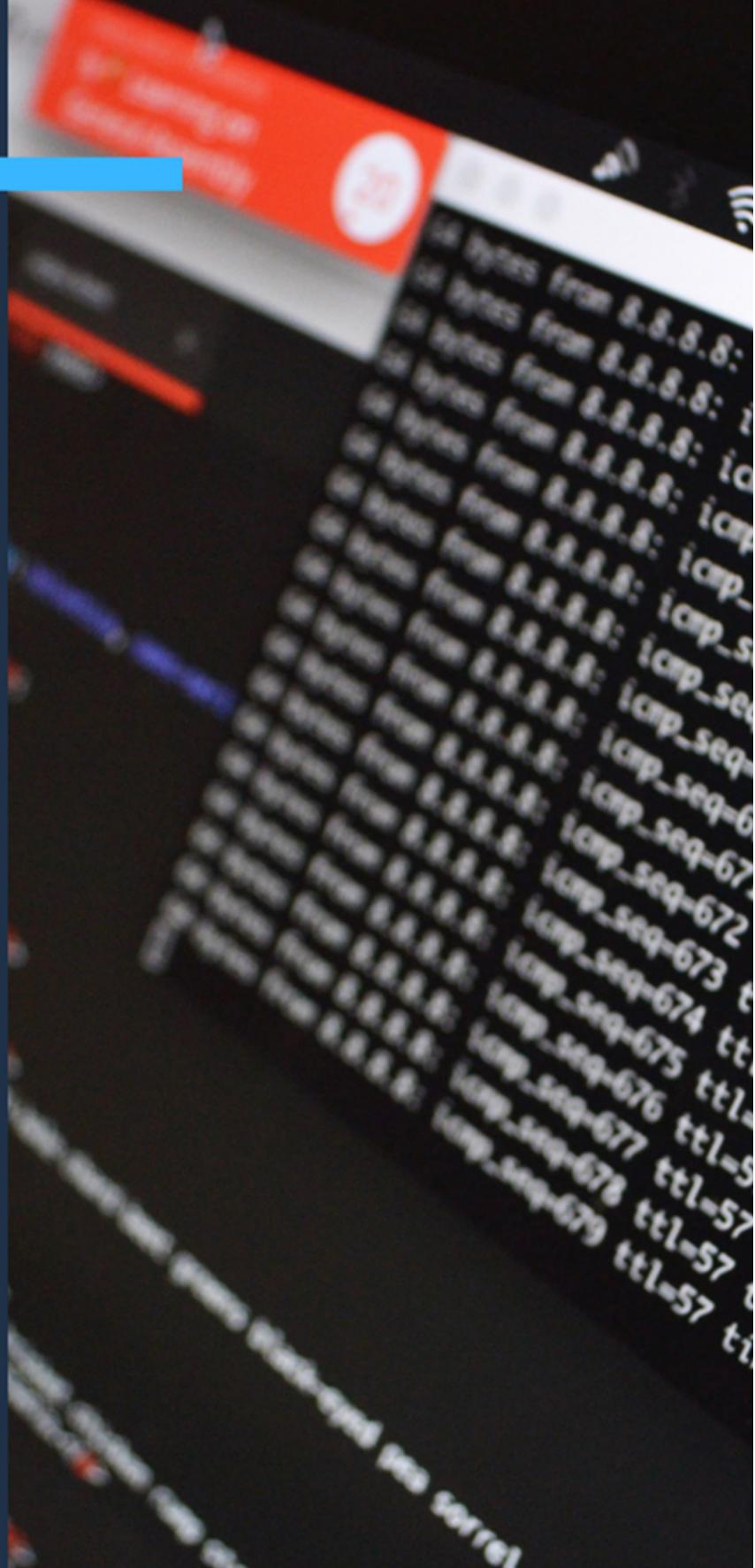


PART 2

Web Development

Membuat Web To-Do
List Menggunakan
Angular



Cascading Training Cyber Security Programme



TEMASEK
FOUNDATION
International



SINGAPORE
POLYTECHNIC | SP



	I
CHAPTER 1	1
CONTAINER VS PRESENTATIONAL	1
CONTAINER	1
PRESENTATIONAL	1
LATIHAN CONTAINER / PRESENTATIONAL	1
ONE WAY DATA FLOW	2
LATIHAN ONE WAY DATA FLOW	3
CHAPTER 2	5
FEATURE MODULES	5
LANGKAH PERCOBAAN	5
BUAT FOLDER CONTAINER, COMPONENT DAN MODELS	8
PINDAHKAN FILE TODO.MODEL.TS	9
BUAT FEATURE MODULE	10
CHAPTER 3	13
CONTAINER COMPONENT TODO DASHBOARD	13
LANGKAH PERCOBAAN	13
CHAPTER 4	17
ADD TODO COMPONENT	17
UPDATE TODOASHBOARD UNTUK MENERIMA OUTPUT	19
BUAT TEMPLATE REF	20
BUAT EVENT CLICK	20
BUAT FUNCTION HANDLE EVENT CLICK	20
BUAT EVENT EMMITTER	21
UPDATE EVENT DI TODOASHBOARD.HTML	22
CHAPTER 5	24
TODOCOUNTCOMPONENT	24
BUAT BOILERPLATE TODOCOUNTCOMPONENT	24
BUAT TAMPILAN TODO COUNT COMPONENT	25
REGISTER COMPONENT KE TODOASHBOARDMODULE	25
AKTIFKAN SELECTOR TODO-COUNT DI TODOASHBOARD	25
INPUT PARAMETER KE TODOCOUNTCOMPONENT	26
UPDATE TODOCOUNTCOMPONENT UNTUK MENERIMA INPUT	26
MENGHITUNG TODO FINISHED / UNFINISHED DI TODOCOUNTCOMPONENT	27
CHAPTER 6	28
TODOITEMCOMPONENT	28

BOILERPLATE TODOITEMCOMPONENT	28
TAMPILAN	29
REGISTER KE MODULE	29
AKTIFKAN SELECTOR	29
LOOPING DAN INPUT PARAMETER	30
UPDATE TODO ITEM UNTUK TERIMA INPUT	31
BUAT EVENT CLICK DI TODOITEM	33
BUAT OUTPUT / EVENT Emitter DARI TODO ITEM	34
BUAT EVENT BINDING DI SELECTOR TODODASHBOARD	35

CHAPTER 1

Container vs Presentational

Pada arsitektur angular terdapat dua jenis component, yang pertama adalah container dan yang kedua adalah component.

- Container (Container Component)
- Component (Presentational Component)

Container

Container adalah component yang berkomunikasi dengan service dan bertugas melakukan rendering terhadap child component nya.

Presentational

Presentational Component adalah component yang menerima input dari Container dan hanya bertugas menampilkan data, tanpa dapat berkomunikasi dengan service. Component ini memiliki input dan melakukan emmit change melalui event output.

Latihan Container / Presentational

Perhatikanlah kembali tugas dari materi hari sebelumnya kemudian mari kita membahas bagaimana membagi component pada aplikasi tersebut.

Pada hari sebelumnya semua kode program masih di simpan di app.component.ts. Pada hari ini kita membahas bagaimana cara memisahkan component menjadi container atau presentational component.

Dari screenshot aplikasi pada hari pertama anda dapat melihat sebagai berikut :



Dari illustrasi ini kita dapat membagi tampilan di atas menjadi beberapa component, yaitu :

- TodoDashboardComponent

- AddTodoComponent
- TodoCountComponent
- TodoItemComponent

Dari ke empat component ini dapat di pisahkan menjadi Container dan Presentational Component. Yang menjadi Container Component adalah :

- TodoDashboardComponent

Dan yang menjadi Presentational Component adalah :

- AddTodoComponent
- TodoCountComponent
- TodoItemComponent

Seperti yang sudah dijelaskan diatas Container Component menjadi yang bertanggungjawab dalam mengatur komunikasi dengan service dan melakukan rendering terhadap child component dalam hal ini Presentational Component.

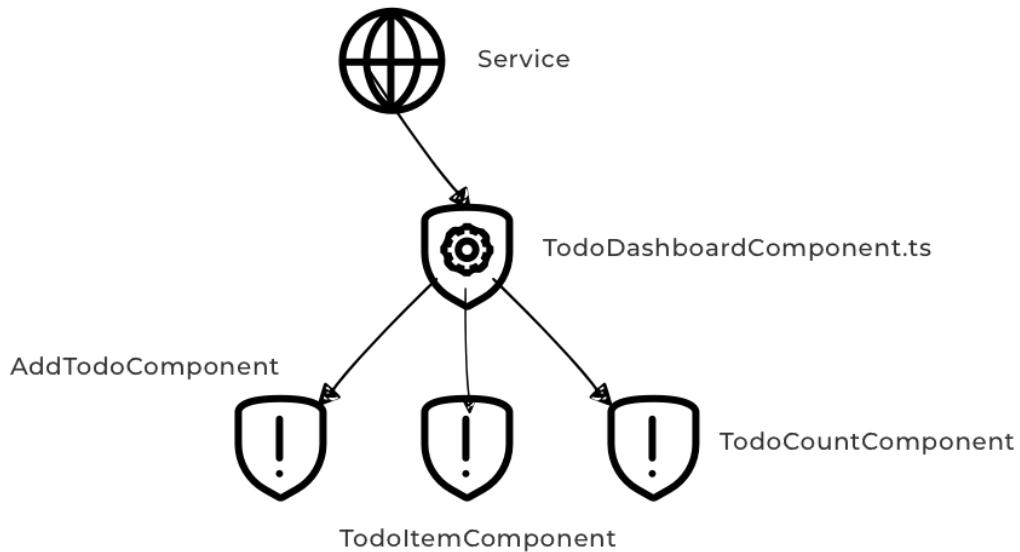
Jadi TodoDashboardComponent mempunyai service yang mengatur data, serta mempunyai child AddTodoComponent, TodoCountComponent dan TodoItemComponent.

AddTodoComponent adalah component yang memiliki tampilan add todo dan mempunyai output yang melakukan event emitter ke appComponent.

TodoCountComponent adalah component yang memiliki tampilan counter todo. Yang berbeda pada component ini adalah dia tidak memiliki output namun memiliki input berupa data array Todos (proses melakukan count dan menampilkan dilakukan di component ini)

TodoItemComponent adalah component yang menampilkan satu item component yang memiliki input berupa data satu item todo serta memiliki output berupa event untuk mengupdate status finished / unfinished

One Way Data Flow



One way data flow merupakan salah satu cara mengatur alur data pada angular. Selain one way data flow juga ada two way data flow yang sudah dijelaskan pada hari pertama.

Dengan menggunakan konsep one way data flow perubahan data dari satu Presentational Component baru akan di simpan secara permanent melalui service setelah Presentational Component mengirimkan event emitter ke Container Component, dimana pada Container Component baru data tersebut di persistence kan melalui service.

Latihan One Way Data Flow

Buatlah One Way Dataflow yang sesuai untuk Container Component dan Presentational Component yang sudah di jabarkan di atas.

ContainerComponent

Component	Input	Output / Event Emitter
TodoDashboardComponent	-	-

- Service untuk mengambil data todo
- Service untuk delete todo
- Service untuk edit todo
- Service untuk add todo
- Service untuk toggle todo

Presentational Component

Component	Input	Output / Event Emitter
TodoCountComponent	Todos[]	-
TodoItemComponent	Todo	ToggleFinished
AddTodoComponent	-	AddTodo

CHAPTER 2

Feature Modules

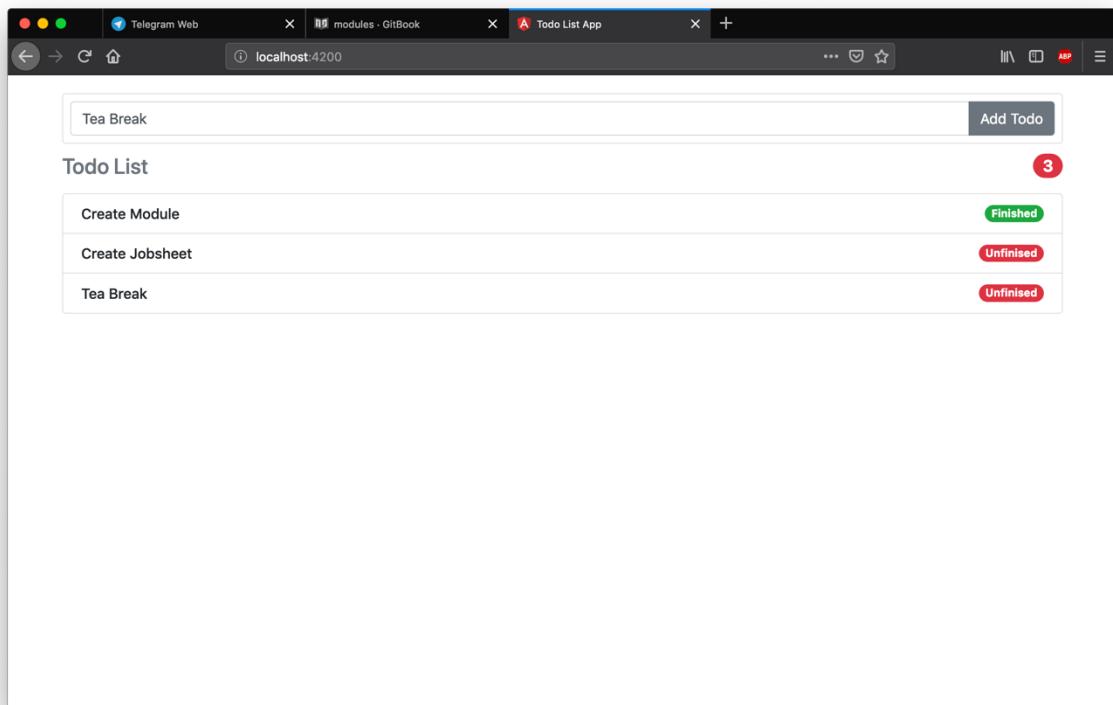
Feature modules adalah sebuah module yang berisi component component yang dibutuhkan untuk menyelesaikan suatu tugas atau suatu fitur. Pada percobaan kali ini setelah anda berhasil memetakan mana yang menjadi Container Component dan mana yang menjadi Presentational Component akan dilanjutkan dengan membuat sebuah feature modules untuk TodoComponent, AddTodoComponent, TodoCountComponent, dan TodoItemComponent dalam sebuah Module TodoDashboard.

A feature module is an organizational best practice, as opposed to a concept of the core Angular API. A feature module delivers a cohesive set of functionality focused on a specific application need such as a user workflow, routing, or forms. While you can do everything within the root module, feature modules help you partition the app into focused areas. A feature module collaborates with the root module and with other modules through the services it provides and the components, directives, and pipes that it shares.

Langkah Percobaan

Clone lah repositori berikut ini dari

github <https://github.com/siubie/DayOneSeed.git> kemudian bukalah menggunakan visual studio code dan switch dari branch `master` ke branch `step1` setelah berhasil mengganti branch jalankan server angular dengan perintah `ng serve` pastikan aplikasi todo yang ada sekarang berjalan dengan baik.



Branch Step 1 adalah branch yang berisi finished code dari latihan sebelumnya. Pada branch ini sudah berhasil dibuat sebuah aplikasi todo list sederhana dengan fitur add dan toggle finished. Namun pada branch ini masih belum menggunakan konsep pengembangan yang berbasis component dengan baik karena semua kode program di buat di file app.component.ts dan app.component.html.

Pada percobaan ini kita akan membuat sebuah feature module yang bertugas menangani fitur Todo yang ada pada branch Step 1, pada module ini akan di isi dengan container component dan presentational component yang telah di analisa pada modul sebelumnya.

Untuk melaksanakan percobaan ini pastikan bahwa branch yang anda gunakan sekarang adalah branch step1, cara melakukan pengecekan dengan menjalankan perintah berikut ini pada terminal di folder project anda. (pastikan git sudah terinstall pada sistem operasi anda).

```
git branch
```

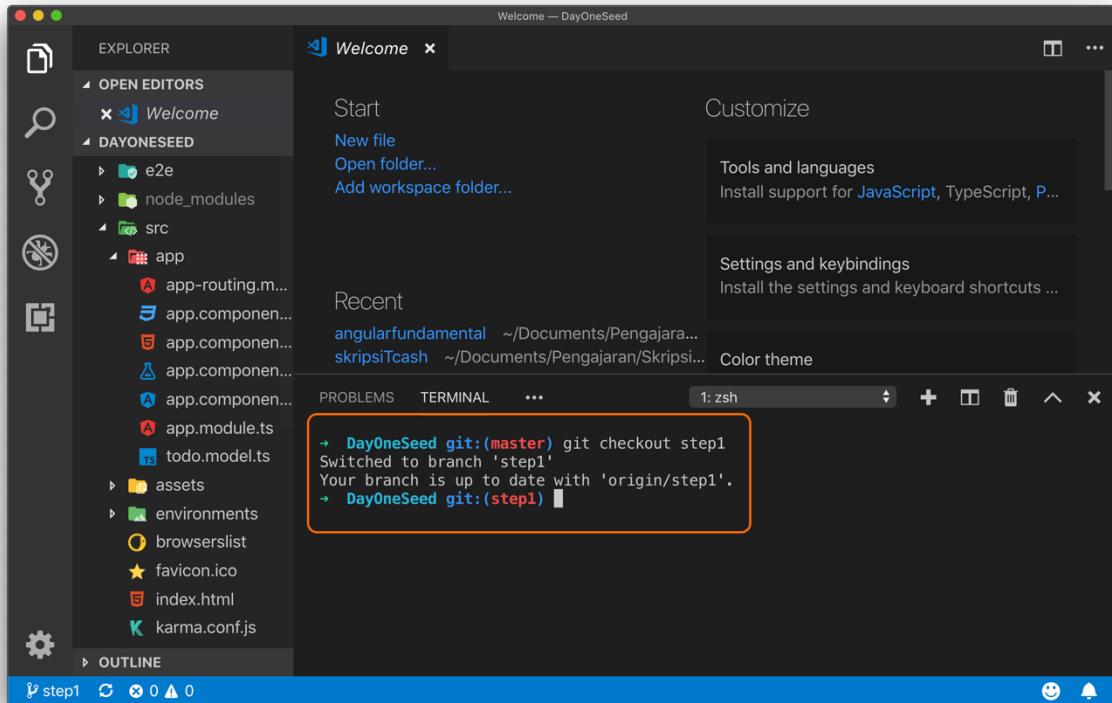
The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays a project structure for 'DAYONESEED' with files like 'e2e', 'node_modules', 'src/app', and various component and module files. The 'TERMINAL' tab is active at the bottom, showing the command 'DayOneSeed git:(step2) git branch'. The output of this command is visible in the terminal window, listing branches: 'master', 'step1', and 'step2'. The entire terminal area is highlighted with an orange rectangle.

Hasil perintah git branch dapat anda lihat pada gambar dibawah ini :

This screenshot is identical to the one above, showing the Visual Studio Code interface with the terminal output of 'git branch'. The terminal window shows the branches 'master', 'step1', and 'step2'. The terminal area is again highlighted with an orange rectangle.

Current branch dapat anda lihat dengan mengecek tanda asterisk * pada hasil terminal jika berdasarkan gambar diatas branch masih berada pada branch

master, untuk itu anda perlu melakukan perubahan untuk pindah ke branch step1 dengan perintah berikut ini.

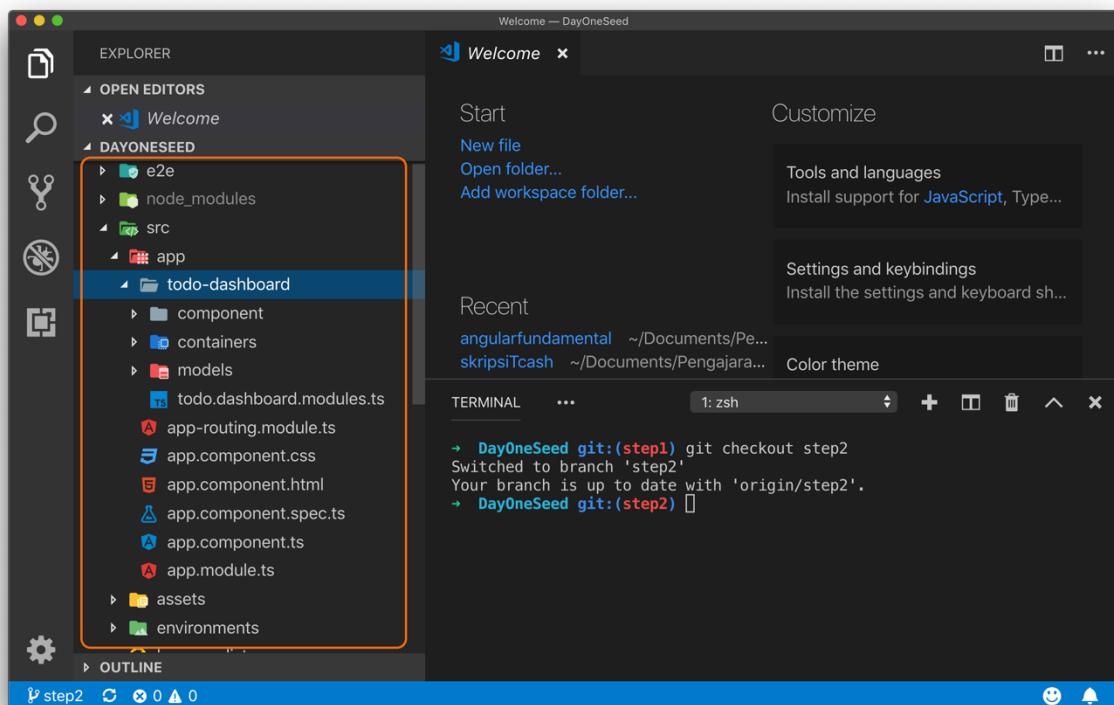


The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows the project structure with folders like e2e, node_modules, and src/app containing various files like app-routing.m... and app.module.ts.
- TERMINAL** view: Displays the command "git checkout step1" being run, with the output showing the branch has been switched to "step1" and is up-to-date with "origin/step1".
- PROBLEMS** view: Shows no problems.
- Customize** dropdown menu: Options include "Tools and languages" (install support for JavaScript, TypeScript, etc.) and "Settings and keybindings" (install settings and keyboard shortcuts).
- Recent** list: Shows recent projects like "angularfundamental" and "skripsiTcash".
- Bottom status bar**: Shows the current branch as "step1".

Buat Folder Container, Component dan Models

Setelah berhasil pindah ke branch step1 mari kita lanjutkan dengan membuat sebuah feature module yang bernama todo-dashboard, untuk membuatnya mulailah dengan menambahkan folder baru bernama todo-dashboard pada folder app setelah itu tambahkan tiga folder lagi yaitu folder containers, component dan models berikut ini folder tree yang harus dibuat :



```
|- todo-dashboard
  |- component
  |- container
  |- models
```

Pindahkan File todo.model.ts

Setelah berhasil membuat folder lanjutkan dengan memindahkan file `todo.model.ts` dari folder `app` ke folder `app/todo-dashboard/model`, ini dilakukan agar semua file yang berhubungan dengan fitur todo berada dalam satu folder.

```

app.module.ts — DayOneSeed
EXPLORER OPEN EDITORS 1 UNSAVED
todo.dashboard.html (deleted from disk) todo.dashboard.css (deleted from disk) app.component.html app.module.ts ...
DAYONSEED
e2e
node_modules
src
app
todo-dashboard
component
container
models
todo.model.ts
app-routing.module.ts
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
assets
environments
browserslist
favicon.ico
index.html
karma.conf.js
main.ts
polyfills.ts
styles.css
OUTLINE
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL
1: node_zsh
problems
app-routing.module.ts
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
todo-dashboard
component
container
models
todo.model.ts
4 directories, 7 files
+ app git:(modulDay2) x []
Ln 14, Col 1 Spaces: 2 UTF-8 LF TypeScript 3.3.1 TSLint Prettier: ✓
modulDay2* 0 0 3 "app.module.ts" 14L 372C written

```

```

.
├── component
└── container
  └── models
    └── todo.model.ts

```

Buat Feature Module

Mari dilanjutkan dengan menambahkan sebuah file baru bernama todo.dashboard.module.ts, Langkah pembuatan module ini sama dengan langkah pada modul sebelumnya, untuk lebih memahami prosesnya mari kita ulang kembali prosesnya.

Setiap module pada file module angular membutuhkan library dari `@angular/core` yaitu library `NgModule` yang merupakan decorator yang membedakan antara module dengan component dan file lain. Untuk itu pada file `todo.dashboard.module.ts` tambahkanlah statement import berikut ini.

```
import { NgModule } from "@angular/core";
```

Selain itu include can juga library `CommonModule` dari `@angular/common`

```
import { CommonModule } from "@angular/common";
```

Setelah menambahkan `NgModule` dan `CommonModule` kita dapat menggunakan decorator `@NgModule` dan mengimport `CommonModule` ke Class `TodoDashboardModule`.

```
import { NgModule } from "@angular/core";
import { CommonModule } from "@angular/common";

@NgModule({
  declarations: [],
  imports: [CommonModule],
```

```

exports: []
})
export class TodoDashboardModule {}

```

Berikut ini hasil kode program todo.dashboard.module

```

todo.dashboard.modules.ts — DayOneSeed
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3
4 @NgModule({
5   declarations: [],
6   imports: [CommonModule],
7   exports: []
8 })
9 export class TodoDashboardModule {}
10

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: zsh
→ DayOneSeed git:(step1) git checkout step2
Switched to branch 'step2'.
Your branch is up to date with 'origin/step2'.
→ DayOneSeed git:(step2) ✘

Ln 10, Col 1 Spaces: 2 UTF-8 LF TypeScript 3.3.1 TSLint Prettier:✓ 🎉 🚨

```

Yang barusaja anda buat adalah sebuah template module pada angular dengan library minimal dimana pada module ini nanti akan ditambahkan beberapa component dan container yang digunakan pada module ini.

Selanjutnya agar TodoDashboardModule yang kita buat dapat digunakan di root module (AppModule), TodoDashboardModule harus di import sebagai sebuah module baru di file app.module.ts. Ubahlah kode program pada file app.module.ts menjadi seperti pada gambar dibawah ini.

```

import { BrowserModule } from "@angular/platform-browser";
import { NgModule } from "@angular/core";

import { AppRoutingModule } from "./app-routing.module";
import { AppComponent } from "./app.component";
import { TodoDashboardModule } from "./todo-dashboard/todo.dashboard.module";

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, AppRoutingModule, TodoDashboardModule],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}

```

```

app.module.ts — DayOneSeed
1 import { BrowserModule } from '@angular/platform-browser';
2 import { NgModule } from '@angular/core';
3
4 import { AppRoutingModule } from './app-routing.module';
5 import { AppComponent } from './app.component';
6 import { TodoDashboardModule } from './todo-dashboard/todo.dashboard.module';
7
8 @NgModule({
9   declarations: [AppComponent],
10   imports: [BrowserModule, AppRoutingModule, TodoDashboardModule],
11   providers: [],
12   bootstrap: [AppComponent]
13 })
14 export class AppModule {}
```

PROBLEMS 17 OUTPUT DEBUG CONSOLE TERMINAL

```

Date: 2019-02-19T17:32:58.625Z - Hash: 24ec8829910d5e060d3a
Time: 326ms
4 unchanged chunks
Chunk {main} main.js, main.js.map (main) 14.5 kB [initial] [rendered]
[wdm]: Compiled successfully.

ERROR in src/app/app.module.ts(6,10): error TS2305: Module './todo-dashboard/todo.dashboard.module' has no exported member 'TodoDashboardModule'.
```

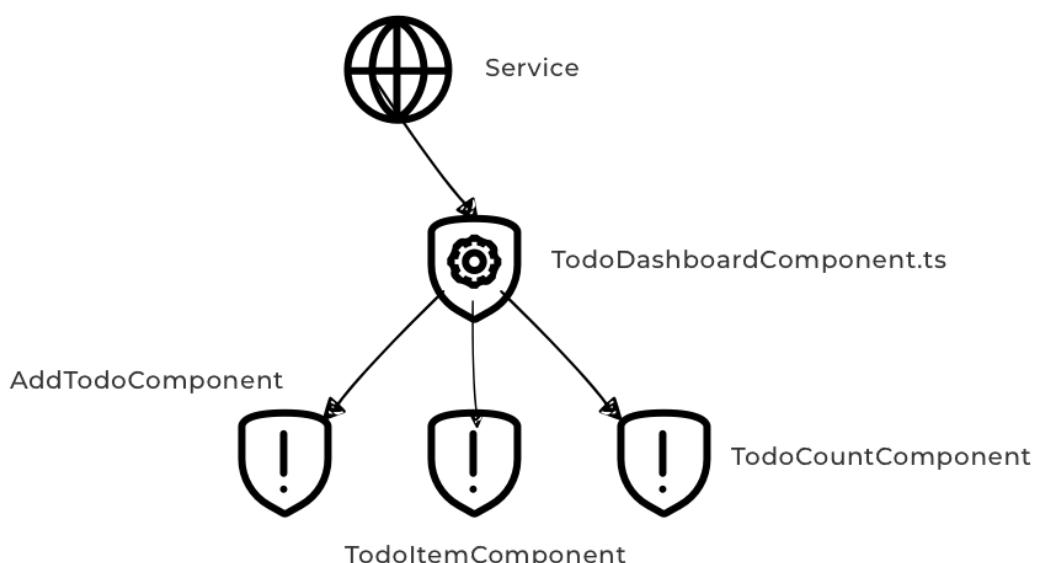
Ln 15, Col 1 Spaces: 2 UTF-8 LF TypeScript 3.3.1 ▲ TSLint Prettier: ✓

Setelah melakukan import TodoDashboardModule compiler angular akan memberikan error bahwa TodoDashboardModule tidak memiliki "exported member", hal ini wajar karena kita akan menambahkan membuat container component dan meng export nya di langkah selanjutnya.

CHAPTER 3

Container Component Todo Dashboard

Selanjutnya kita akan membuat container component sesuai dengan hasil analisa pada bab sebelumnya seperti pada gambar dibawah ini.



Pada component tree di atas dapat anda lihat bahwa terdapat 4 component dengan TodoDashboardComponent menjadi container dengan 3 presentational component yaitu AddTodoComponent, TodoItemComponent, dan TodoCountComponent. Pada percobaan kali ini kita akan membuat item item pada component tree tersebut dimulai dari TodoDashboardComponent.

Langkah Percobaan

Langkah pertama siapkan sebuah struktur folder untuk menyimpan file file yang dibutuhkan berdasarkan component tree yang sudah di desain.

```
└── todo-dashboard
    ├── component
    │   ├── add-todo
    │   │   ├── add.todo.component.ts
    │   │   └── add.todo.html
    │   ├── todo-count
    │   │   ├── todo.count.component.ts
    │   │   └── todo.count.html
    │   ├── todo-item
    │   │   ├── todo.item.component.ts
    │   │   └── todo.item.html
    └── containers
```

The screenshot shows the VS Code interface with the following details:

- File Tree (EXPLORER):** Shows the project structure with a folder named "todo-dashbaord" highlighted by a red box. Inside this folder are subfolders "component", "containers", and "models".
- Code Editor:** Displays the file `todo.dashboard.component.ts`. The code defines a component named `TodoDashboardComponent` with a selector of `'todo-dashboard'`, style URLs pointing to `todo.dashboard.css`, and a template URL pointing to `todo.dashboard.html`.
- Terminal:** Shows a series of git commands related to branches `step1` and `step2`.
- Status Bar:** Shows the current file is `todo.dashboard.component.ts`, line 20, column 26, using UTF-8 encoding, with TypeScript 3.3.1, TSLint, and Prettier enabled.

```

todo.dashboard.component.ts
├── todo.dashboard.component.ts
├── todo.dashboard.css
└── todo.dashboard.html

models
└── todo.model.ts

todo.dashboard.modules.ts

```

```

import { Component } from '@angular/core';
import { Todo } from '../models/todo.model';
@Component({
  selector: 'todo-dashboard',
  styleUrls: ['todo.dashboard.css'],
  templateUrl: 'todo.dashboard.html'
})
export class TodoDashboardComponent {
  todoList: Todo[] = [];
  handleAddTodo(task: string) {
    const itemTodo = new Todo('1', task, false);
  }
}

```

Seperti pada gambar dan folder tree di atas buatlah sebuah tiga buah di dalam folder todo-dasboard yaitu :

- component
- containers
- models

setelah itu di dalam folder component siapkan folder lain untuk menyimpan file file yang digunakan untuk presentational component :

- add-todo
- todo-count
- todo-item

Di dalam folder containers siapkan file untuk container component `TodoDashboardComponen`. Setelah semua folder dibuat lengkapi folder tersebut dengan file file yang ada di file tree, biarkan file tersebut kosong tapi dengan ekstensi file yang sesuai.

Kemudian untuk file `todo.dashboard.component.ts` kita tambahkan boilerplate code untuk pembuatan component dibawah ini.

```

import { Component } from "@angular/core";
@Component({
  selector: "todo-dashboard",
  styleUrls: ["todo.dashboard.css"],
  templateUrl: "todo.dashboard.html"
})
export class TodoDashboardComponent {}

```

Kemudian lanjutkan dengan mengedit file todo.dashboard.html

```

<div class="row">
  <div class="col-md-12 order-md-2 mb-4">
    <!-- <add-todo></add-todo>
    <todo-count></todo-count>
    <todo-item></todo-item> -->
  </div>
</div>

```

Kode html diatas digunakan pada todo.dashboard.html sebagai template container component dimana TodoDashboardComponent menjadi parent dari add-todo, todo-count dan todo-item. Perhatikan bahwa child component masih di pasang sebagai komentar agar tidak di eksekusi untuk menghindari error.

Untuk file todo.dashboard.css tambahkan kode program seperti dibawah ini :

```

.todo-input {
  margin-bottom: 10px;
}

```

Setelah semua file pada folder containers selesai di isi lanjutkan dengan melakukan registrasi component tersebut ke module dalam hal ini ke file todo.dashboard.module.ts. Untuk menambahkannya lakukan import todo.dashboard.component di todo.dashboard.modules dengan menggunakan kode program dibawah ini:

```
import { TodoDashboardComponent } from "./containers/todo.dashboard.component";
```

Setelah import dilakukan maka TodoDashboardComponent dapat digunakan sebagai declaration di decorator@NgModule. Berikut ini kode program todo.dashboard.modules.ts setelah proses ini dilakukan

```

import { NgModule } from "@angular/core";
import { CommonModule } from "@angular/common";
import { TodoDashboardComponent } from "./containers/todo.dashboard.component";

@NgModule({
  declarations: [TodoDashboardComponent],
  imports: [CommonModule],
  exports: [TodoDashboardComponent]
})
export class TodoDashboardModule {}

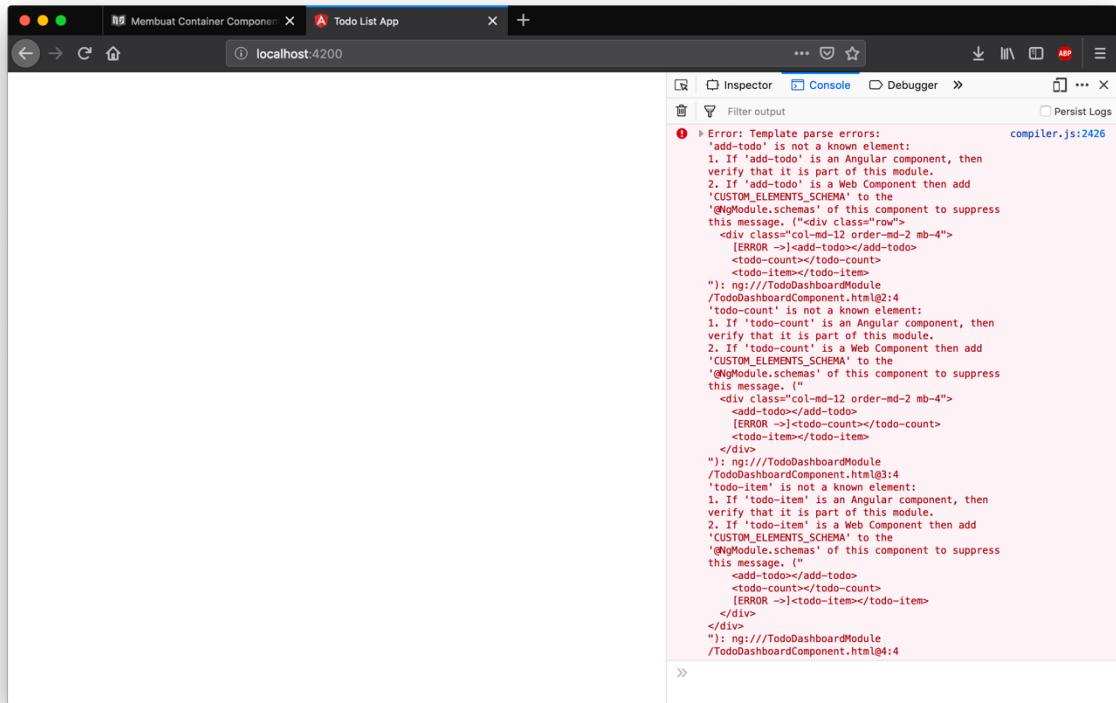
```

Selain menjadikan TodoDashboardComponent sebagai declaration, module ini juga melakukan export TodoDashboardComponent, ini artinya

`TodoDashboardComponent` dapat digunakan oleh module lain yang melakukan import module ini.

Langkah selanjutnya adalah menggunakan container component ini di app component dengan merubah file `app.component.html` menjadi seperti dibawah ini.

```
<div class="container">
  <todo-dashboard></todo-dashboard>
</div>
```



Jika server angular dijalankan pada tahap ini dan anda mendapatkan error pada console seperti gambar di atas itu karena child component belum di pasang komentar pada file `todo.dashboard.html`

CHAPTER 4

Add Todo Component

Selanjutnya kita akan menyelesaikan langkah selanjutnya yaitu membuat presentational component AddTodoComponent. Presentational component ini mempunyai tugas menerima input dari user untuk disimpan sebagai Item Todo. Untuk menyelesaikan component ini lakukanlah langkah percobaan dibawah ini.

Sesuai dengan desain pada chapter sebelumnya berdasarkan tabel dibawah ini

Component	Input	Output / Event Emitter
TodoCountComponent	Todos[]	-
TodoItemComponent	Todo	ToggleFinished
AddTodoComponent	-	AddTodo

AddTodoComponent memiliki output / event emitter yang bertujuan untuk menambahkan Todo. Selanjutnya mari kita isi file component yang sebelumnya sudah di buat pada file `add.todo.component.ts`. Bukalah file tersebut kemudian tambahkan import yang menjadi syarat untuk membuat sebuah file menjadi component.

```
import { Component } from '@angular/core';
```

Component di import agar kita dapat menggunakan decorator `@component`. Kemudian kita dapat menggunakan decorator `@Component`.

```
@Component({
  selector: 'add-todo',
  styleUrls: [],
  templateUrl: 'add.todo.html'
})
```

Pada kode program diatas kita menggunakan decorator `@Component` kemudian menggunakan selector, styleUrls dan templateUrl seperti pada kode program di atas. Dengan decorator diatas kita dapat menggunakan tag `<add-todo></add-todo>` di template html, untuk component ini tidak digunakan style dan template nya di pisah di file `add.todo.html`.

Selanjutnya setelah decorator buatlah class typescript dengan nama `AddTodoComponent`

```
export class AddTodoComponent {
  constructor() {}
```

Di kode program di atas kita membuat sebuah class typescript dengan nama `AddTodoComponent`, penting untuk diperhatikan ada statement `export` sebelum class ini diperlukan agar class ini dapat di import di file lain, jika tidak ada statement ini class ini tidak dapat di import dari file lain.

Selanjutnya lengkapilah file `add.todo.html`, di file ini kita isikan tampilan dari component add todo yang kita buat sebelumnya.

```

<form class="card p-2 todo-input" target="#">
  <div class="input-group">
    <input
      type="text"
      class="form-control"
      placeholder="Add Your Todo List Here"
    />
    <div class="input-group-append">
      <button
        type="submit"
        class="btn btn-secondary"
      >
        Add Todo
      </button>
    </div>
  </div>
</form>

```

Setelah menambahkan file html ini selanjutnya kita harus melakukan registrasi component add.todo.component.ts ke module TodoDashboard, untuk melakukannya dapat dilakukan dengan mengimport add.todo.component kemudian menjadikannya sebagai declarations pada module TodoDashboard. Tambahkan kode program dibawah ini untuk mengimport add.todo.component.

```

import { AddTodoComponent } from './component/add-todo/add.todo.component';
@NgModule({
  declarations: [TodoDashboardComponent, AddTodoComponent],
  imports: [CommonModule],
  exports: [TodoDashboardComponent]
})

```

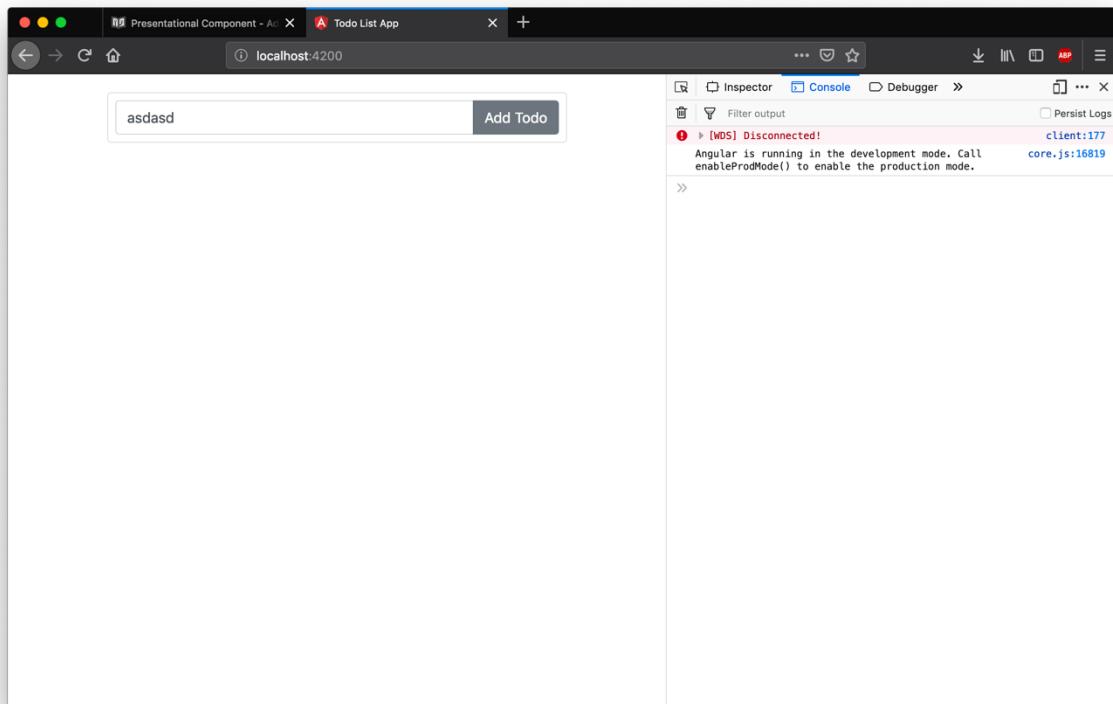
Selanjutnya bukalah kembali file todo.dashboard.html dan hapuslah komentar untuk selector add-todo sehingga kode program pada file ini menjadi seperti dibawah ini.

```

<div class="row">
  <div class="col-md-12 order-md-2 mb-4">
    <add-todo></add-todo>
    <!-- <todo-count></todo-count>
    <todo-item></todo-item> -->
  </div>
</div>

```

Jika semua kode program diketikkan dengan benar akan didapatkan tampilan seperti dibawah ini jika server dijalankan:



Sejauh ini kita sudah berhasil memindahkan tampilan untuk menambah todo ke AddTodoComponent namun ketika di klik tombol add todo aplikasi masih tidak berjalan, ingat di awal bahwa AddTodoComponent memiliki output dengan EventEmitter bernama AddTodo.

Selanjutnya untuk menambahkan output dan memastikan AddTodoComponent bekerja dengan baik lakukanlah langkah langkah berikut.

Update TodoDashboard untuk menerima output

Langkah pertama yang dilakukan adalah menambahkan sebuah custom event pada file todo.dashboard.html dimana pada file ini terdapat selector `<add-todo></add-todo>`. Custom event ini akan digunakan sebagai output dari AddTodoComponent, custom event pada dasarnya sama dengan event binding biasa namun pada custom event kita dapat memberikan nama sesuai dengan kebutuhan kita. Ubahlah file todo.dashboard.html menjadi seperti dibawah ini.

```
<div class="row">
  <div class="col-md-12 order-md-2 mb-4">
    <add-todo (create)="handleAddTodo($event)"></add-todo>
    <!-- <todo-count></todo-count>
    <todo-item></todo-item> -->
  </div>
</div>
```

Kemudian tambahkan implementasi `handleAddTodo` pada file todo.dashboard.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'todo-dashboard',
  styleUrls: ['todo.dashboard.css'],
  templateUrl: 'todo.dashboard.html'
})
export class TodoDashboardComponent {
```

```

handleAddTodo(task: string) {
  console.log(task);
  return false;
}
}

```

Buat Template Ref

Bukalah kembali file `add.todo.html` pada folder `add-todo`, kemudian tambahkan template reference dengan nama task, sehingga kode program pada `add.todo.html` menjadi seperti ini :

```

<form class="card p-2 todo-input" target="#">
  <div class="input-group">
    <input
      type="text"
      class="form-control"
      placeholder="Add Your Todo List Here"
      #task
    />
    <div class="input-group-append">
      <button type="submit" class="btn btn-secondary">
        Add Todo
      </button>
    </div>
  </div>
</form>

```

Buat Event Click

Kemudian buatlah event binding untuk click pada tombol add todo.

```

<form class="card p-2 todo-input" target="#">
  <div class="input-group">
    <input
      type="text"
      class="form-control"
      placeholder="Add Your Todo List Here"
      #task
    />
    <div class="input-group-append">
      <button
        type="submit"
        class="btn btn-secondary"
        (click)="onClickAddTodo(task.value)"
      >
        Add Todo
      </button>
    </div>
  </div>
</form>

```

Pada kode program diatas ditambahkan event binding click pada button add todo dengan handler `onClickAddTodo`.

Buat Function Handle Event Click

Setelah membuat event binding, lanjutkan dengan membuat handler function nya.

```

import { Component } from '@angular/core';
@Component({
  selector: 'add-todo',
  styleUrls: [],
})

```



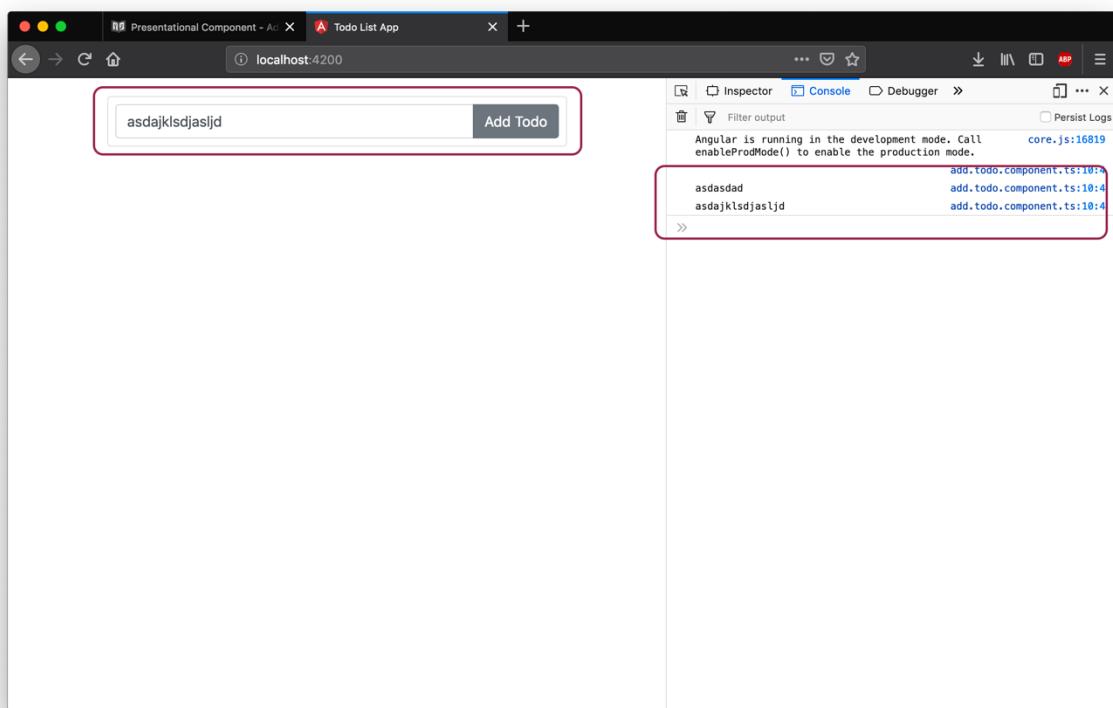
```

        templateUrl: 'add.todo.html'
    })
export class AddTodoComponent {
    constructor() {}
    onClickAddTodo(value: string) {
        console.log(value);
        return false;
    }
}

```

Jika server dijalankan pada tahap ini sudah bisa menerima event click dan membaca serta menampilkan input dari user ke console browser.

Return false pada fungsi onClickAddtodo berfungsi untuk menghentikan proses klik sehingga tidak berpindah ke halaman baru.



Buat Event Emitter

Setelah berhasil mendeteksi perubahan data di click event pada AddTodoComponent, kita harus mengirim perubahan ini ke parent component TodoDashboardcomponent, pengiriman data ini dilakukan dengan menggunakan event emitter.

Tambahkan import Event Emitter dan Output dari @angular/core

```
import { Component, EventEmitter, Output } from '@angular/core';
```

Setelah menambahkan import kita dapat menggunakan decorator @output dan membuat sebuah event emitter untuk mengirim data keluar dari AddTodoComponent ke TodoDashboardComponent.

```

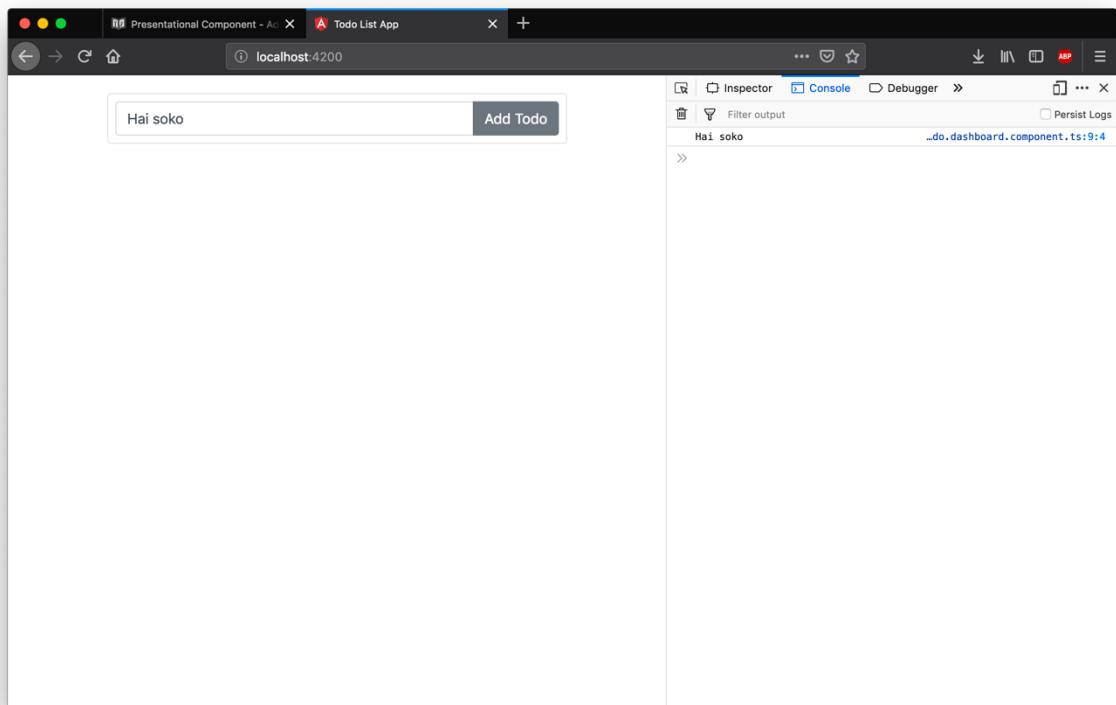
import { Component, EventEmitter, Output } from '@angular/core';
@Component({
  selector: 'add-todo',
  styleUrls: [],
  templateUrl: 'add.todo.html'
})
export class AddTodoComponent {
  @Output()
  create: EventEmitter<any> = new EventEmitter();

  constructor() {}

  onClickAddTodo(value: string) {
    this.create.emit(value);
    return false;
  }
}

```

Pada kode program diatas ditambahkan decorator @output untuk menandakan bahwa component ini memiliki output dengan nama create yang berupa sebuah event emitter. kemudain pada function onClickAddtodo dilakukan emit create event dengan nilai data dari input text.



Update Event di TodoDashboard.html

Setelah data dikirim ke todo.dashboard.component kita dapat menggunakan data tersebut untuk merubah dan menambahkan data todo. Untuk menambah data todo lakukan perubahan berikut pada todo.dashboard.component.

Tambahkan import untuk model Todo sehingga dapat digunakan pada component ini.

```
import { Todo } from '../models/todo.model';
```

Buatlah variabel todoList sebagai attribut component tododashboard

```
todoList: Todo[] = [];

Ubah fungsi handle addtodo menjadi seperti berikut ini
```

```
handleAddTodo(task: string) {
  const itemTodo = new Todo('1', task, false);
  this.todoList.push(itemTodo);
  console.log(this.todoList);
  return false;
}
```

Pada kode program di atas dibuat sebuah item todo dan menambahkannya ke todoList.

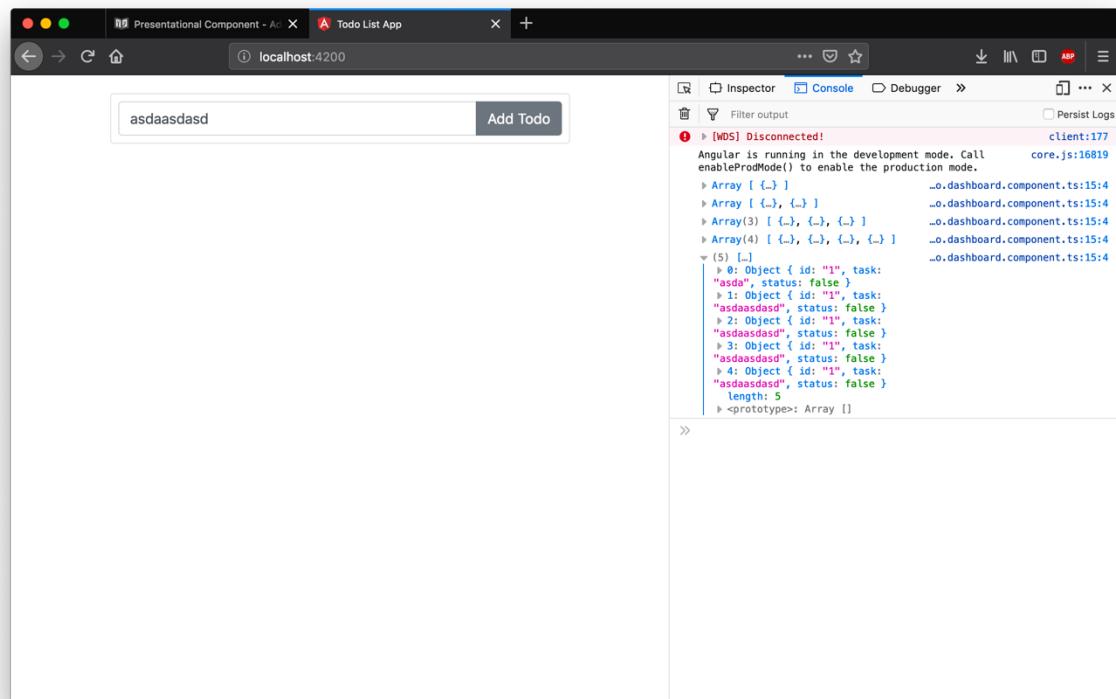
Berikut ini kode program lengkapnya dari todo.dashboard.component.

```
import { Component } from '@angular/core';

import { Todo } from '../models/todo.model';
@Component({
  selector: 'todo-dashboard',
  styleUrls: ['todo.dashboard.css'],
  templateUrl: 'todo.dashboard.html'
})
export class TodoDashboardComponent {
  todoList: Todo[] = [];

  handleAddTodo(task: string) {
    const itemTodo = new Todo('1', task, false);
    this.todoList.push(itemTodo);
    console.log(this.todoList);
    return false;
  }
}
```

Jika server dijalankan dapat dilihat di console bahwa data todo baru ditambahkan ke array todoList



CHAPTER 5

TodoCountComponent

Selanjutnya kita akan menyelesaikan langkah selanjutnya yaitu membuat presentational component TodoCountComponent. Presentational component ini mempunyai tugas menampilkan jumlah todo yang sudah selesai atau belum selesai. Untuk menyelesaikan component ini lakukanlah langkah percobaan dibawah ini.

Sesuai dengan desain pada chapter sebelumnya berdasarkan tabel dibawah ini

Component	Input	Output / Event Emitter
TodoCountComponent	Todos[]	-
TodoItemComponent	Todo	ToggleFinished
AddTodoComponent	-	AddTodo

TodoCountComponent memiliki input yang bertujuan untuk menghitung Todo yang sudah selesai (finished) atau belum.

Buat Boilerplate TodoCountComponent

Bukalah file `todo.count.component.ts` dan tambahkan kode program berikut ini

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'todo-count',
  styleUrls: [],
  templateUrl: 'todo.count.html'
})
export class TodoCountComponent {
  constructor() {}
}
```

Pada kode program diatas dibuat sebuah boilerplate untuk TodoCountComponent dan yang berbeda dengan AddTodoComponent adalah ditambahkan import untuk Input, dimana input ini adalah decorator yang menyatakan bahwa suatu variabel merupakan input untuk component tersebut.

Buat Tampilan Todo Count Component

Tampilan untuk component ini adalah tampilan yang menunjukkan perhitungan jumlah todo yang sudah selesai atau belum.

```
<h4 class="d-flex justify-content-between align-items-center mb-3">
  <span class="text-muted">Todo List</span>
  <span class="badge badge-danger badge-pill">5</span>
</h4>
```

Register Component ke TodoDashboardModule

Selanjutnya daftarkan component ini ke module dengan cara mengimportnya dan menambahkan ke declaration.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { TodoDashboardComponent } from './containers/todo.dashboard.component';
import { AddTodoComponent } from './component/add-todo/add.todo.component';
import { TodoCountComponent } from './component/todo-
count/todo.count.component';

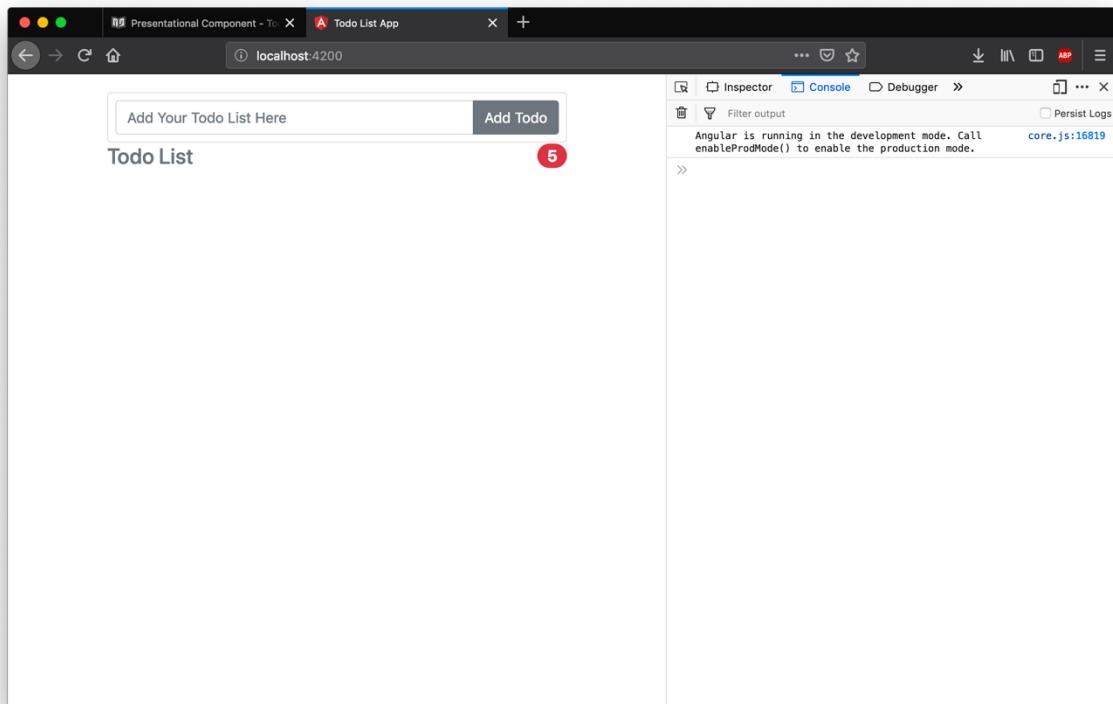
@NgModule({
  declarations: [TodoDashboardComponent, AddTodoComponent, TodoCountComponent],
  imports: [CommonModule],
  exports: [TodoDashboardComponent]
})
export class TodoDashboardModule {}
```

Aktifkan Selector todo-count di TodoDashboard

Bukalah file todo.dashboard.html kemudian aktifkan selector todo-count

```
<div class="row">
  <div class="col-md-12 order-md-2 mb-4">
    <add-todo (create)="handleAddTodo($event)"></add-todo>
    <todo-count></todo-count>
    <!-- <todo-item></todo-item> -->
  </div>
</div>
```

Jika server dijalankan pada kondisi ini tampilan yang didapat seperti gambar dibawah ini, namun ketika diberikan input masih belum memberikan respon apapun terhadap todo count karena belum diberikan input dari todo dashboard.



Input Parameter Ke TodoCountComponent

Berilah input dari variabel `todoList` ke component `TodoCount` dengan menambahkan sebuah property binding di file `todo.dashboard.html`

```
<div class="row">
  <div class="col-md-12 order-md-2 mb-4">
    <add-todo (create)="handleAddTodo($event)"></add-todo>
    <todo-count [todos]="todoList"></todo-count>
    <!-- <todo-item></todo-item> -->
  </div>
</div>
```

Update TodoCountComponent Untuk Menerima Input

Pada langkah sebelumnya sudah di import input decorator namun di kode program sebelumnya decorator ini belum digunakan, selanjutnya gunakanlah decorator ini agar `TodoItemComponent` dapat menerima input dari `TodoDashboardComponent`.

```
import { Component, Input } from '@angular/core';
import { Todo } from '../../../../../models/todo.model';
@Component({
  selector: 'todo-count',
  styleUrls: [],
  templateUrl: 'todo.count.html'
})
export class TodoCountComponent {
  @Input()
  todos: Todo[];
```

```
    constructor() {}  
}
```

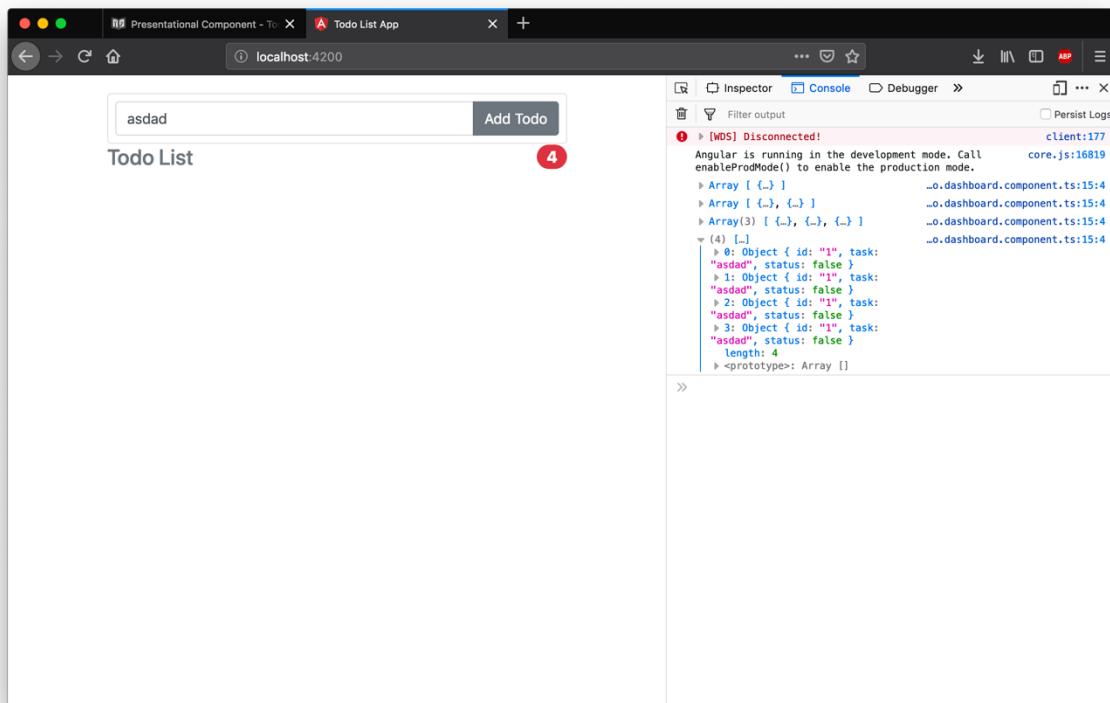
Pada kode program di atas ditambahakan terlebih dahulu import model Todo dan ditambahkan decorator input todo.

Menghitung Todo finished / unfinished di TodoCountComponent

Untuk menghitung jumlah todo dapat dilakukan dengan menambahkan interpolasi untuk menampilkan size dari variabel todos pada component TodoCount.

```
<h4 class="d-flex justify-content-between align-items-center mb-3">
  <span>Todo List</span>
  <span class="badge badge-danger badge-pill">{{ todos.length }}</span>
</h4>
```

Jalankan server dan perhatikan perubahan yang terjadi jika anda menambahkan todo.



Sampai pada langkah ini jika ditambahkan sebuah todo baru data todo akan tersimpan di array `todoList` dan `todo count` component sudah menampilkan jumlah yang sesuai dengan jumlah todo.

CHAPTER 6

TodoItemComponent

Selanjutnya kita akan menyelesaikan component selanjutnya yaitu presentational component TodoItemComponent. Presentational component ini mempunyai tugas menampilkan satu item todo dengan sebuah button yang dapat mengubah status todo dari finished ke unfinished.

Buatlah TodoItemComponent sesuai dengan desain pada chapter sebelumnya berdasarkan tabel dibawah ini

Component	Input	Output / Event Emitter
TodoCountComponent	Todos[]	-
TodoItemComponent	Todo	ToggleFinished
AddTodoComponent	-	AddTodo

TodoItemComponent memiliki satu input Todo dan satu Output ToggleFinished. Untuk menyelesaikannya lakukanlah langkah percobaan berikut ini.

Boilerplate TodoItemComponent

Bukalah file todo.item.component.ts kemudian tambahkan kode program berikut ini.

```
import { Component, Input, Output, EventEmitter } from '@angular/core';
import { Todo } from '../../models/todo.model';
@Component({
  selector: 'todo-item',
  styleUrls: [],
  templateUrl: 'todo.item.html'
})
export class TodoItemComponent {
  constructor() {}
}
```

Pada kode program diatas dilakukan import terhadap component, input, output dan eventemitter hal ini dilakukan karena pada component ini memiliki keduanya baik input maupun output.

Tampilan

Tampilan todo item berisi keterangan todo dan status dari todo tersebut baik finished maupun unfinished, bukalah file todo.item.html kemudian isikan kode program berikut.

```
<li class="list-group-item d-flex justify-content-between lh-condensed">
  <div>
    <h6 class="my-0">Contoh Task</h6>
  </div>
  <span class="badge badge-success badge-pill">
    >Finished</span>
  </div>
</li>
```

Tampilan ini berisi item task dan badge finished, nanti berdasarkan data yang diberikan dari todoDashboard todoltem akan merender satu item berdasarkan data tersebut

Register ke module

Selanjutnya dengan component yang sudah dibuat daftarkan component ini ke module todo.dashboard.module.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { TodoDashboardComponent } from './containers/todo.dashboard.component';
import { AddTodoComponent } from './component/add-todo/add.todo.component';
import { TodoCountComponent } from './component/todo-count/todo.count.component';
import { TodoItemComponent } from './component/todo-item/todo.item.component';

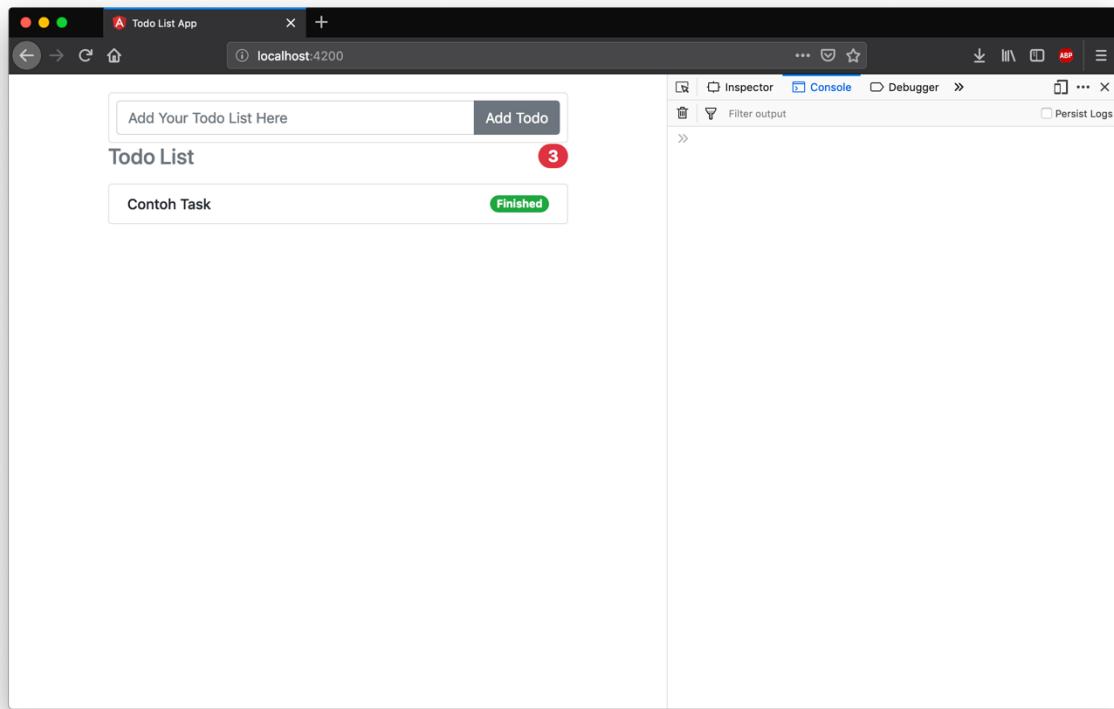
@NgModule({
  declarations: [
    TodoDashboardComponent,
    AddTodoComponent,
    TodoCountComponent,
    TodoItemComponent
  ],
  imports: [CommonModule],
  exports: [TodoDashboardComponent]
})
export class TodoDashboardModule {}
```

Aktifkan selector

Setelah melakukan registrasi component, lanjutan dengan mengaktifkan selector todo-item di todo.dashboard.html.

```
<div class="row">
  <div class="col-md-12 order-md-2 mb-4">
    <add-todo (create)="handleAddTodo($event)"></add-todo>
    <todo-count [todos]="todoList"></todo-count>
    <todo-item></todo-item>
  </div>
</div>
```

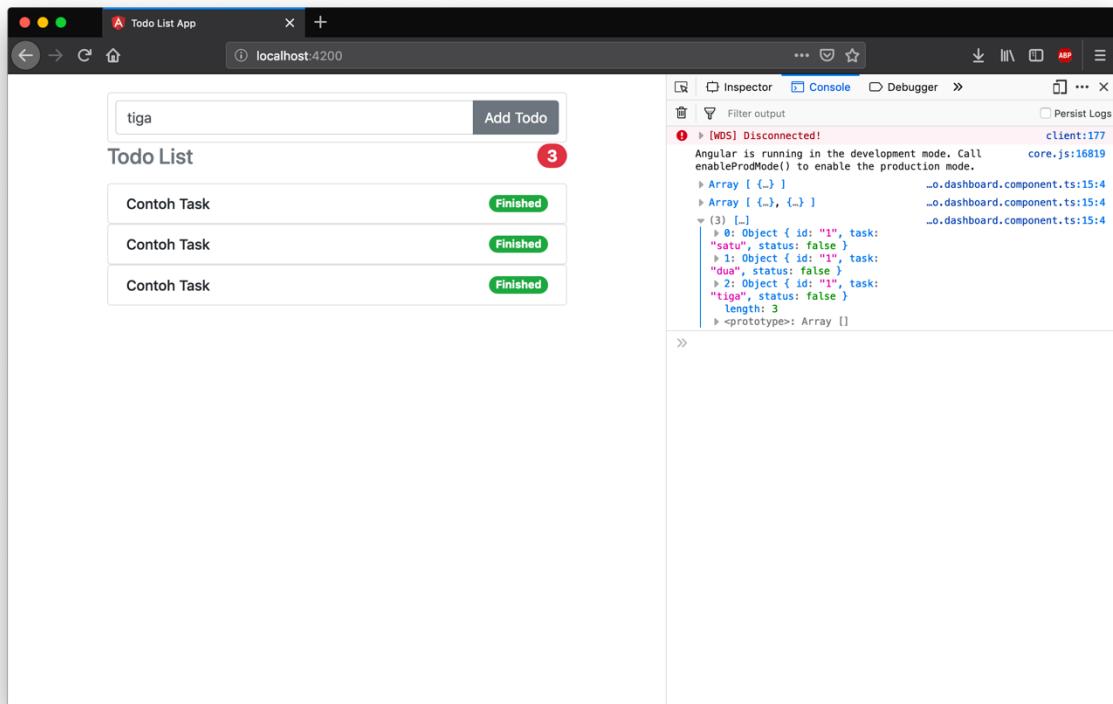
Pada kode program diatas dibuat import todoItemComponent dan ditambahkan ke declarations. Jika dijalankan sekarang server akan memberikan tampilan sebagai berikut.



Looping dan Input Parameter

Karena data dari TodoDashboardComponent berupa list of task maka diperlukan sebuah perulangan untuk menampilkan semua data todo yang ada. Untuk itu ubahlah kode program pada todo.dashboard.html menjadi seperti dibawah ini.

```
<div class="row">
  <div class="col-md-12 order-md-2 mb-4">
    <add-todo (create)="handleAddTodo($event)"></add-todo>
    <todo-count [todos]="todoList"></todo-count>
    <todo-item *ngFor="let todo of todoList"></todo-item>
  </div>
</div>
```



Jika server dijalankan dapat dilihat jumlah todoitem bertambah sesuai dengan jumlah input yang kita berikan namun belum memiliki data yang sesuai dengan input yang kita berikan.

Update Todo Item Untuk terima input

Agar data sesuai dengan input diperlukan cara agar data dari TodoDashboard terhubung ke Todoltem, hal ini dapat dilakukan dengan memberikan input ke Todoltem dengan membuat property binding di file todo.dashboard.html ke selector todo-item.

```

<div class="row">
  <div class="col-md-12 order-md-2 mb-4">
    <add-todo (create)="handleAddTodo($event)"></add-todo>
    <todo-count [todos]="todoList"></todo-count>
    <todo-item *ngFor="let todo of todoList" [item]="todo"></todo-item>
  </div>
</div>

```

Todoltem harus memiliki decorator input agar dapat menerima input dari todo dashboard.

```

import { Component, Input, Output, EventEmitter } from '@angular/core';
import { Todo } from '../../../../../models/todo.model';
@Component({
  selector: 'todo-item',
  styleUrls: [],
  templateUrl: 'todo.item.html'
})
export class TodoItemComponent {
  @Input()

```

```

item: Todo;
constructor() {}
}

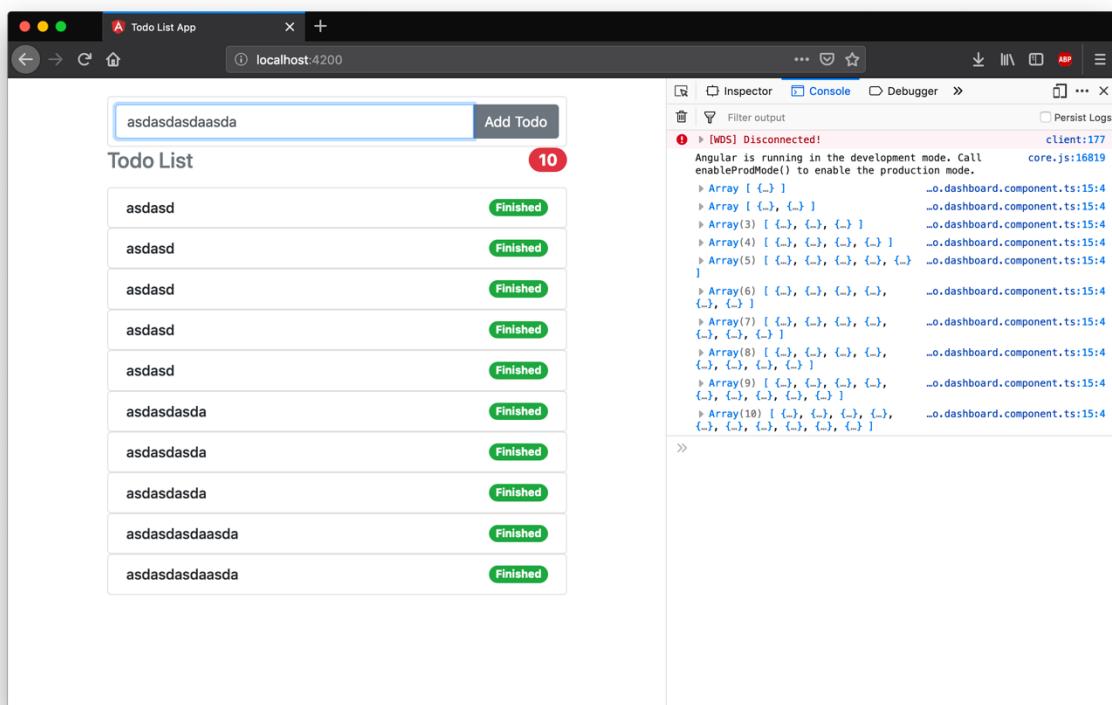
```

Setelah mendapatkan input modifikasi file todo.item.html sehingga dapat memanfaatkan input item dari todoDashboard.

```

<li class="list-group-item d-flex justify-content-between lh-condensed">
  <div>
    <h6 class="my-0">{{ item.task }}</h6>
  </div>
  <span class="badge badge-success badge-pill">Finished</span>
</li>

```

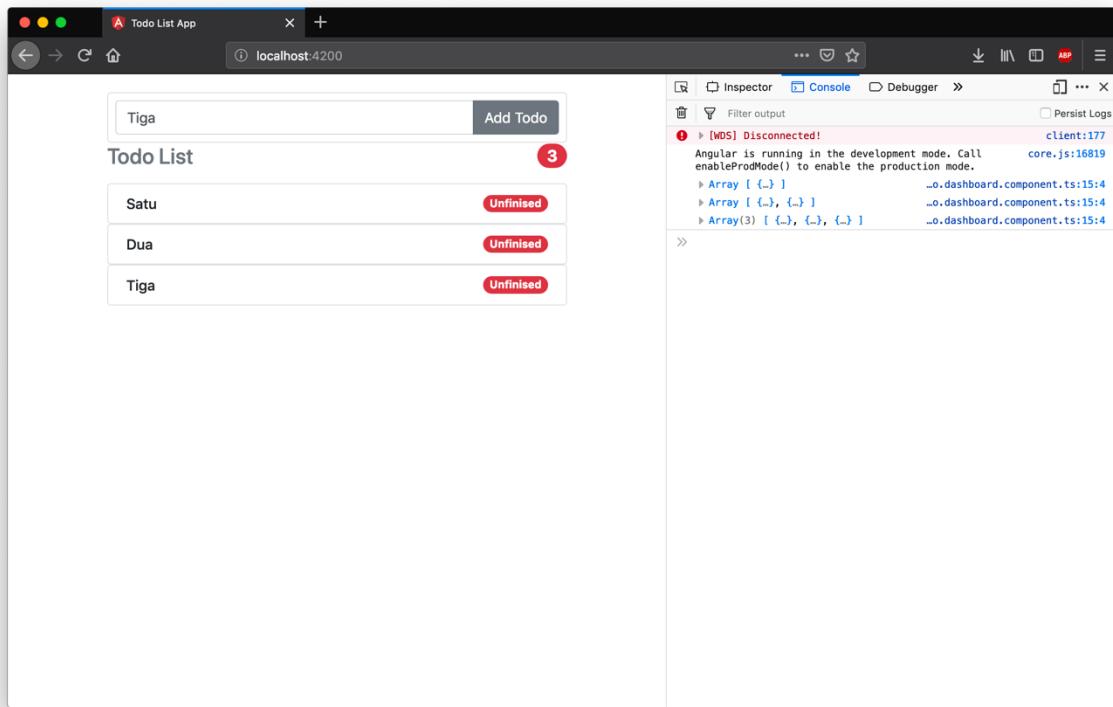


Selanjutnya modifikasi dengan menambahkan template untuk membedakan tampilan todo ketika finished atau unfinished.

```

<li class="list-group-item d-flex justify-content-between lh-condensed">
  <div>
    <h6 class="my-0">{{ item.task }}</h6>
  </div>
  <ng-container
    *ngIf="item.status; then finished; else unfinished">
  </ng-container>
  <ng-template #finished>
    <span class="badge badge-success badge-pill">
      Finished</span>
  </ng-template>
  <ng-template #unfinished>
    <span class="badge badge-danger badge-pill">
      Unfinished</span>
  </ng-template>
</li>

```



Sejauh ini kita sudah berhasil menambahkan todo dan menampilkan statusnya, selanjutnya berikan event click pada todoitem sehingga dapat mengirimkan data ke tododashboard dan mengubah status todo dari unfinished ke finished atau sebaliknya.

Buat Event Click di TodoItem

Untuk membuat event dilakukan dengan menambahkan event binding di file todo.item.html dan membuat handler nya di file todo.item.component.ts

```

<li class="list-group-item d-flex justify-content-between lh-condensed">
  <div>
    <h6 class="my-0">{{ item.task }}</h6>
  </div>
  <ng-container
    *ngIf="item.status; then finished; else unfinished">
  </ng-container>
  <ng-template #finished>
    <span class="badge badge-success badge-pill" (click)="onStatusClick(item)">Finished</span>
  </ng-template>
  <ng-template #unfinished>
    <span class="badge badge-danger badge-pill" (click)="onStatusClick(item)">Unfinished</span>
  </ng-template>
</li>

```

```

import { Component, Input, Output, EventEmitter } from '@angular/core';
import { Todo } from '../../../../../models/todo.model';
@Component({
  selector: 'todo-item',
  styleUrls: [],
  templateUrl: 'todo.item.html'
})

```

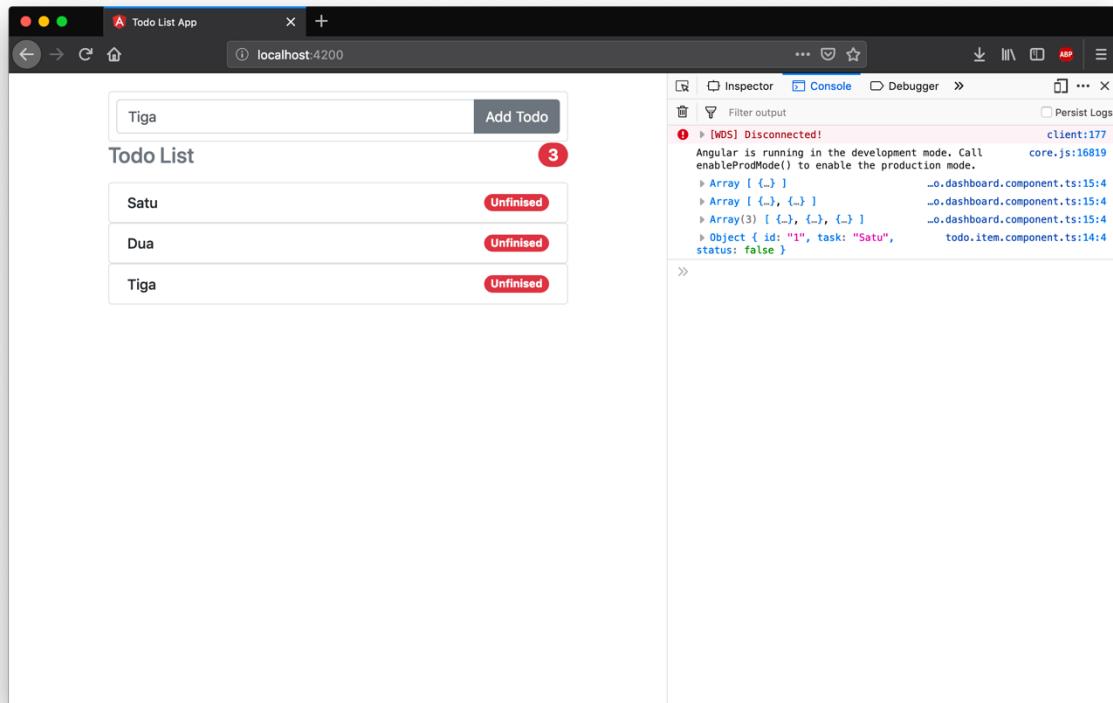
```

export class TodoItemComponent {
  @Input()
  item: Todo;
  constructor() {}

  onStatusClick(todo: Todo) {
    console.log(todo);
  }
}

```

Pada kode program di atas kita melakukan event click sehingga ketika di click akan menampilkan item todo yang di click di console.



Buat Output / Event Emitter dari Todo Item

```

import { Component, Input, Output, EventEmitter } from '@angular/core';
import { Todo } from '../../../../../models/todo.model';
@Component({
  selector: 'todo-item',
  styleUrls: [],
  templateUrl: 'todo.item.html'
})
export class TodoItemComponent {
  @Input()
  item: Todo;

  @Output()
  status: EventEmitter<any> = new EventEmitter();

  constructor() {}

  onStatusClick(todo: Todo) {
    this.status.emit(todo);
  }
}

```

Pada kode program di atas ditambahkan output bernama status yang memiliki tipe data EventEmitter, kemudian pada fungsi onStatusClick output ini di emmit melalui event emitter.

Buat Event Binding di Selector TodoDashboard

Agar output yang di emmit oleh TodoItem dapat diterima oleh TodoDashboard maka pada todo.dashboard.html perlu ditambahkan event binding pada selector todo-item.

```
<div class="row">
  <div class="col-md-12 order-md-2 mb-4">
    <add-todo (create)="handleAddTodo($event)"></add-todo>
    <todo-count [todos]="todolist"></todo-count>
    <todo-item
      *ngFor="let todo of todoList"
      [item]="todo"
      (status)="handleToggleFinished($event)"
    ></todo-item>
  </div>
</div>
```

Pada kode program diatas ditambahkan event binding status dan membutuhkan handlerToggleFinished oleh karena itu tambahkan fungsi ini di todo.dashboard.ts

```
import { Component } from '@angular/core';

import { Todo } from '../models/todo.model';
@Component({
  selector: 'todo-dashboard',
  styleUrls: ['todo.dashboard.css'],
  templateUrl: 'todo.dashboard.html'
})
export class TodoDashboardComponent {
  todoList: Todo[] = [];

  handleAddTodo(task: string) {
    const itemTodo = new Todo('1', task, false);
    this.todoList.push(itemTodo);
    console.log(this.todoList);
    return false;
  }

  handleToggleFinished(todo: Todo) {
    console.log(this.todoList);
    return (todo.status = !todo.status);
  }
}
```

Jika server dijalankan anda sudah dapat mengubah status todo dari unfinished ke finished dengan mengklik tombol unfinished.

Todo List

Empat

Satu

Dua

Tiga

Empat

Add Todo

4

[WDS] Disconnected!

Angular is running in the development mode. Call enableProdMode() to enable the production mode.

core.js:16819

client:177

...o.dashboard.component.ts:15:4

...o.dashboard.component.ts:15:4

...o.dashboard.component.ts:15:4

...o.dashboard.component.ts:15:4

...o.dashboard.component.ts:20:4

...o.dashboard.component.ts:20:4

