Angular
Developer workshop

- i4.0 IT Team

- 2020/9/14

Agenda

- Develop Tools
- New Project
- Angular Structure
- Component Introduction
- Data Binding
- Structural Directive
- Template Variables
- Component Communication
- Form
- Router and Navigator
- Directive
- Service
- Pipe
- Guard
- Module
- Library Usage
- Atomic Design

Develop Tools

Node.js	NPM	Angular CLI
VS Code	Will 保哥套件	Cmd Tool

Develop Tools

Angular CLI 優點

1. 量身打造
2. 最好的Starter模板
3. 自動產生目錄結構及開發所需的檔案
4. 支援程式碼產生
5. 包含單元、整合測試
6. 程式碼最佳化
7. 統一開發體驗

Develop Tools

Angular CLI 常用指令

- 新建專案

```
> ng new <專案名稱>
```

- 建立Component、Directive、Service、Pipe、Guard等Schematic

```
> ng g <schematic> <schematic名稱>
```

- 啟動Server

```
> ng serve
```

- 匯出專案

```
> ng build
```

New Project

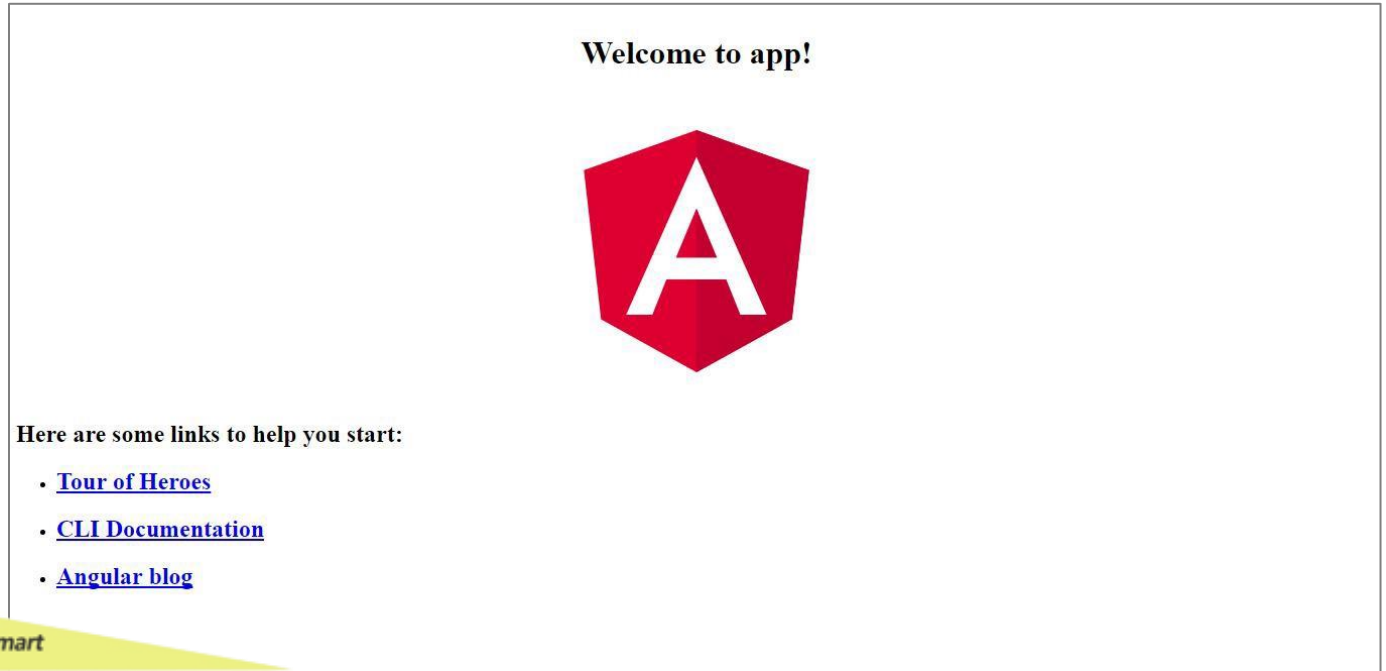
- 安裝Node.js
- 安裝Angular CLI

```
> npm install –g @angular/cli
> ng new <專案名稱>
> cd <專案名稱>
> ng serve
```

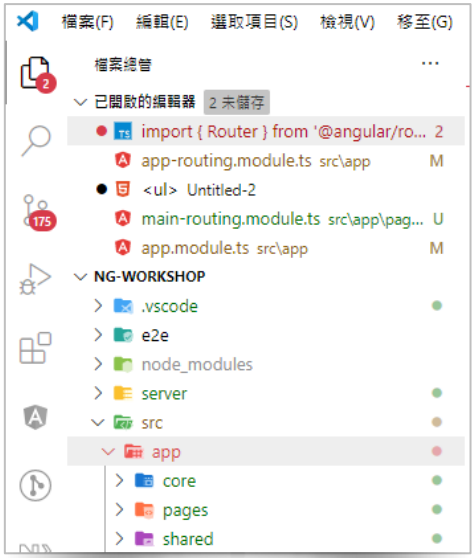
- 開啟Browser http://localhost:4200

New Project

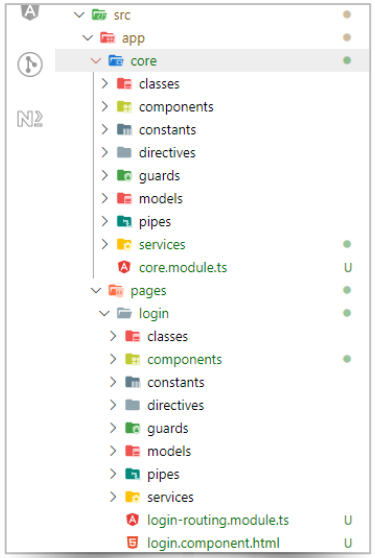
- 執行後畫面

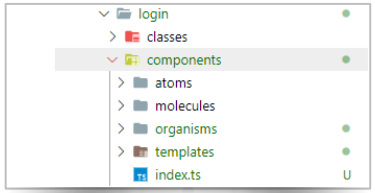


Project Architecture



展開





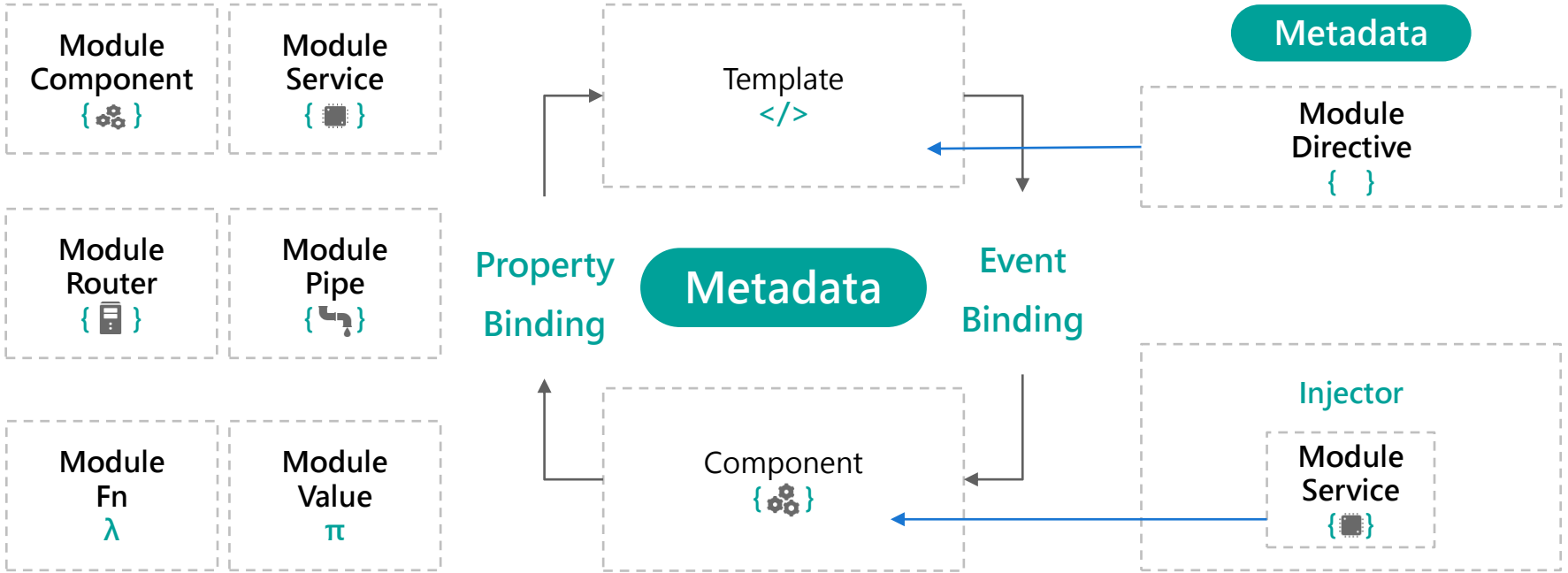
- **core**: 共享單例資源
- **shared** : 共享多個實例資源
- **pages**: 依照頁面劃分 module

- **classes**: 供該Module使用的類別
- **models**: 供該Module使用的介面或資料模型
- **constants** :供該Module使用的常數資源
- **components**: 供該Module使用的元件
- **services**: 供該Module使用的服務
- **directive**: 供該Module使用的元件指令
- **pipes**: 供該Module使用的管道
- **guards**: 供該Module使用的路由守衛

使用Atomic Design的方式分類Component中的目錄，

待Atomic Design章節再進行詳細的分類基準

Angular Structure



Component Introduction

何謂Component？

- 最小單位
- 包裝HTML、CSS(LESS or SCSS)及Typescript
- 可重複使用
- 處理View的操作
- 反應物件狀態

Component Introduction

元件化設計

- 將畫面拆分成多個可重複使用的component
- 主要用來呈現資料
- 盡可能不處理運算方面的邏輯；僅處理View的呈現邏輯
- 所有的元件都由Component組合而成

Component Introduction

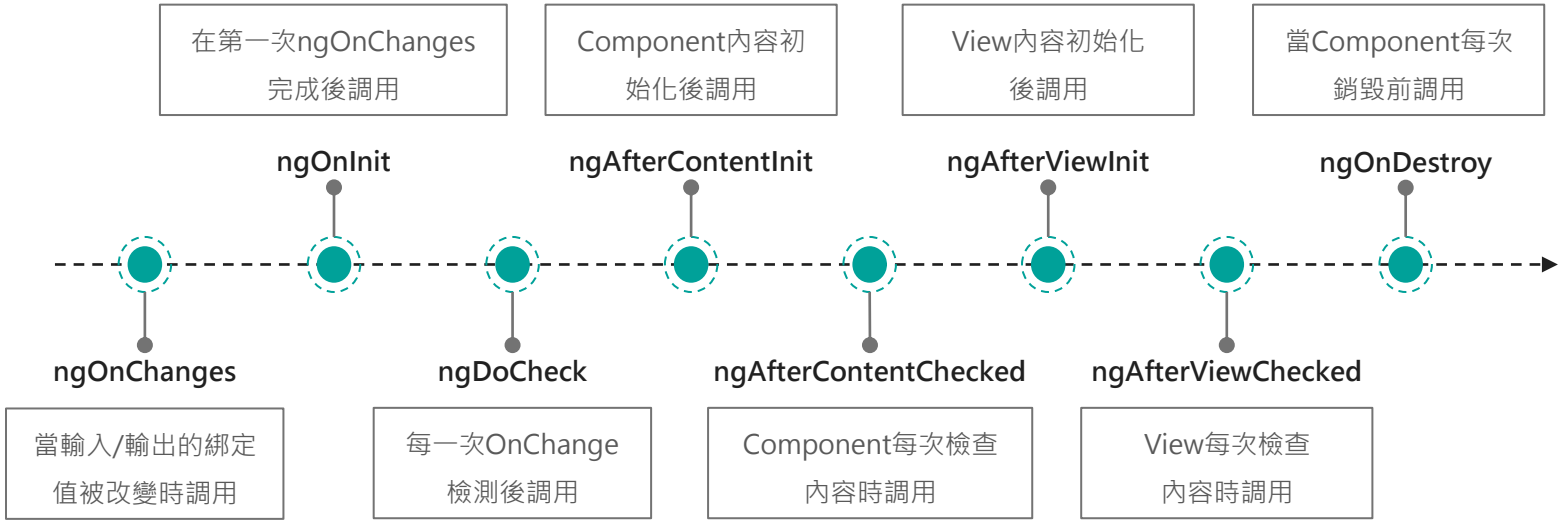
建立Component

> ng g c <component名稱>

```
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-content-card',
5   templateUrl: './content-card.component.html',
6   styleUrls: ['./content-card.component.less']
7 })
8 export class ContentCardComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit() {
13   }
14
```

Component Introduction

Component 生命週期



Data Binding

資料繫結

- 將資料呈現在畫面上
- 根據條件呈現
- 接收使用者輸入的資料
- 提供四種綁定方式插值、屬性綁定、事件綁定及雙向綁定

Data Binding

- 插值(Interpolation)

```
1 <span>{{ name }}</span>
```

- 屬性綁定(Property Binding)

```
1 <img [src]="imageSource" [attr.data-source]="imageSource" />
```

Data Binding

- 事件綁定(Event Binding)

```
1 <button  (click)=“onClickedButton(‘clicked’)”>按鈕</button>
```

- 雙向綁定(Two-way Binding)

```
1 <input type="text" [(ngModel)]="name" />
```

Structural Directive

- *ngIf

```
1 <div *ngIf="textSHow">Show Me ?</div>
```

- *ngFor

```
1 <ul>
2   <li *ngFor="let person of people">{{ person.name }}</li>
3 </ul>
```

Structural Directive

- ngSwitch

```
<p [ngSwitch]="condition">
  <span *ngSwitchCase="'a'">a</span>
  <span *ngSwitchCase="'b'">b</span>
  <span *ngSwitchCase="'c'">c</span>
  <span *ngSwitchDefault>other</span>
</p>
```

- 注意：每個元素僅能使用一個結構指令

Template Variables

- 直接定義變數在View(HTML)當中

```html
<input type="text" #demo>
<input type="button" (click)="send(demo)">
```

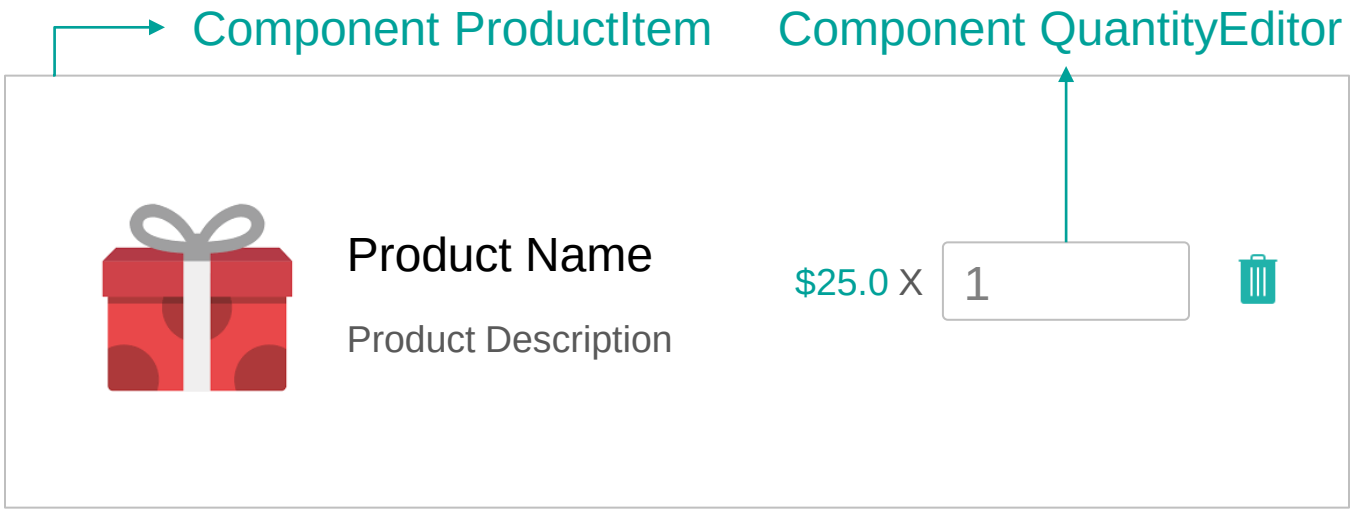
```typescript
export class AppComponent implements OnInit {

  send(demo: HTMLInputElement): void {
    // demo 目前為取得該元素
    console.log(demo);

    // demo.value 可取得該 input 的數值
    console.log(demo.value);
  }
```

Component Communication

- 透過組合Component完成功能
- 彼此使用Property和Event溝通
- 最外層的Component才有資料
  - Controller View
  - Child Component 負責顯示資料和觸發事件
- 讓元件更容易重複使用

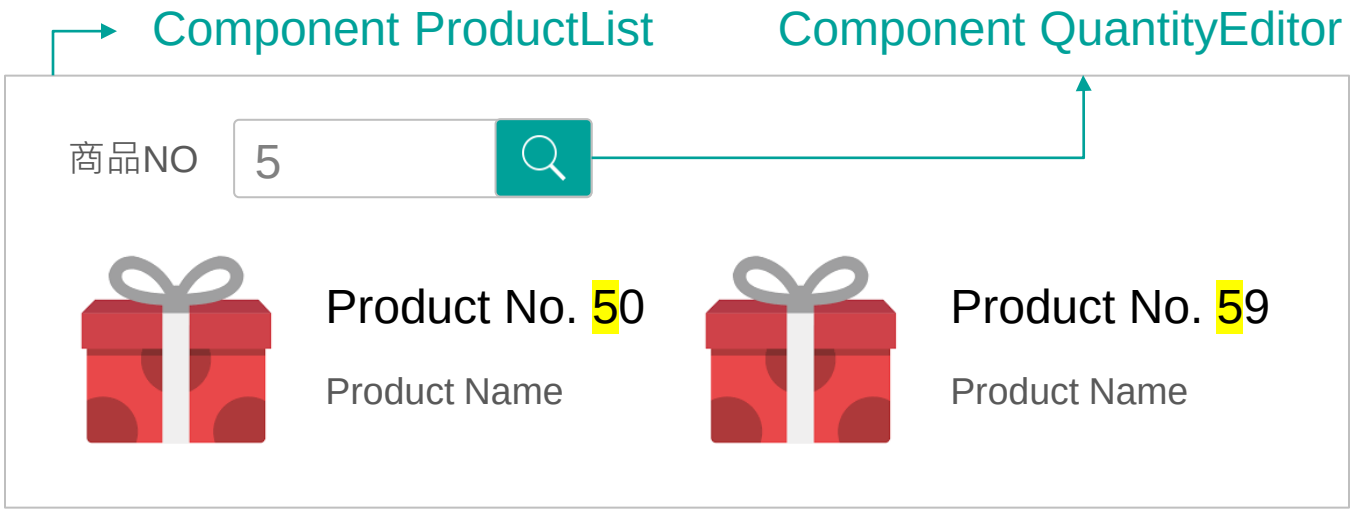
Component Communication

- 提高重用性示例，商品數量變更時，Ajax回傳該商品購買數量至server存檔



Component Communication

- 若QuantityEditor不再負責商品數量，而是改為檢索商品編號時，當輸入編號後，Ajax給server取得對應商品

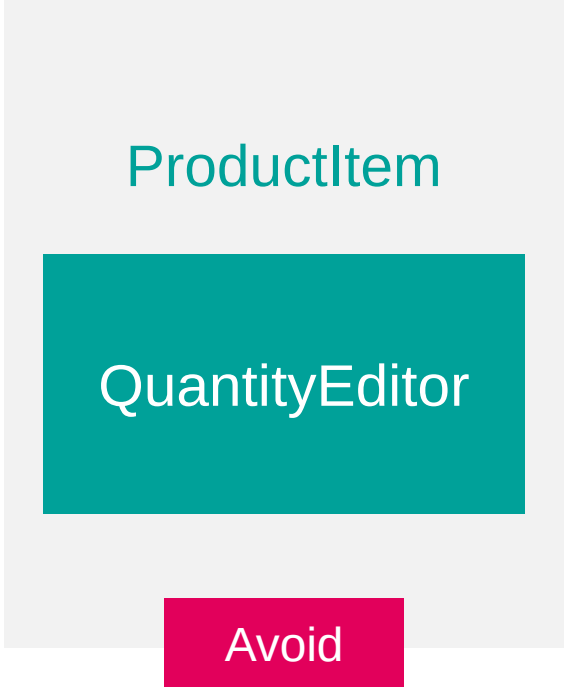


Component Communication

How to design ?

- Ajax商品資訊的方法要寫在 ProductItem? QuantityEditor?還是 ProductList?

Component Communication



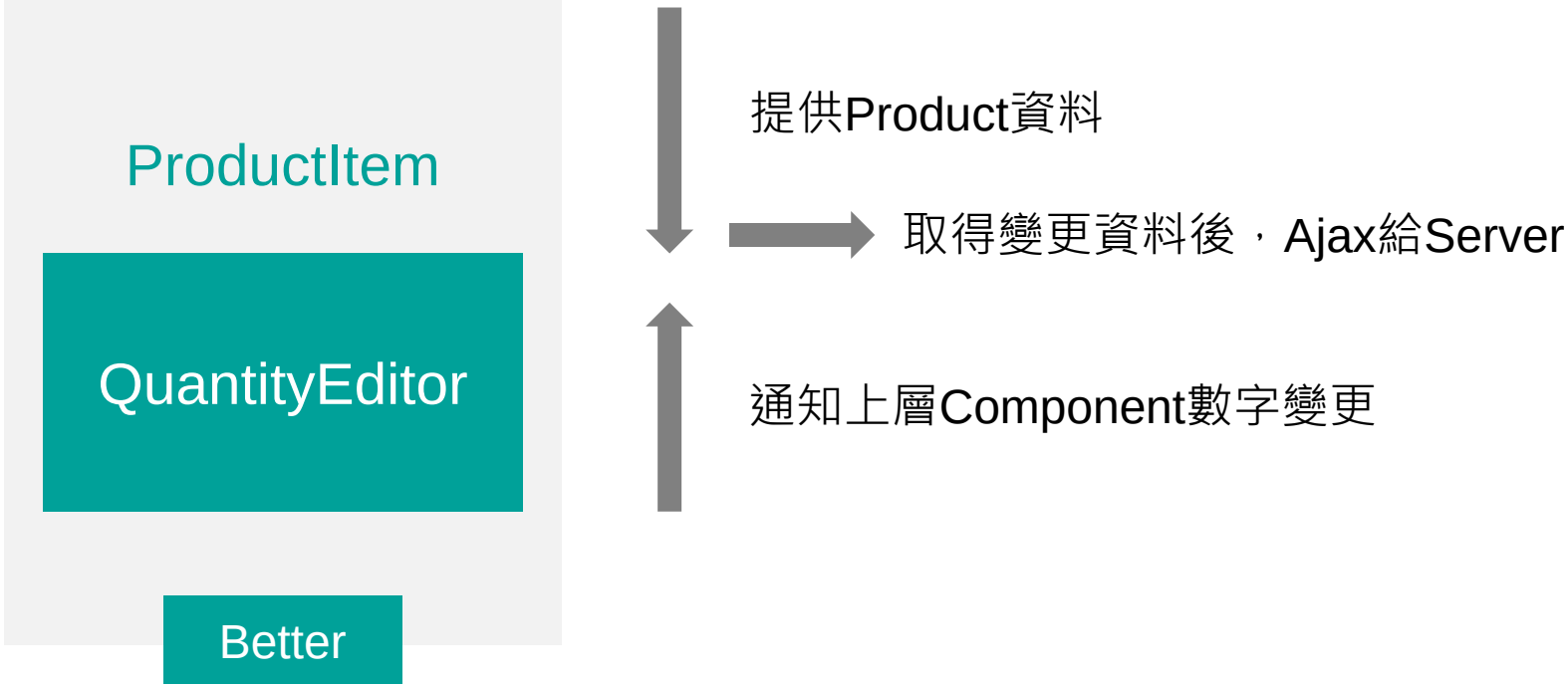
提供Product資料

變更時，Ajax給Server

QuantityEditor受特定component限制

不易重複使用

Component Communication



Component Communication

這樣設計帶來的好處

1. 提高QuantityEditor的重用性
2. 避免QuantityEditor會因需要多附屬在某個Component下，而在衍生更多的方法
3. 邏輯單純，讓每個Component都是單一職責(SRP)

Component Communication

- 資料傳遞方法：@Input() 【屬性傳遞，向下】

```
<app-quantity-editor [quantity]="5"></app-quantity-editor>
```

```
import { ..., Input } from '@angular/core';

...
export class QuantityEditorComponent {

  @Input() quantity: number;

  constructor() { }

}
```

Component Communication

- 資料傳遞方法：@Output() 【事件傳遞，向上】

```
<app-status-bar (status)="getStatus($event)"></app-status-bar>
```

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

...
export class StatusBarComponent {

  @Output() status = new EventEmitter<boolean>;

  constructor() {}

  clickStatusButton() {
    this.status.emit(true);
  }
```

Form

Template Driven Form

- 適合時做簡易的表單
- 表單所需的元件都宣告在HTML中
- Import FormsModule

```
1 import { FormsModule } from '@angular/forms';
```

Form

Template Driven Form範例

```
<!-- 利用 #var 建立變數並給予ngForm，內建表單驗證的功能 -->
<form #myForm="ngForm">
  <label>帳號</label>
  <!-- touched 為第一次 focus 在輸入欄位後會轉為 true -->
  <span *ngIf="account.touched && account.errors">
    <code *ngIf="account.errors.required">必填</code>
  </span>

  <!-- 加入 ngModel 進行資料綁定，並給上 name 屬性定義 -->
  <!-- required 為驗證項目 -->
  <input type="text" #account="ngModel" ngModel name="account" required class="form-control">

  <input type="submit" [disabled]="!myForm.valid" class="btn btn-success">
</form>
```

Form

Reactive Form

- 適合時做複雜的表單
- 使用屬性綁定對表單進行宣告，表單的宣告是建立formGroup屬性，而針對控制項，則是建立formControlName
- Import ReactiveFormsModule

```
1 import { ReactiveFormsModule } from '@angular/forms';
```

Form

Reactive Form範例

```
1 import { FormGroup, FormControl, Validators } from '@angular/forms';
2
3 ...
4 export class SignupComponent {
5
6   signupForm: FormGroup;
7
8   constructor() {
9     this.signupForm = this.createFormGroup();
10  }
11
12  createFormGroup() {
13    return new FormGroup({
14      personalData: new FormGroup({
15        email: new FormControl('', Validators.required),
16        mobile: new FormControl('', Validators.pattern(/^\+[1-9]{1}[0-9]{3,14}$/g)),
17        country: new FormControl()
18      }),
19      requestType: new FormControl(),
20      text: new FormControl()
21    });
```

Form

Reactive Form範例

```
<form [formGroup]="signupForm" (ngSubmit)="onSubmit()">
  <div formGroupName="personalData">
    <input formControlName="email">
    <input formControlName="mobile">
    <select formControlName="country">
      <option *ngFor="let country of countries" [value]="country">{{ country }}</option>
    </select>
  </div>
  <select formControlName="requestType">
    <option *ngFor="let requestType of requestTypes" [value]="requestType">{{ requestType }}</option>
  </select>
  <input formControlName="text">
  <button type="submit" [disabled]="!signupForm.valid">Sign Up</button>
</form>
```

Form

Validators List

1. min：限制最小值
2. max：限制最大值
3. required：必填
4. requiredTrue：必須為True
5. email：符合email格式
7. minLength：最短長度
8. maxLength：最常長度
9. Pattern：使用RegExp自訂驗證格式
10. nullValidator：不為NULL
11. compose：將多個驗證器合成一個

Router and Navigate

建立路由Module

- 在創建專案時，就必須先將routing的功能加入其中

```
> ng new <專案名稱> --routing
```

- 在angular cli 7的版本中，會有詢問提示

```
> ng new <專案名稱>
> Would you like to add Angular routing (y/N) y
```

Router and Navigate

路由Module

- app-routing.module.ts

```
1 import { NgModule } from '@angular/core';
2 import { Routes, RouterModule } from '@angular/router';
3
4 const routes: Routes = [];
5
6 @NgModule({
7   imports: [RouterModule.forRoot(routes)],
8   exports: [RouterModule]
9 })
10 export class AppRoutingModule { }
```

Router and Navigate

路由Parameters介紹

- path：路由的路徑
- redirectTo：重新導向哪個路徑
- component：路由中會導入哪個component
- pathMatch：設置路由的匹配規則，’full’表示完全匹配；’prefix’表示只匹配前綴，當path值為home時，/home、/home/123, 都能匹配到該路由。
- children：子路由的設定

Router and Navigate

路由範例

- path為''時，表示路徑為空時匹配該路由；值為** 時所有路徑都匹配不到時匹配該路由，**路徑有順序性**，**符合就導向該路由**。

```
const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent, pathMatch: 'full' },
  {
    path: 'dashboard',
    component: DashboardComponent,
    children: [
      { path: 'profile/:id', component: AccountProfileComponent, pathMatch: 'full' }
    ]
  },
  { path: '**', component: NotFoundComponent, pathMatch: 'full' }
];
```

Router and Navigate

路由Navigate範例

- 在HTML中使用

```
<ul>
  <li>
    <a [routerLink]="['/login']"
       routerLinkActive="item-active">
      Login
    </a>
  </li>
</ul>

<router-outlet></router-outlet>
```

- 在Typescript中使用

```
import { Router } from '@angular/router';

...
export class LoginComponent {

  constructor(private router: Router) { }

  login(): void {
    this.router.navigate(['/dashboard']);
  }

}
```

Router and Navigate

取得路由參數

```
1 import { ActivatedRoute } from '@angular/router';
2
3 ...
4 export class AccountProfileComponent implements OnInit {
5
6   private accountID: string;
7
8   constructor(private router: ActivatedRoute) { }
9
10  ngOnInit() {
11    this.router.queryParams
12      .toPromise()
13      .then(params => {
14        this.accountID = params.id;
15      })
16      .catch(error => {
17        console.error(error);
18      });
```

Router and Navigate

路由Navigate帶參數範例

- 在HTML中使用

```
<ul>
  <li>
    <a [routerLink]="['/dashboard/profile',
                      { id: 'accountid001' }]"
      routerLinkActive="item-active">
      Login
    </a>
  </li>
</ul>

<router-outlet></router-outlet>
```

- 在Typescript中使用

```
import { Router } from '@angular/router';

...
export class LoginComponent {

  constructor(private router: Router) { }

  login(): void {
    this.router.navigate(['/dashboard/profile'], {
      queryParams: { id: 'accountid001' }
    });
  }

}
```

Directive

建立屬性指令

> ng g d <指令名稱>

```
1 import { Directive, Input, ElementRef  } from '@angular/core';
2
3 @Directive({
4   selector: '[appHightlight]'
5 })
6 export class HightlightDirective {
7
8   @Input() color: string;
9
10  constructor(private element: ElementRef ) {
11    this.element.nativeElement.style.backgroundColor = this.color;
12  }
13
```

Directive

建立屬性指令

```
1 <p appHightlight [color]="'red'">Hightlight Directive</p>
```

Service

建立Service

- Service負責處理運算邏輯；Component處理UI邏輯
- Service為Singleton【單例模式】，且使用DI【相依注入】

> ng g s <service名稱>

```
import { MqttService } from './mqtt.service';

@NgModule({
  ...
  providers: [
    MqttService,
    ...
  ]
})
export class AppModule { }
```

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

@Injectable({
  providedIn: 'root'
})
export class MqttService {

  constructor() { }

}
```

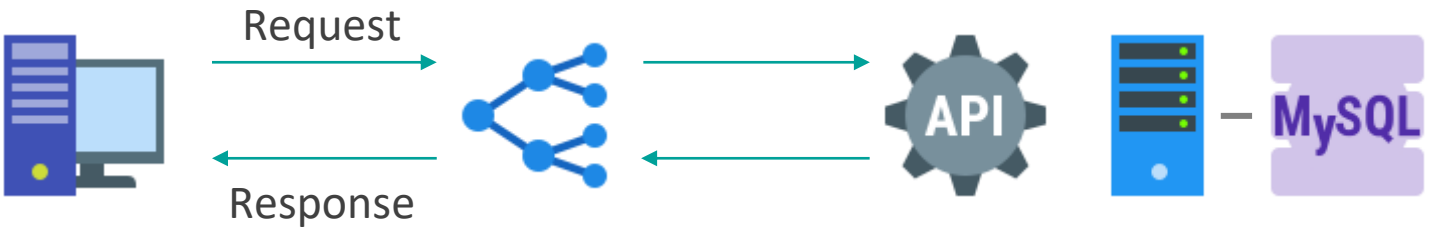
Service

在Component中使用Service

```
1 import { Component, OnInit } from '@angular/core';
2 import { MqttService } from '../mqtt.service';
3
4 @Component({
5   selector: 'app-content-card',
6   templateUrl: './content-card.component.html',
7   styleUrls: ['./content-card.component.less']
8 })
9 export class ContentCardComponent implements OnInit {
10
11   constructor(private mqttService: MqttService) { }
12
13   ngOnInit() {
14   }
15
16 }
```

Service

使用HttpClient



app.module.ts

```
1 import { HttpClientModule } from '@angular/common/http';
```

Service

為何要將Service與HttpClient放在一起介紹？

- 將HttpClient寫在Service中有以下幾個用意

1. Service處理運算邏輯，而HttpClient可以透過API取得資料
2. 從外部取得的資料，有時需要透過數個API才能蒐集齊全、有時需要修改資料的格式或將不需要的資料進行filter，若將這些功能寫在Component中，會讓邏輯更佳複雜且不易更動

Service

HttpClient

- 從Angular 4.3版以後，新的網路傳輸HttpClient在Angular中被引入
- 從Angular 5版以後，捨棄Http，而是改採用HttpClient

HttpClient V.S. Http

1. 若API中使用JSON格式來進行回傳，則自動轉換成JSON的格式，不需要再使用JSON.parse(<回傳資料>);
2. Header的簡化語法

Service

HttpClient範例

```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4
5 import { Hero } from './models/hero';
6
7 @Injectable({
8   providedIn: 'root'
9 })
10 export class HerosDataService {
11
12   private apiUrl: string;
13
14   constructor(private http: HttpClient) {
15     this.apiUrl = 'localhost:30000/api/hero';
16   }
17
18   fetchHerosData(): Observable<Hero> {
19     return this.http.get<Hero>(this.apiUrl);
```

Service

方法一 Subscription

```
1 import { HerosDataService } from './../heros-data.service';
2 import { Component, OnInit } from '@angular/core';
3
4 import { Hero } from './../models/hero';
5
6 @Component({
7   selector: 'app-content-card',
8   templateUrl: './content-card.component.html',
9   styleUrls: ['./content-card.component.less']
10 })
11 export class ContentCardComponent implements OnInit {
12
13   constructor(private herosDataService: HerosDataService) { }
14
15   ngOnInit() {
16     this.herosDataService.fetchHerosData()
17       .subscribe(result => {
18         // TODO:
19       }, error => {
20         console.error(error);
21       });
22   }
23
24 }
```

方法二 Promise

```
1 import { HerosDataService } from './../heros-data.service';
2 import { Component, OnInit } from '@angular/core';
3
4 import { Hero } from './../models/hero';
5
6 @Component({
7   selector: 'app-content-card',
8   templateUrl: './content-card.component.html',
9   styleUrls: ['./content-card.component.less']
10 })
11 export class ContentCardComponent implements OnInit {
12
13   constructor(private herosDataService: HerosDataService) { }
14
15   ngOnInit() {
16     this.herosDataService.fetchHerosData()
17       .toPromise()
18       .then(result => {
19         // TODO:
20       })
21       .catch(error => {
22         console.error(error);
23       });
24   }
25
26 }
```

Service

使用Subscription須注意

- 當Component銷毀時，需做unsubscribe的動作，釋放資源

```
export class ContentCardComponent implements OnInit, OnDestroy {

  private herosDataSubscription: Subscription;

  constructor(private herosDataService: HerosDataService) { }

  ngOnInit() {
    this.herosDataSubscription = this.herosDataService.fetchHerosData()
      .subscribe(result => {
        // TODO:
      }, error => {
        console.error(error);
      });
  }

  ngOnDestroy() {
    this.herosDataSubscription.unsubscribe();
  }
```

Service

Subject

- 內部有一份 observer 的清單，並在接收到值時遍歷這份清單並送出值

BehaviorSubject 【實作上較好運用】

- 新的訂閱產生時，希望 Subject 能立即給出最新的值，而不是沒有回應

ReplaySubject 【實作上較好運用】

- 新的訂閱產生時，希望Subject能重新發送前幾個元素

AsyncSubject

- Subject complete時，發送最後一元素給訂閱者

Service

Subject示例【以ReplaySubject為例】

- heros-data.service.ts

```
export class HerosDataService {

  private heroSubject = new ReplaySubject<Hero>(1);

  constructor(private http: HttpClient) {
    this.heroSubject.next({
      name: 'Batman',
      age: 35,
      ability: 'genius talent'
    });
  }

  listenHeroProfile(): Observable<Hero> {
    return this.heroSubject.asObservable();
  }
```

Service

Subject示例【以ReplaySubject為例】

- heros-profile.component.ts

```
1  import { HerosDataService } from '../heros-data.service';
2
3  ...
4  export class HerosProfileComponent implements OnInit {
5
6    constructor(private herosDataService: HerosDataService) { }
7
8    ngOnInit() {
9      this.herosDataService.listenHeroProfile()
10       .subscribe(herosProfile => {
11         // TODO:
12       }, error => {
13         console.error(error);
14       });
15   }
```

Pipe

Built-in Pipe範例

- 更多內建Pipe可參考 https://angular.io/api?type=pipe

```
<!-- Decimal Pipe -->
<!-- output '002.71828' -->
<p>e (3.1-5): {{ e | number:'3.1-5' }}</p>

<!-- Currency Pipe -->
<!-- output '$0.26' -->
<p>A: {{ a | currency }}</p>

<!-- Date Pipe -->
<!-- output '(6/15/15, 9:03 AM)' -->
<p>The time is {{ today | date:'short' }}</p>
```

Pipe

建立Custom Pipe

> ng g p <pipe名稱>

```
1 import { Pipe, PipeTransform } from '@angular/core';
2
3 @Pipe({
4   name: 'todoDone'
5 })
6 export class TodoDonePipe implements PipeTransform {
7
8   transform(value: any, args?: any): any {
9     return null;
10  }
11
12 }
```

Pipe

Custom Pipe範例【Todo Items List】

- 利用自訂的Pipe，顯示代辦事項的完成狀態

Pipe

Custom Pipe範例【Todo Items List】

```
1 import { Pipe, PipeTransform } from '@angular/core';
2
3 @Pipe({
4   name: 'todoDone'
5 })
6 export class TodoDonePipe implements PipeTransform {
7
8   transform(done: boolean, completeDate: string): any {
9     if (done) {
10      return `(已完成) 完成時間：${completeDate}`;
11    } else {
12      return '(未完成)';
13    }
14  }
```

Pipe

Custom Pipe範例【Todo Items List】

```
1 <ul>
2   <li *ngFor="let todo in todoList">
3     <app-checkbox [done]="todo.done"></app-checkbox>
4     <p>{{ todo.item }}</p> |
5     <span (click)="deleteItem(todo.id)">刪除</span>
6     <span>
7       {{ todo.done | todoDone: todo.completeDate }}
8     </span>
9   </li>
10 </ul>
```

Guard

建立Guard

> ng generate g <guard名稱>

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from '@angular/router';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {

  constructor(private router: Router) {}

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): boolean {
    if (!User.login) {
      this.router.navigate(['/login']);
      return false;
    }

    return true;
```

Guard

建立Guard

- 在app-routing.module.ts中加入guard

```
import { AuthGuard } from './auth.guard';

...

const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent, pathMatch: 'full' },
  { path: 'profile', component: AccountProfileComponent, pathMatch: 'full', canActivate: [AuthGuard] },
  { path: '**', component: NotFoundComponent, pathMatch: 'full' }
];

...
```

Module

建立Module

- 透過模組化的機制，管理所有components

> ng g m <module名稱>

```
1 import { NgModule } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { ReferenceComponent } from './reference.component';
4
5 @NgModule({
6   imports: [
7     CommonModule
8   ],
9   declarations: []
10 })
11 export class ReferenceModule { }
```

Module

Module Meta Data

- imports： 在這個module下，需要匯入的module，提供其中的components、directives和pipes使用。
- declarations： 屬於這module的components、directives和pipes。
- exports：公開給外部使用的module中的類別。
- providers：提供的services，讓應用程式中所有的組件都可以使用。

Library Usage

i18n：Template內建的多語系支援

- 先使用預設的語言做開發，

```
1 <h1>Hello i18n!</h1>
```

- 增加i18n的標記

```
1 <h1 i18n>Hello i18n!</h1>
```

Library Usage

i18n： Template內建的多語系支援

- 利用CLI產生messages.xlf

```
> ng xi18n
```

```
<trans-unit id="myId" datatype="html">
  <source>Hello i18n!</source>
  <target state="new">Bonjour i18n!</target>
</trans-unit>
```

Library Usage

i18n： Template內建的多語系支援

- 利用CLI產生messages.xlf

```
> ng serve --aot --i18nFile=src/locale/messages.fr.xlf
  --i18nFormat=xlf --locale=fr
```

- 也可自行加上翻譯字串的ID

```
1 <h1 i18n="@@myId">Hello i18n!</h1>
```

Library Usage

i18n：使用 ngx-translate Library

- 安裝 ngx-translate

```
> npm install @ngx-translate/core --save
> npm install @ngx-translate/http-loader --save
```

- 參考來源

  https://github.com/ngx-translate/core

Library Usage

i18n：使用 ngx-translate Library

- Import ngx-translate Module

```
import {TranslateService} from '@ngx-translate/core';

export function createTranslateLoader(http: HttpClient) {
  return new TranslateHttpLoader(http, './assets/i18n/', '.json');
}

@NgModule({
  imports: [
    ...,
    TranslateModule.forRoot({
      loader: {
        provide: TranslateLoader,
        useFactory: (createTranslateLoader),
        deps: [HttpClient]
      }
   })
  ],
  bootstrap: [AppComponent]
```

Library Usage

i18n：使用 ngx-translate Library

- 翻譯的Mapping檔放置assets/i18n/<翻譯語系名稱>.json
- 初始化TranslateService

```
1 import { TranslateService } from '@ngx-translate/core';
2
3 ...
4 export class AppComponent {
5
6   constructor(translate: TranslateService) {
7     translate.setDefaultLang('en');
8
9     translate.use('en');
10  }
11
```

Library Usage

i18n：使用 ngx-translate Library

- 定義mapping的json檔

assets/i18n/en/json

```
{
  "HelloTrans": "Implement i18n!"
}
```

assets/i18n/en/json

```
{
  "HelloTrans": "實作多國語系!"
}
```

Library Usage

i18n：使用 ngx-translate Library

- 使用Pipe或Service

```
<div>{{ 'HelloTrans' | translate }}</div>
```

```
this.translate.get('HelloTrans').subscribe((val: string) => {
    alert(val);
});
```

Library Usage

使用 Echarts

- 安装echarts

```
> npm install echarts --save
> npm install @types/echarts --save
```

- 或者安装ngx-echarts

```
> npm install echarts -S
> npm install ngx-echarts -S
> npm install @types/echarts -D
```

Library Usage

使用 Echarts

- 建立echarts容器

```
1 <div id="line" style="width: 600px;height:400px;"></div>
```

```
1  import * as echarts from 'echarts';
2
3  ...
4  export class AppComponent implements OnInit {
5
6    ...
7
8    initEchart() {
9      this.lineChart = echarts.init(<HTMLDivElement>document.getElementById('charts'));
10     const option = {
11       // 略
12     };
13     lineChart.setOption(option);
14   }
```

Library Usage

使用 Echarts

- Echarts javascript library

```
1 <div id="line" style="width: 600px;height:400px;"></div>
```

```
1  import * as echarts from 'echarts';
2
3  ...
4  export class AppComponent implements OnInit {
5
6    ...
7
8    initEchart() {
9      this.lineChart = echarts.init(<HTMLDivElement>document.getElementById('charts'));
10     const option = {
11       // 略
12     };
13     lineChart.setOption(option);
14   }
```

Library Usage

使用 Echarts

- ngx-echarts module

```
1 import { NgxEchartsModule } from 'ngx-echarts';
```

```
1 <div echarts [options]="chartOption" class="demo-chart"></div>
```

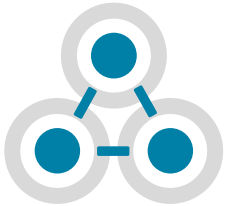
```
1 import { EChartOption } from 'echarts';
2
3 ...
4
5 chartOption: EChartOption = {
6   ...
7 }
```

Atomic Design

Atoms

原子

網頁構成基本元素，如輸入、按鈕，也可為抽象的概念，如字體、色調等

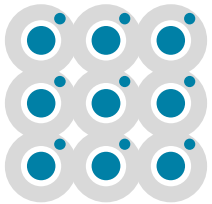


Molecules

分子

由元素或原子構成的簡單UI物件，構成形式為

原子 + 原子

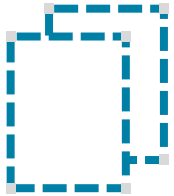


Organisms

組織

對分子而言，較為複雜的構成物，構成形式為

原子 + 分子



Templates

模板

以頁面為基礎的架構，將以上元素進行排版

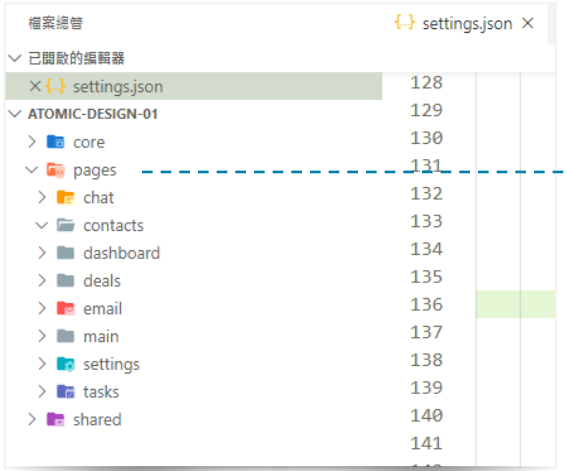
原子 + 分子 + 組織

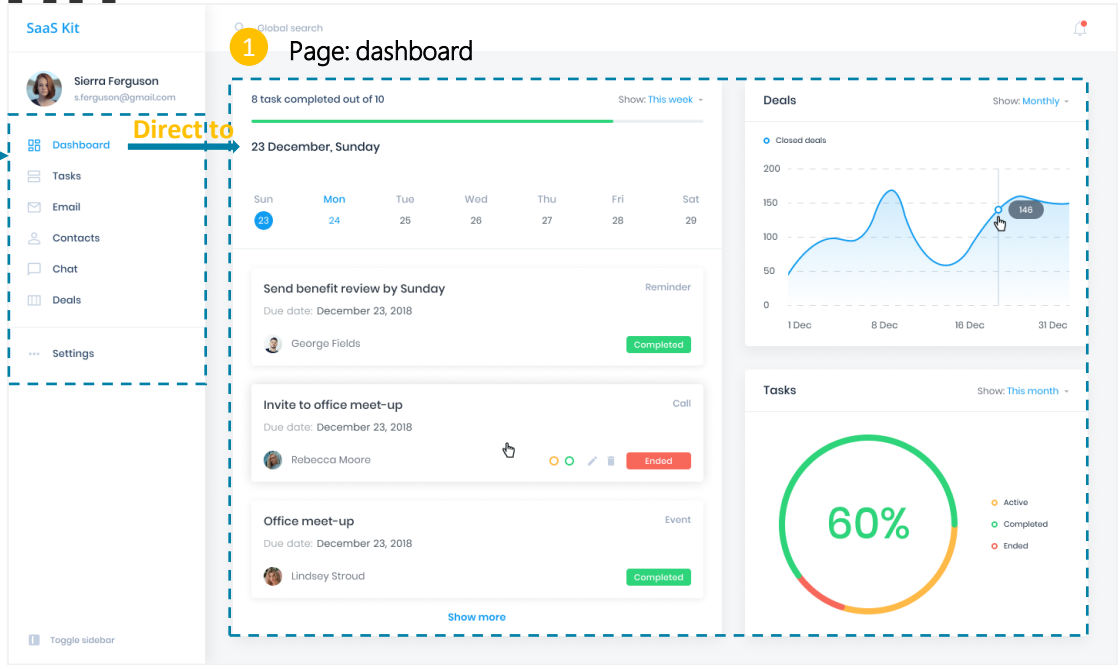
Pages

頁面

將實際內容（圖片、文章、圖表等）嵌入在特定模板當中

Atomic Design





pages (by feature)

依照頁面(或功能)劃分

core

共用模組，僅單一實例元件或服務

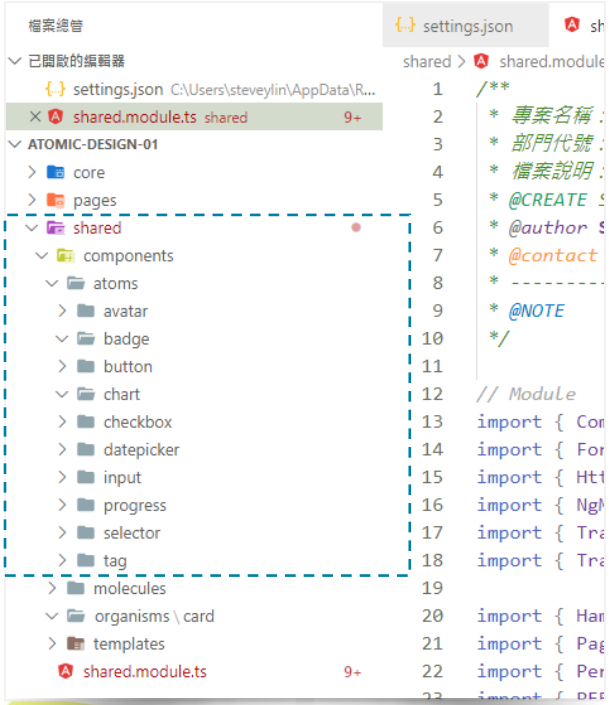
shared

共用模組，多個實例元件

Atomic Design

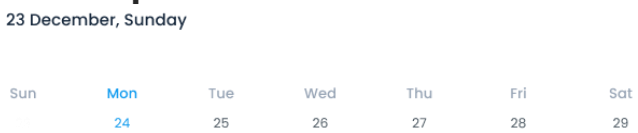
Shared Atoms

Path shared/components/atoms/{by feature}



selector

datepicker



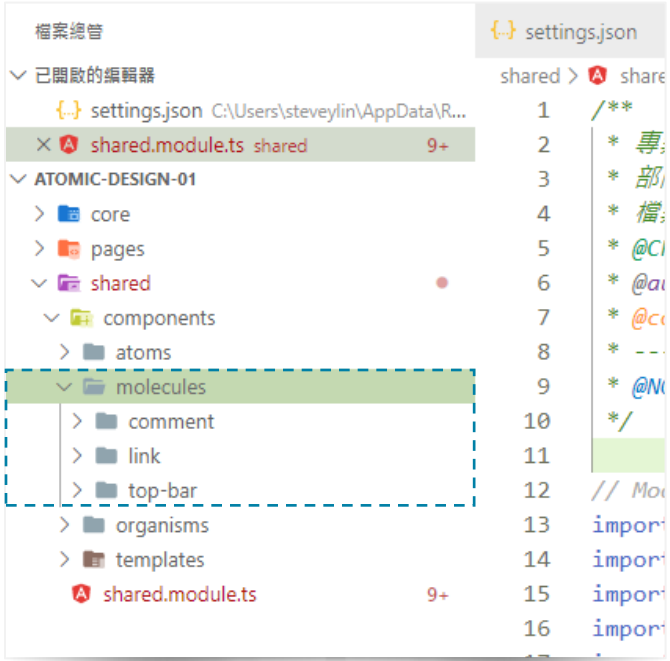
chart

※ 利用echarts繪製canvas，故可視為最小單位

Atomic Design

Shared Molecules

Path shared/components/molecules/{by feature}



comment

Sierra Ferguson
s.ferguson@gmail.com

Rebecca Moore

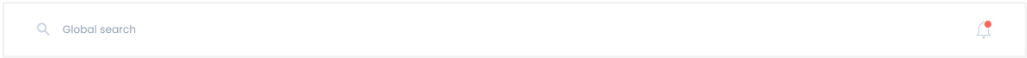
Compositio n: avatar typography

link

Dashboard ••• Settings

Compositio n: icon button

top-bar

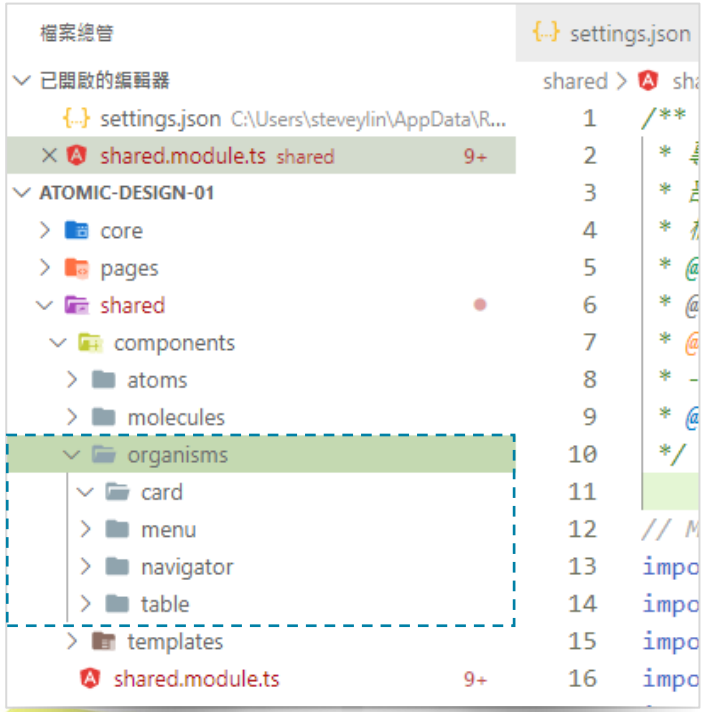


Compositio n: input badge

Atomic Design

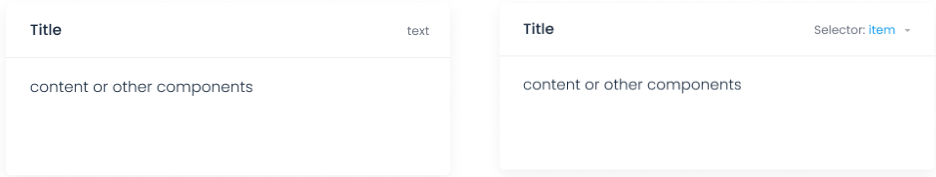
Shared Organisms

Path shared/components/organisms/{by feature}



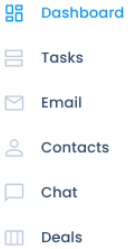
card

※ card可以嵌入其他元件



Compositio n: selector typography transclude

menu

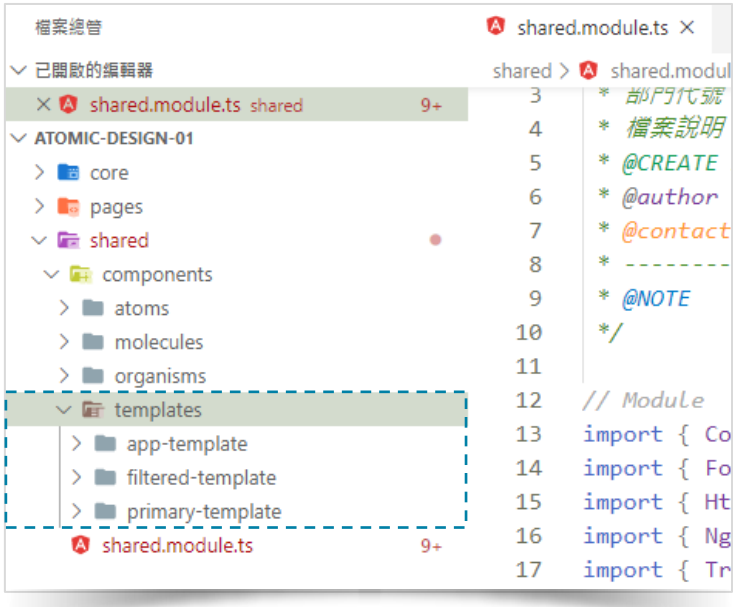


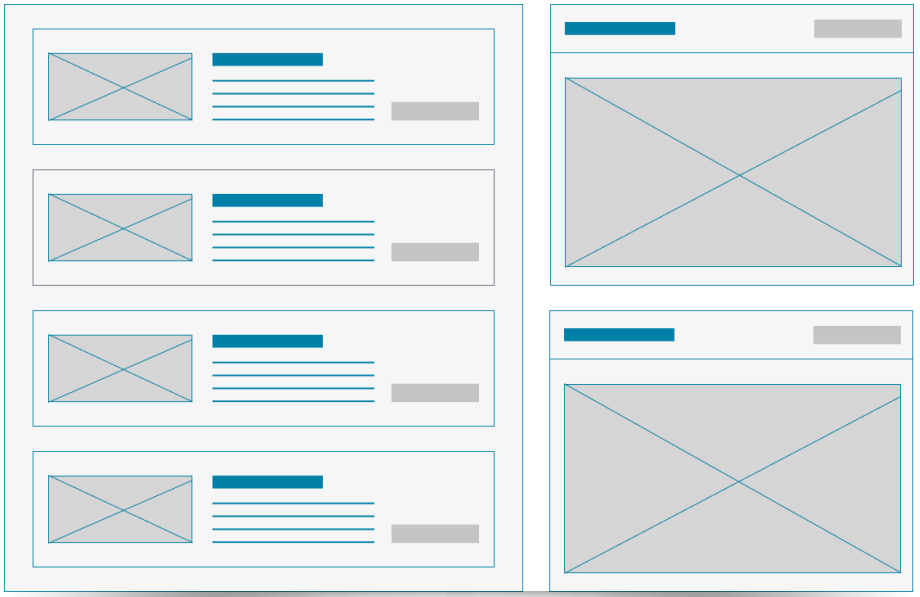
Compositio n: link

Atomic Design

primary-template

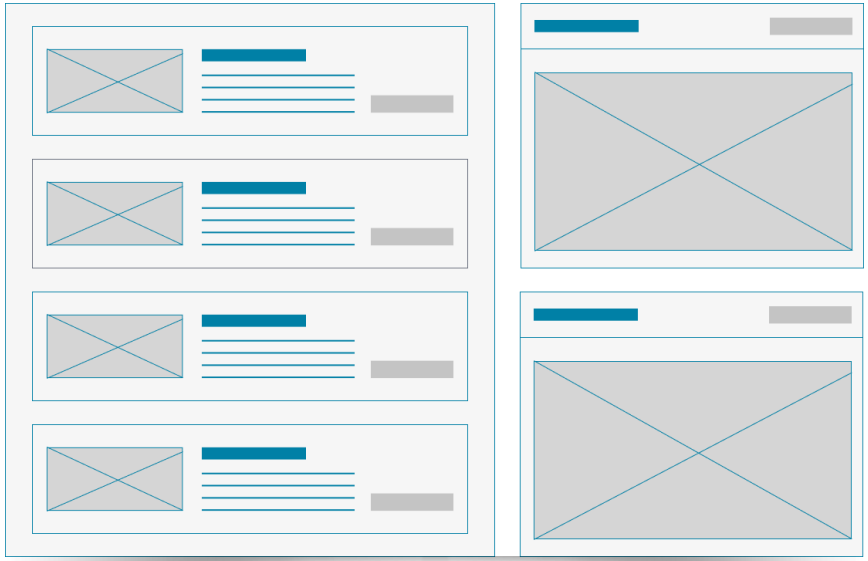
Path shared/components/templates/{by feature}



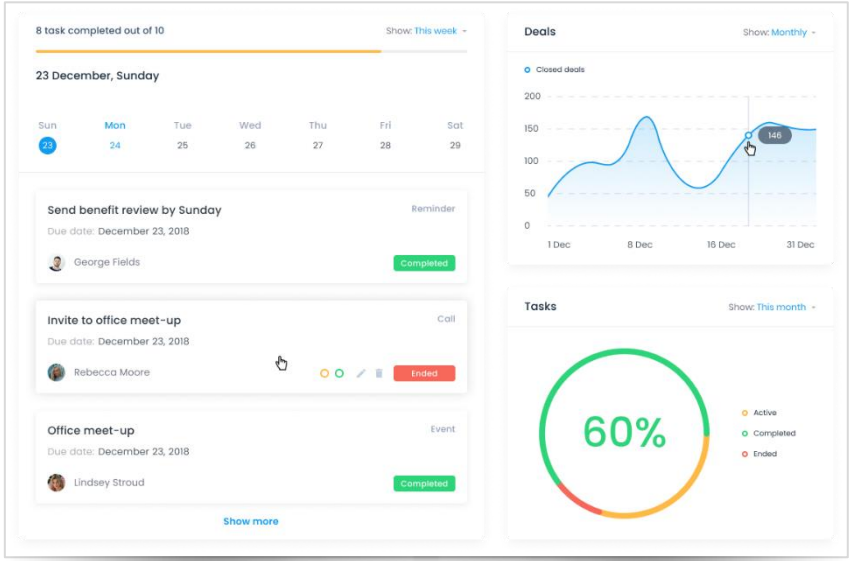


Atomic Design

primary-template



Page Dashboard



Thank you ☺