Angular 2 Meetup

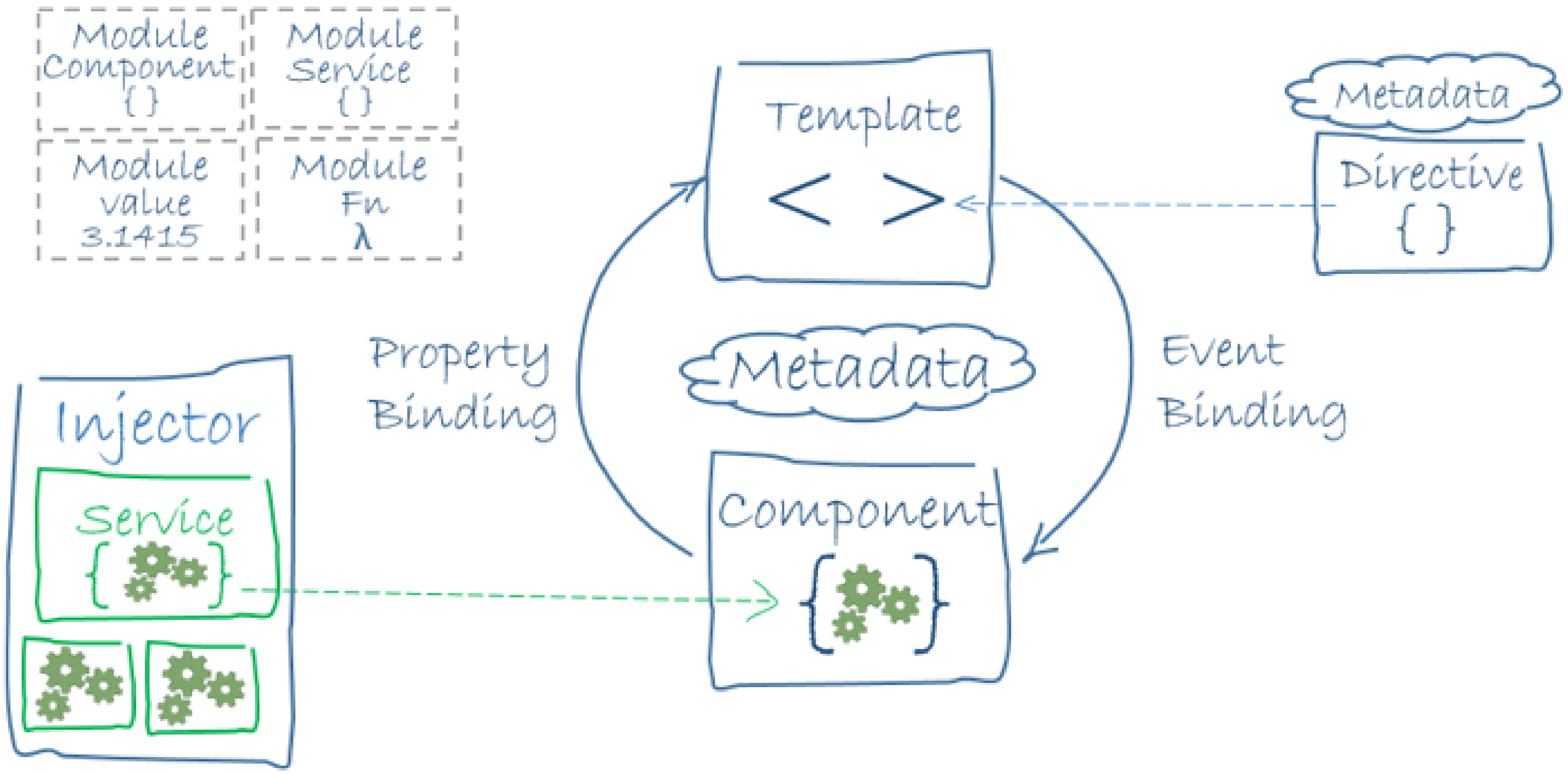
par Michel Doucerain

Pourquoi utiliser Angular 2 ?

- Architecture MVC, inheritance (extends, implements), interface, OOP
- Modularization, separation of concerns, lazy loading of modules
- TypeScript, catch errors on compilation, ES6, Auto change detection, faster development
- Dependency Injection, clean loose coupling entre components et service
- Ui compatible natvement avec iOs et Android
- Routing pour single page app
- Unit test et integration test bien supporté (Karma, Protractor)
- Rx.js librairie, données asynchrones et basé sur méchanisme de subscription, observable – observer
- Form Builder, Custom validation

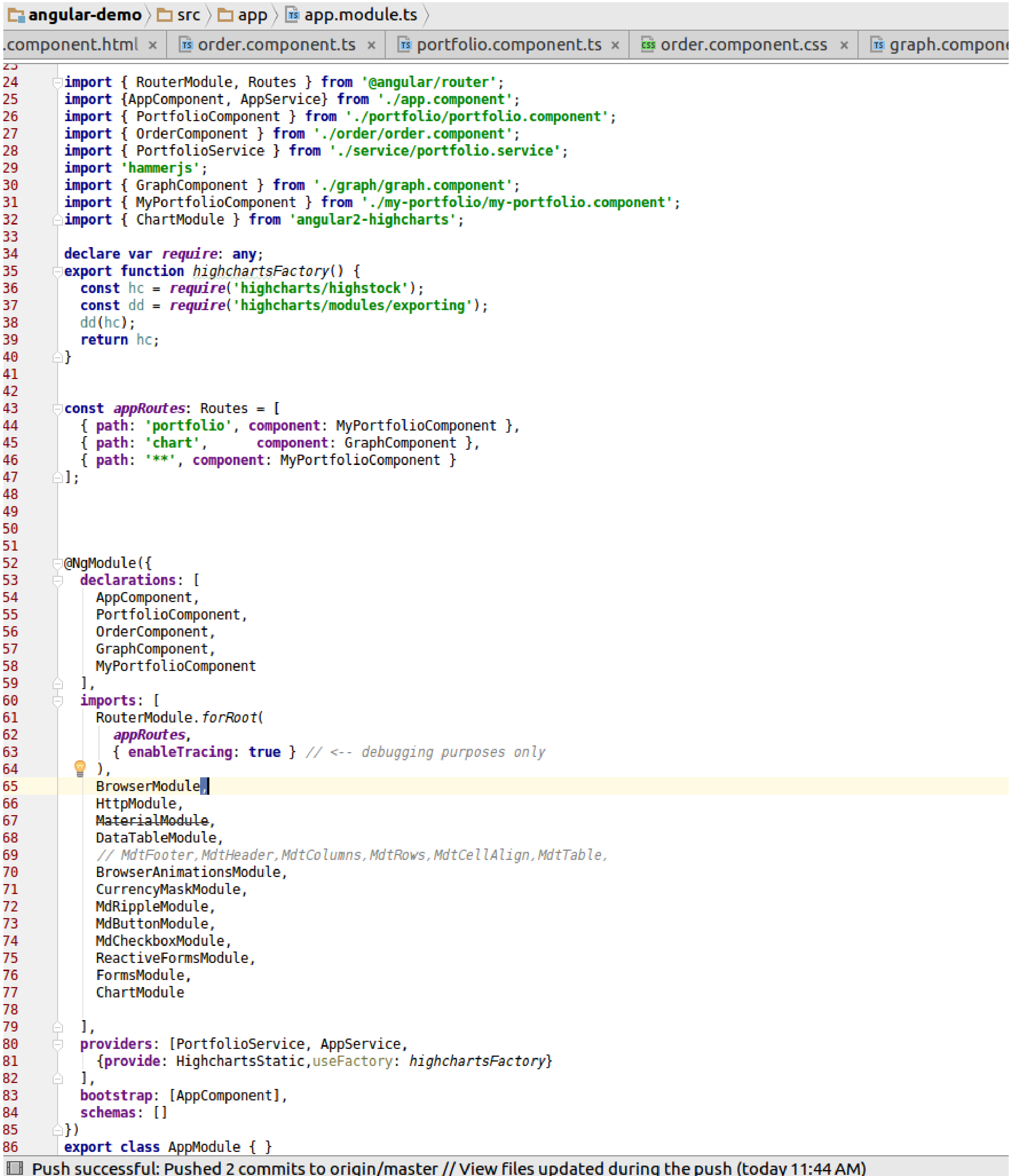
Angular Demo

Architecture Angular 2



Modules

- Toutes les apps ont un root module
- Décorateur @NgModule, bloc fonctionellement cohésif
- Ces propriétés sont
  - Déclarations : les classes de vues
    - Components, Directives, Pipes
  - Providers:
    - Services
  - Imports
    - Autres modules importés dans ce module
  - Exports
    - Déclarations disponibles à d’autres module
  - Bootstrap
    - Root component



```
import { RouterModule, Routes } from '@angular/router';
import {AppComponent, AppService} from './app.component';
import { PortfolioComponent } from './portfolio/portfolio.component';
import { OrderComponent } from './order/order.component';
import { PortfolioService } from './service/portfolio.service';
import 'hammerjs';
import { GraphComponent } from './graph/graph.component';
import { MyPortfolioComponent } from './my-portfolio/my-portfolio.component';
import { ChartModule } from 'angular2-highcharts';

declare var require: any;
export function highchartsFactory() {
  const hc = require('highcharts/highstock');
  const dd = require('highcharts/modules/exporting');
  dd(hc);
  return hc;
}


const appRoutes: Routes = [
  { path: 'portfolio', component: MyPortfolioComponent },
  { path: 'chart',      component: GraphComponent },
  { path: '**', component: MyPortfolioComponent }
];




@NgModule({
  declarations: [
    AppComponent,
    PortfolioComponent,
    OrderComponent,
    GraphComponent,
    MyPortfolioComponent
  ],
  imports: [
    RouterModule.forRoot(
      appRoutes,
      { enableTracing: true } // <-- debugging purposes only
    ),
    BrowserModule,
    HttpModule,
    ~~MaterialModule~~,
    DataTableModule,
    // MdtFooter,MdtHeader,MdtColumns,MdtRows,MdtCellAlign,MdtTable,
    BrowserAnimationsModule,
    CurrencyMaskModule,
    MdRippleModule,
    MdButtonModule,
    MdCheckboxModule,
    ReactiveFormsModule,
    FormsModule,
    ChartModule

  ],
  providers: [PortfolioService, AppService,
    {provide: HighchartsStatic,useFactory: highchartsFactory}
  ],
  bootstrap: [AppComponent],
  schemas: []
})
export class AppModule { }
```

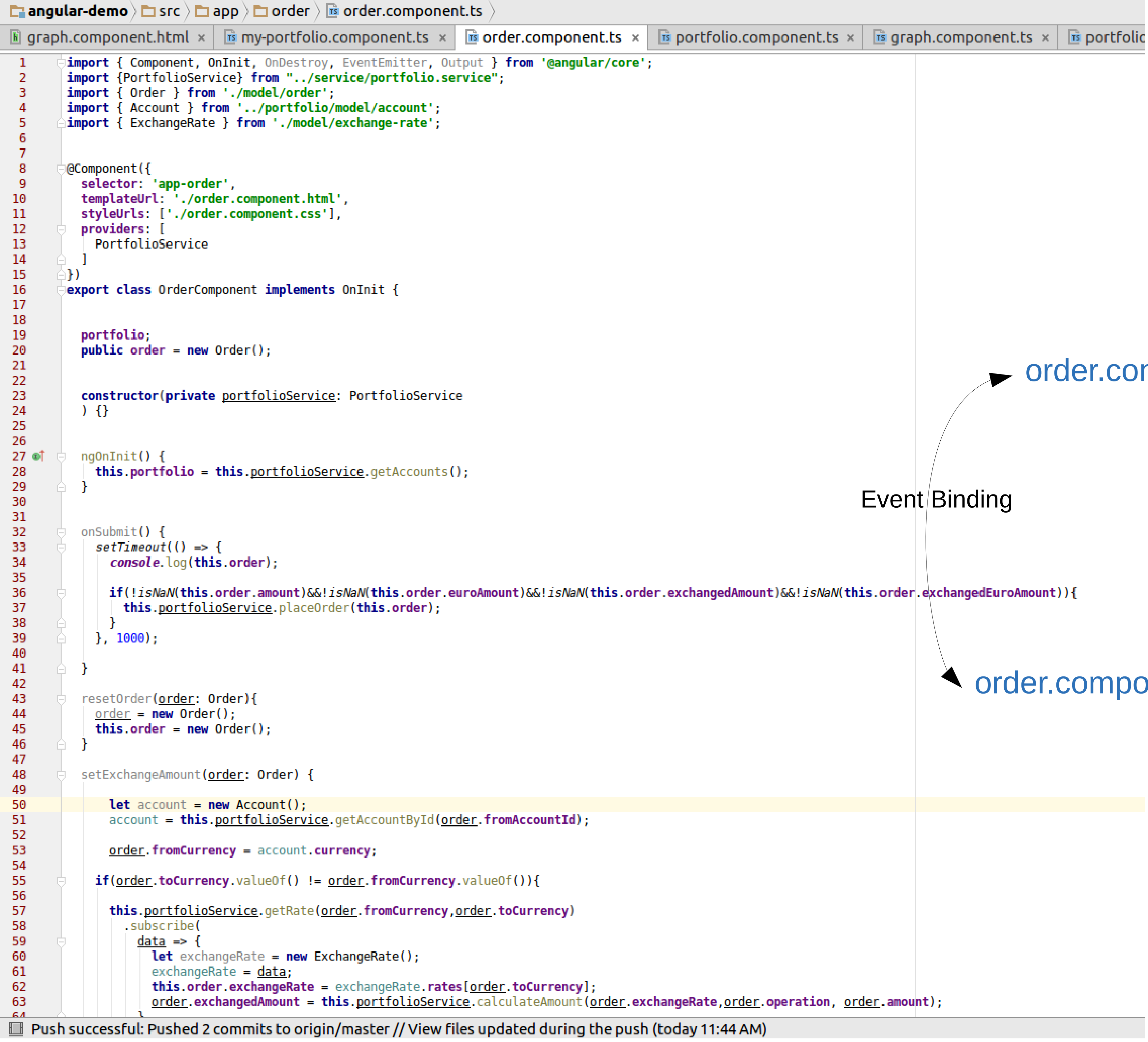
Push successful: Pushed 2 commits to origin/master // View files updated during the push (today 11:44 AM)

app-module.ts

main.ts

@Component()

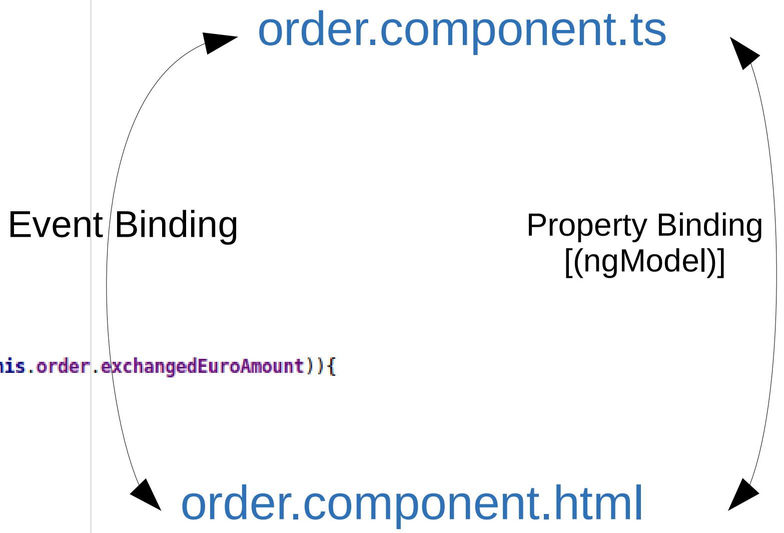
- Classes en TypeScript, communique avec le template html avec des evènements ou des ‘propriety binding’, définit le comportement logique
- Dependency Injection avec des services, loose coupling, passe services dans constructor
- Besoin de définir providers et directives dans le component
- Lifecyle hooks: ngOnChanges, ngOnInit, ngDoCheck (change detection), ngAfterContentInit, ngAfterContentChecked, ngAfterViewInit, ngAfterViewChecked


angular-demo › src › app › order › order.component.ts

graph.component.html × | my-portfolio.component.ts × | order.component.ts × | portfolio.component.ts × | graph.component.ts × | portfolic

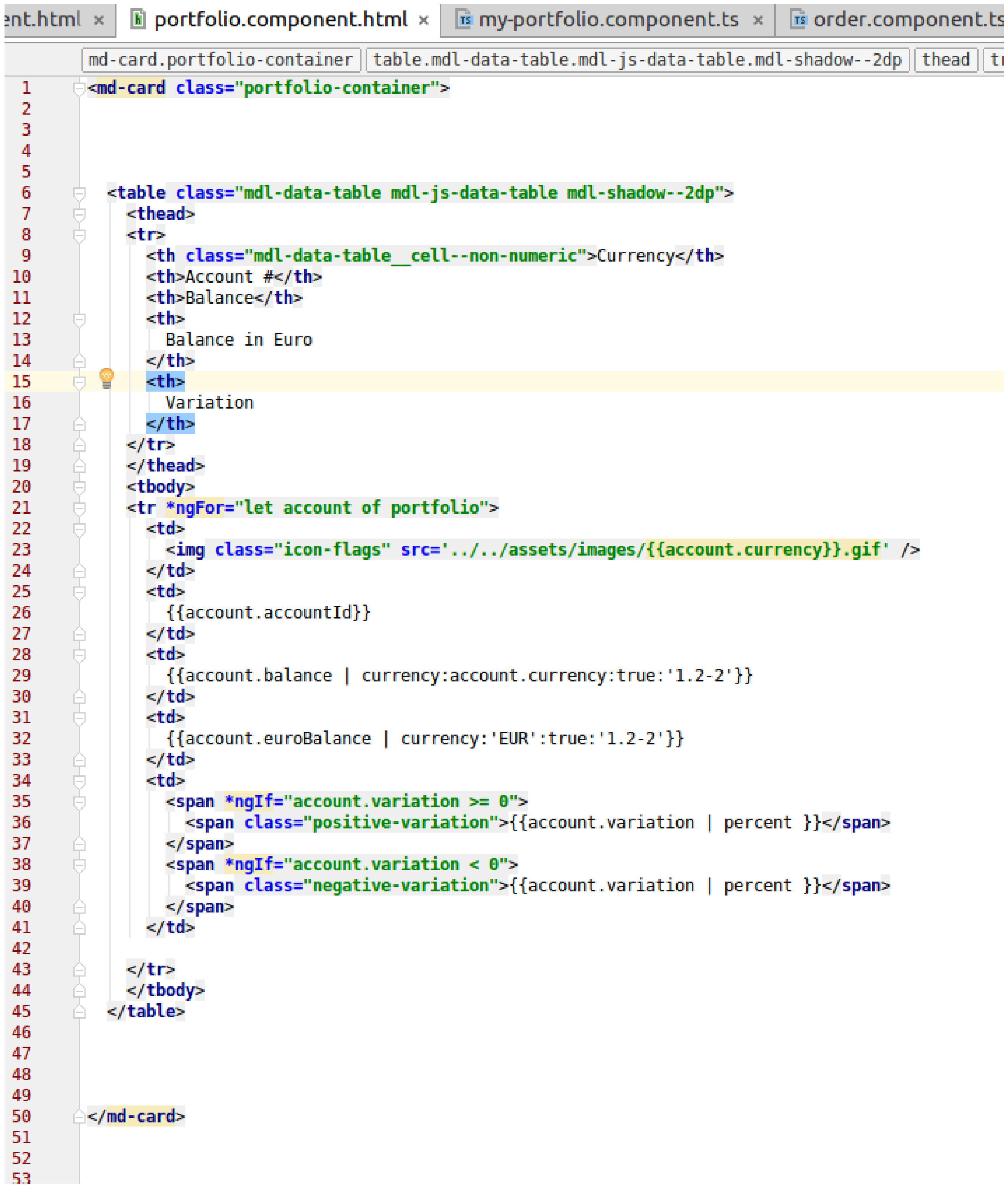
```
import { Component, OnInit, OnDestroy, EventEmitter, Output } from '@angular/core';
import {PortfolioService} from "../service/portfolio.service";
import { Order } from './model/order';
import { Account } from '../portfolio/model/account';
import { ExchangeRate } from './model/exchange-rate';


@Component({
  selector: 'app-order',
  templateUrl: './order.component.html',
  styleUrls: ['./order.component.css'],
  providers: [
    PortfolioService
  ]
})
export class OrderComponent implements OnInit {


  portfolio;
  public order = new Order();


  constructor(private portfolioService: PortfolioService
  ) {}


  ngOnInit() {
    this.portfolio = this.portfolioService.getAccounts();
  }


  onSubmit() {
    setTimeout(() => {
      console.log(this.order);

      if(!isNaN(this.order.amount)&&!isNaN(this.order.euroAmount)&&!isNaN(this.order.exchangedAmount)&&!isNaN(this.order.exchangedEuroAmount)){
        this.portfolioService.placeOrder(this.order);
      }
    }, 1000);

  }

  resetOrder(order: Order){
    order = new Order();
    this.order = new Order();
  }

  setExchangeAmount(order: Order) {

      let account = new Account();
      account = this.portfolioService.getAccountById(order.fromAccountId);

      order.fromCurrency = account.currency;

    if(order.toCurrency.valueOf() != order.fromCurrency.valueOf()){

      this.portfolioService.getRate(order.fromCurrency,order.toCurrency)
        .subscribe(
          data => {
            let exchangeRate = new ExchangeRate();
            exchangeRate = data;
            this.order.exchangeRate = exchangeRate.rates[order.toCurrency];
            order.exchangedAmount = this.portfolioService.calculateAmount(order.exchangeRate,order.operation, order.amount);
```

Push successful: Pushed 2 commits to origin/master // View files updated during the push (today 11:44 AM)



Template

- View en HTML, présentation UI, avec de l’encapsulation en utilisant des ‘customs tags’ qui font appel à d’autre components
- Utilise interpolation {{}} pour récupérer un objet du component, peut aussi utiliser des templates expressions operators (| pipe pour formattage, ?. Elvis operator pour protégé d’une erreur null pointer)
- Data binding et two way binding


ent.html
portfolio.component.html
my-portfolio.component.ts
order.component.ts
md-card.portfolio-container
table.mdl-data-table.mdl-js-data-table.mdl-shadow--2dp
thead

```html
<md-card class="portfolio-container">




  <table class="mdl-data-table mdl-js-data-table mdl-shadow--2dp">
    <thead>
    <tr>
      <th class="mdl-data-table__cell--non-numeric">Currency</th>
      <th>Account #</th>
      <th>Balance</th>
      <th>
        Balance in Euro
      </th>
      <th>
        Variation
      </th>
    </tr>
    </thead>
    <tbody>
    <tr *ngFor="let account of portfolio">
      <td>
        <img class="icon-flags" src='../../assets/images/{{account.currency}}.gif' />
      </td>
      <td>
        {{account.accountId}}
      </td>
      <td>
        {{account.balance | currency:account.currency:true:'1.2-2'}}
      </td>
      <td>
        {{account.euroBalance | currency:'EUR':true:'1.2-2'}}
      </td>
      <td>
        <span *ngIf="account.variation >= 0">
          <span class="positive-variation">{{account.variation | percent }}</span>
        </span>
        <span *ngIf="account.variation < 0">
          <span class="negative-variation">{{account.variation | percent }}</span>
        </span>
      </td>

    </tr>
    </tbody>
  </table>




</md-card>
```

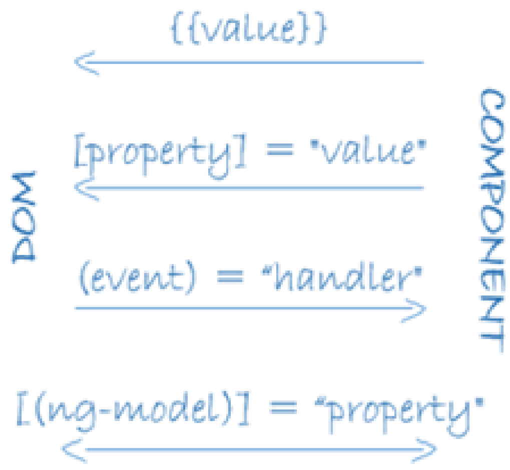
portfolio.component.html

order.component.html

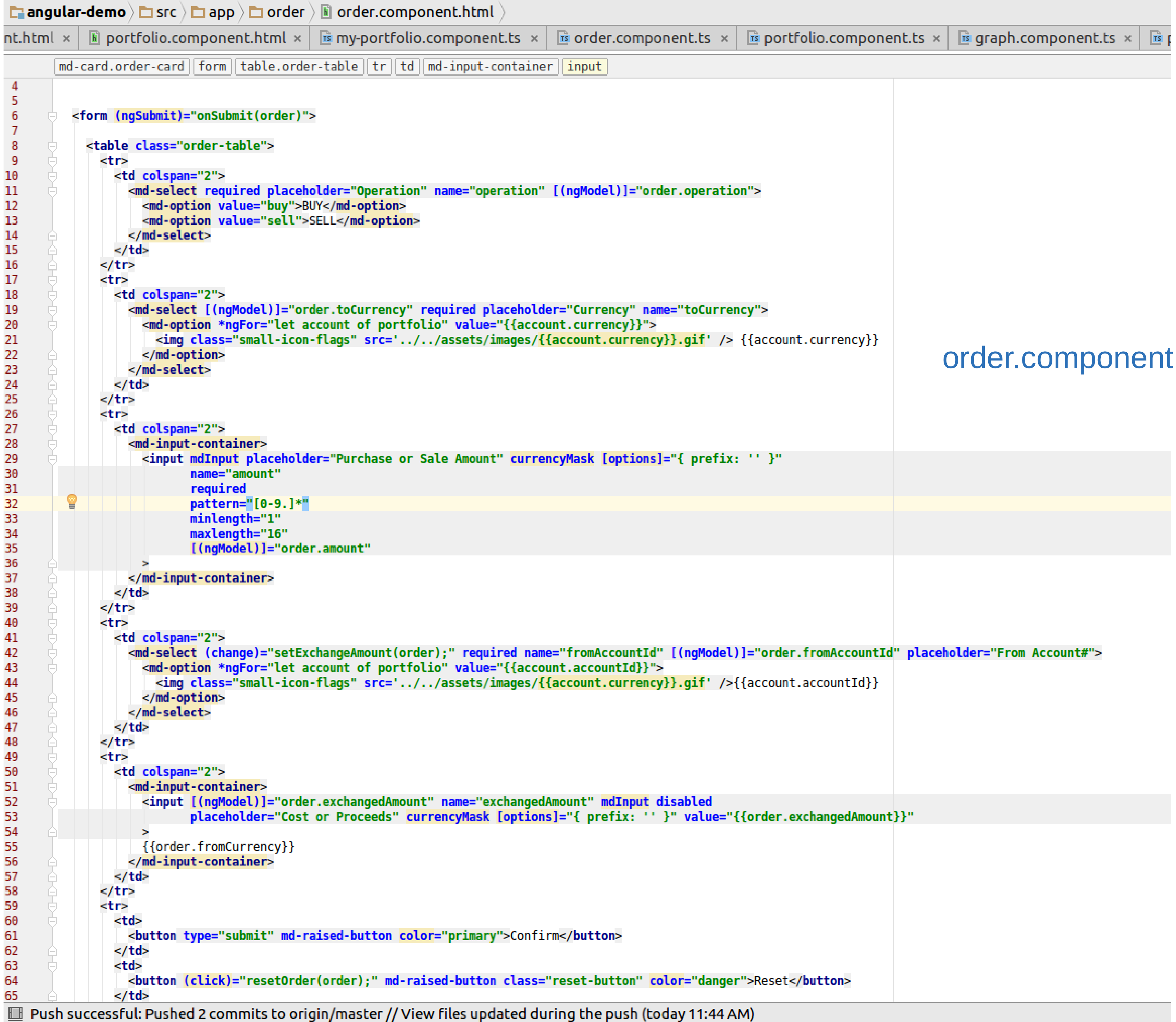
Metadata

- Décorator qui spécifie type de classe et son comportement par défault
- Examples de décorator:
  - @Component
  - @Directive
  - @Pipe
  - @Injectable
  - @Input()
  - @Output()
  - @NgModule

Data binding



- Communique data entre template et component
- Property Binding, @Input() myProperty; est mise à jour dans le template [myProperty]="someExpression">
- Interpolation {{account.accountId}} réfère à l'objet account dans le component depuis le template
- Event binding (change)="setExchangeAmount(order);", sur le changement du dropdown menu, on calcule le montant échangé dans une autre devise dans le component
- [(ngModel)]="order.toCurrency lie l'objet order dans le template formulaire et le component


angular-demo 〉 src 〉 app 〉 order 〉 order.component.html 〉

nt.html × | portfolio.component.html × | my-portfolio.component.ts × | order.component.ts × | portfolio.component.ts × | graph.component.ts × |

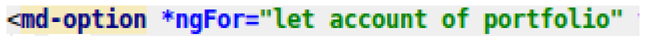
md-card.order-card | form | table.order-table | tr | td | md-input-container | input

```


<form (ngSubmit)="onSubmit(order)">

  <table class="order-table">
    <tr>
      <td colspan="2">
        <md-select required placeholder="Operation" name="operation" [(ngModel)]="order.operation">
          <md-option value="buy">BUY</md-option>
          <md-option value="sell">SELL</md-option>
        </md-select>
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <md-select [(ngModel)]="order.toCurrency" required placeholder="Currency" name="toCurrency">
          <md-option *ngFor="let account of portfolio" value="{{account.currency}}">
            <img class="small-icon-flags" src='../../assets/images/{{account.currency}}.gif' /> {{account.currency}}
          </md-option>
        </md-select>
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <md-input-container>
          <input mdInput placeholder="Purchase or Sale Amount" currencyMask [options]="{ prefix: '' }"
                 name="amount"
                 required
                 pattern="[0-9.]*"
                 minlength="1"
                 maxlength="16"
                 [(ngModel)]="order.amount"
          >
        </md-input-container>
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <md-select (change)="setExchangeAmount(order);" required name="fromAccountId" [(ngModel)]="order.fromAccountId" placeholder="From Account#">
          <md-option *ngFor="let account of portfolio" value="{{account.accountId}}">
            <img class="small-icon-flags" src='../../assets/images/{{account.currency}}.gif' />{{account.accountId}}
          </md-option>
        </md-select>
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <md-input-container>
          <input [(ngModel)]="order.exchangedAmount" name="exchangedAmount" mdInput disabled
                 placeholder="Cost or Proceeds" currencyMask [options]="{ prefix: '' }" value="{{order.exchangedAmount}}"
          >
          {{order.fromCurrency}}
        </md-input-container>
      </td>
    </tr>
    <tr>
      <td>
        <button type="submit" md-raised-button color="primary">Confirm</button>
      </td>
      <td>
        <button (click)="resetOrder(order);" md-raised-button class="reset-button" color="danger">Reset</button>
      </td>
```

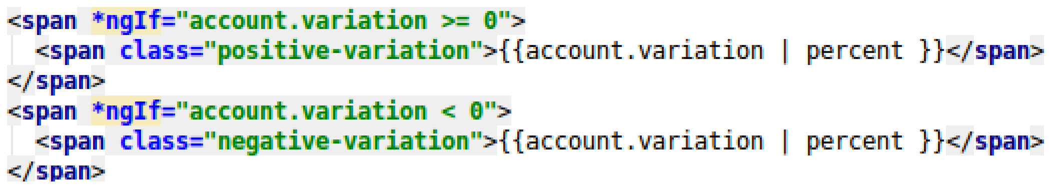
Push successful: Pushed 2 commits to origin/master // View files updated during the push (today 11:44 AM)

order.component.html

@Directive

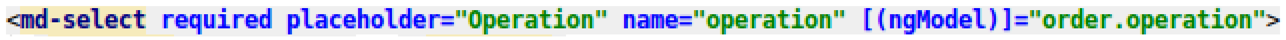
- Transform template, altère l’apparence ou le comportement d’un element du DOM.
  - Structural directives (rajoute, retire element du DOM)
    - 

      ```
      <md-option *ngFor="let account of portfolio"
      ```

    - 

      ```
      <span *ngIf="account.variation >= 0">
        <span class="positive-variation">{{account.variation | percent }}</span>
      </span>
      <span *ngIf="account.variation < 0">
        <span class="negative-variation">{{account.variation | percent }}</span>
      </span>
      ```

  - Attribute directives (altère apparence ou comportement d’un élément)
    - ngModel directive, qui implémente le ‘two-way data binding’

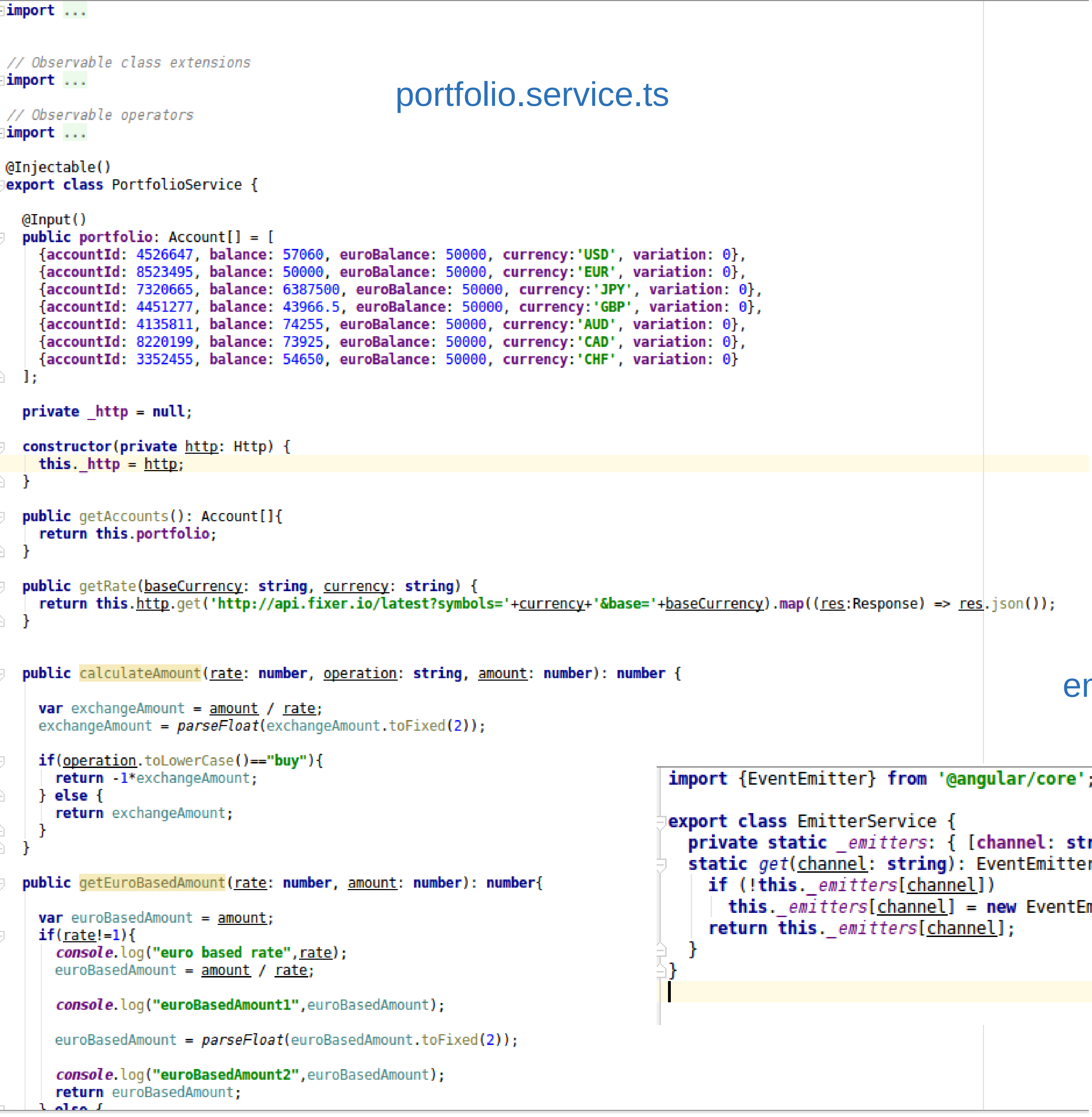


```
<md-select required placeholder="Operation" name="operation" [(ngModel)]="order.operation">
```

Services

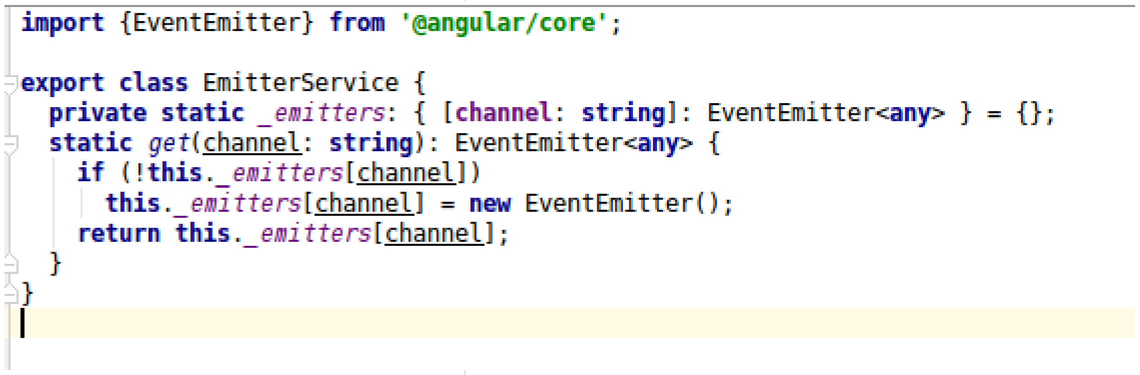
- Class qui accomplit une fonction bien précise dans l’application
- Communique avec le serveur avec des services Rest
- Components font appèle aux services pour communiquer avec serveurs ou autres components, (définit dans la propriété providers du component)
  - Publish/subscibe ou event emitter/listener
- Examples:
  - logging service
  - data service
  - message bus
  - Foreign exchange rate service

portfolio.service.ts



```
import ...


// Observable class extensions
import ...

// Observable operators
import ...

@Injectable()
export class PortfolioService {

  @Input()
  public portfolio: Account[] = [
    {accountId: 4526647, balance: 57060, euroBalance: 50000, currency:'USD', variation: 0},
    {accountId: 8523495, balance: 50000, euroBalance: 50000, currency:'EUR', variation: 0},
    {accountId: 7320665, balance: 6387500, euroBalance: 50000, currency:'JPY', variation: 0},
    {accountId: 4451277, balance: 43966.5, euroBalance: 50000, currency:'GBP', variation: 0},
    {accountId: 4135811, balance: 74255, euroBalance: 50000, currency:'AUD', variation: 0},
    {accountId: 8220199, balance: 73925, euroBalance: 50000, currency:'CAD', variation: 0},
    {accountId: 3352455, balance: 54650, euroBalance: 50000, currency:'CHF', variation: 0}
  ];

  private _http = null;

  constructor(private http: Http) {
    this._http = http;
  }

  public getAccounts(): Account[]{
    return this.portfolio;
  }

  public getRate(baseCurrency: string, currency: string) {
    return this.http.get('http://api.fixer.io/latest?symbols='+currency+'&base='+baseCurrency).map((res:Response) => res.json());
  }


  public calculateAmount(rate: number, operation: string, amount: number): number {

    var exchangeAmount = amount / rate;
    exchangeAmount = parseFloat(exchangeAmount.toFixed(2));

    if(operation.toLowerCase()=="buy"){
      return -1*exchangeAmount;
    } else {
      return exchangeAmount;
    }
  }

  public getEuroBasedAmount(rate: number, amount: number): number{

    var euroBasedAmount = amount;
    if(rate!=1){
      console.log("euro based rate",rate);
      euroBasedAmount = amount / rate;

      console.log("euroBasedAmount1",euroBasedAmount);

      euroBasedAmount = parseFloat(euroBasedAmount.toFixed(2));

      console.log("euroBasedAmount2",euroBasedAmount);
      return euroBasedAmount;
    } else {
```

emitter-service.ts

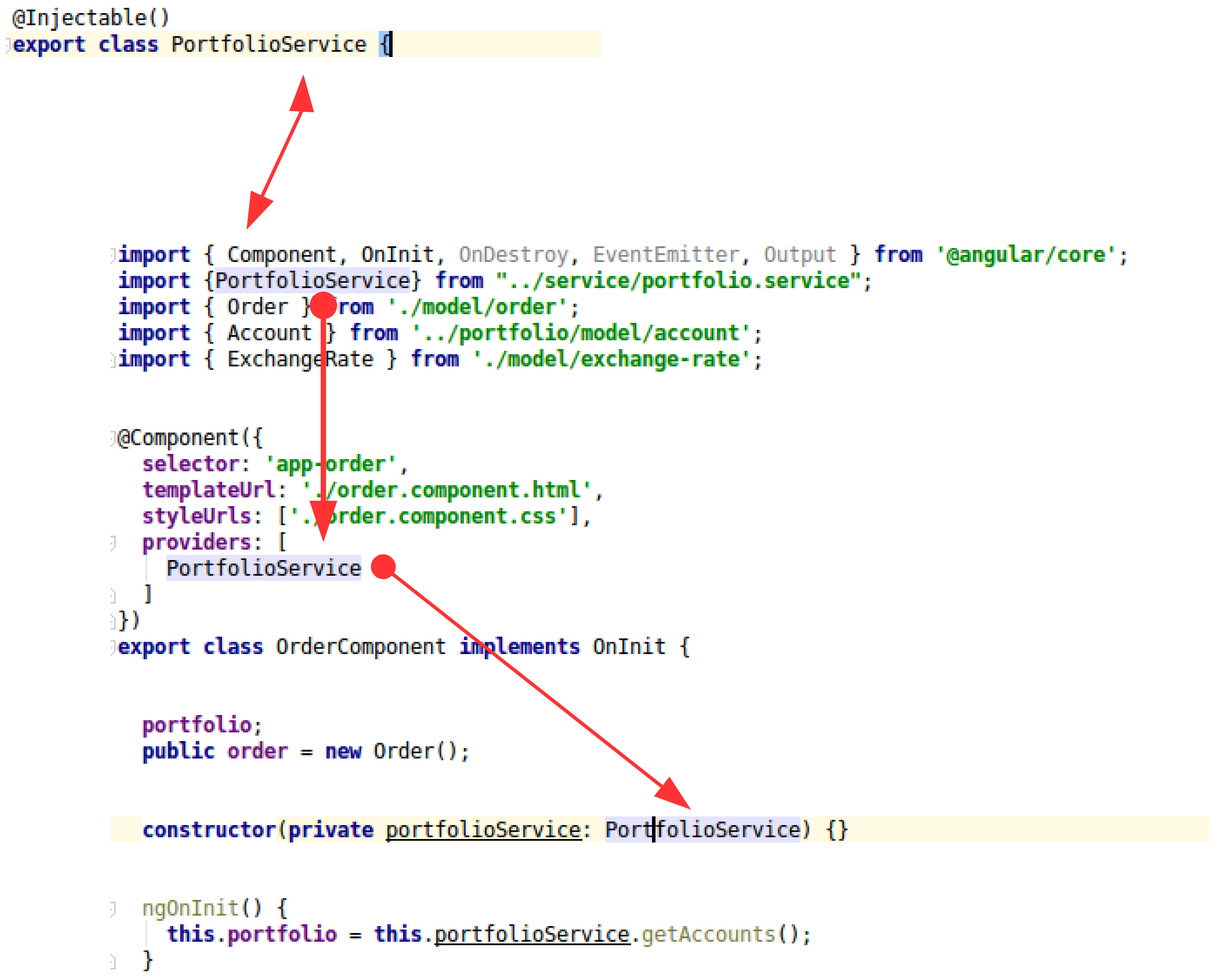


```
import {EventEmitter} from '@angular/core';

export class EmitterService {
  private static _emitters: { [channel: string]: EventEmitter<any> } = {};
  static get(channel: string): EventEmitter<any> {
    if (!this._emitters[channel])
      this._emitters[channel] = new EventEmitter();
    return this._emitters[channel];
  }
}
```

Component
service
{Constructor(service)}

Dependency injection

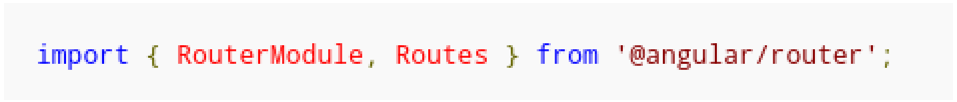
- Principe de Loose Coupling pour définir dépendences de services dans le constructor du component
- Utilise décorator @Injectable() pour définir que le service est disponible à être injecté par le component
- Facilite la communication entre component, service à travers l’application



```
@Injectable()
export class PortfolioService {
```

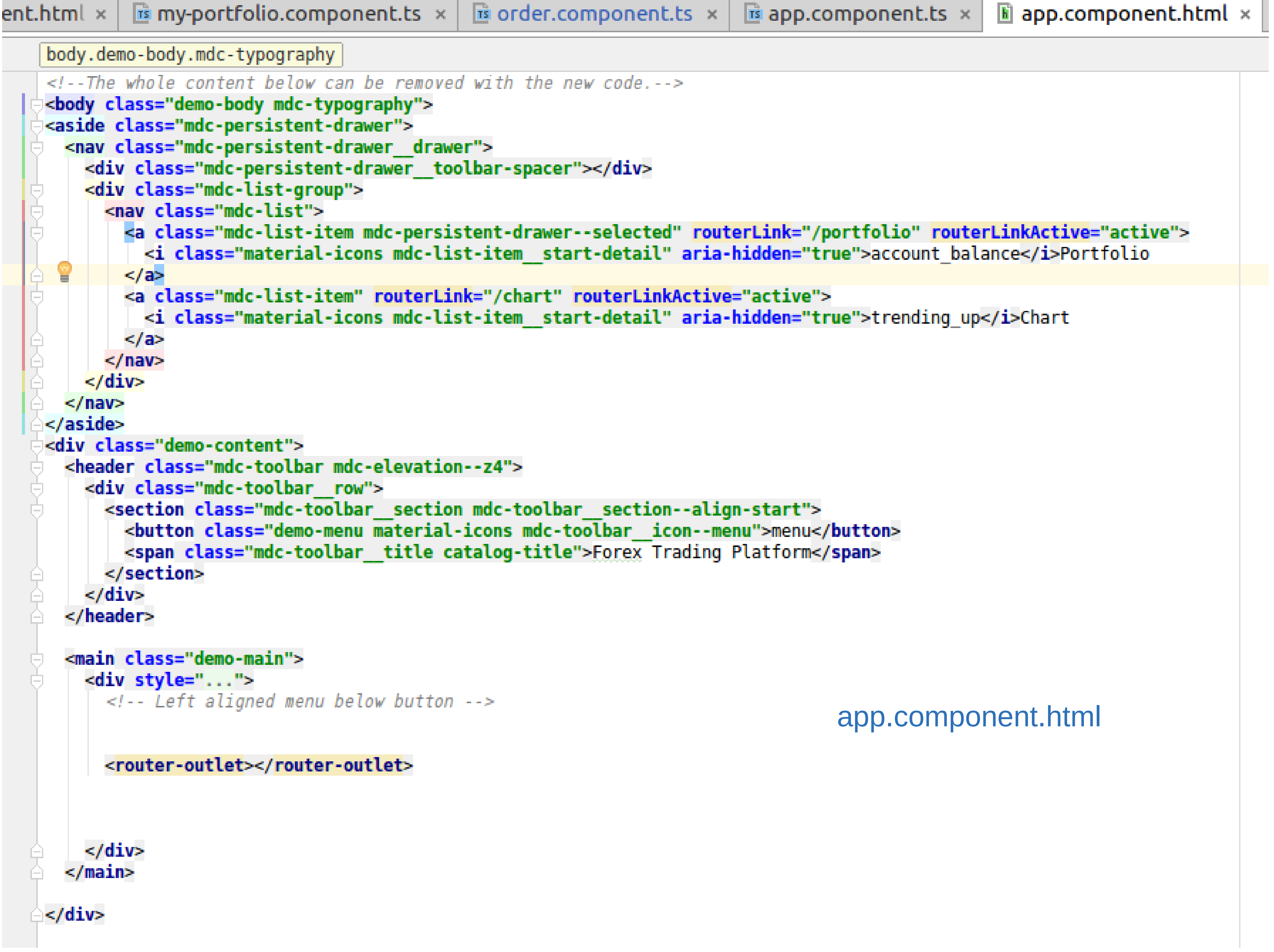
```
import { Component, OnInit, OnDestroy, EventEmitter, Output } from '@angular/core';
import {PortfolioService} from "../service/portfolio.service";
import { Order } from './model/order';
import { Account } from '../portfolio/model/account';
import { ExchangeRate } from './model/exchange-rate';


@Component({
  selector: 'app-order',
  templateUrl: './order.component.html',
  styleUrls: ['./order.component.css'],
  providers: [
    PortfolioService
  ]
})
export class OrderComponent implements OnInit {


  portfolio;
  public order = new Order();


  constructor(private portfolioService: PortfolioService) {}


  ngOnInit() {
    this.portfolio = this.portfolioService.getAccounts();
  }
```

Routing

- Navigation dans l’application
- Single page app
- Navigation à travers plusieurs components sur la même page
- 

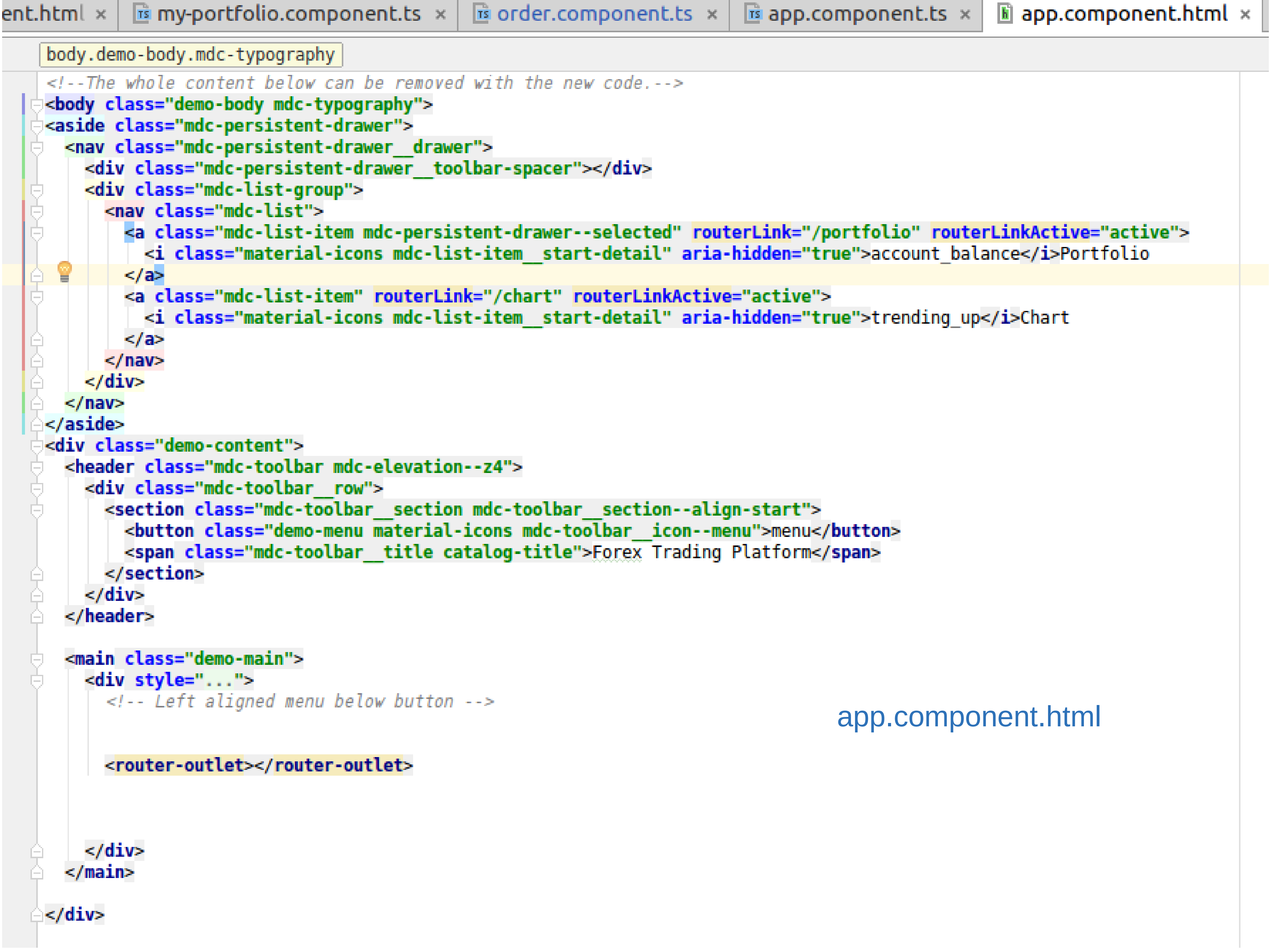
```
import { RouterModule, Routes } from '@angular/router';
```

- Router outlet permet de définir une navigation dans une tag et garder un menu commun



```html
<!--The whole content below can be removed with the new code.-->
<body class="demo-body mdc-typography">
<aside class="mdc-persistent-drawer">
  <nav class="mdc-persistent-drawer__drawer">
    <div class="mdc-persistent-drawer__toolbar-spacer"></div>
    <div class="mdc-list-group">
      <nav class="mdc-list">
        <a class="mdc-list-item mdc-persistent-drawer--selected" routerLink="/portfolio" routerLinkActive="active">
          <i class="material-icons mdc-list-item__start-detail" aria-hidden="true">account_balance</i>Portfolio
        </a>
        <a class="mdc-list-item" routerLink="/chart" routerLinkActive="active">
          <i class="material-icons mdc-list-item__start-detail" aria-hidden="true">trending_up</i>Chart
        </a>
      </nav>
    </div>
  </nav>
</aside>
<div class="demo-content">
  <header class="mdc-toolbar mdc-elevation--z4">
    <div class="mdc-toolbar__row">
      <section class="mdc-toolbar__section mdc-toolbar__section--align-start">
        <button class="demo-menu material-icons mdc-toolbar__icon--menu">menu</button>
        <span class="mdc-toolbar__title catalog-title">Forex Trading Platform</span>
      </section>
    </div>
  </header>

  <main class="demo-main">
    <div style="...">
      <!-- Left aligned menu below button -->


      <router-outlet></router-outlet>



    </div>
  </main>

</div>
```

app.component.html



```
<!--The whole content below can be removed with the new code.-->
<body class="demo-body mdc-typography">
<aside class="mdc-persistent-drawer">
  <nav class="mdc-persistent-drawer__drawer">
    <div class="mdc-persistent-drawer__toolbar-spacer"></div>
    <div class="mdc-list-group">
      <nav class="mdc-list">
        <a class="mdc-list-item mdc-persistent-drawer--selected" routerLink="/portfolio" routerLinkActive="active">
          <i class="material-icons mdc-list-item__start-detail" aria-hidden="true">account_balance</i>Portfolio
        </a>
        <a class="mdc-list-item" routerLink="/chart" routerLinkActive="active">
          <i class="material-icons mdc-list-item__start-detail" aria-hidden="true">trending_up</i>Chart
        </a>
      </nav>
    </div>
  </nav>
</aside>
<div class="demo-content">
  <header class="mdc-toolbar mdc-elevation--z4">
    <div class="mdc-toolbar__row">
      <section class="mdc-toolbar__section mdc-toolbar__section--align-start">
        <button class="demo-menu material-icons mdc-toolbar__icon--menu">menu</button>
        <span class="mdc-toolbar__title catalog-title">Forex Trading Platform</span>
      </section>
    </div>
  </header>

  <main class="demo-main">
    <div style="...">
      <!-- Left aligned menu below button -->


      <router-outlet></router-outlet>



    </div>
  </main>

</div>
```

app.component.html