## Fix all the bugs before implementing new features

***ERROR 1***

zone.js:516 Unhandled Promise rejection:

Component AppComponent is not part of any NgModule or the module has not been imported into your module.

SUGGESTION:

- Declare ***AppComponent*** in @NgModule --> see: **app.module.ts**

***ERROR 2***

Error in :0:0 caused by: The selector "app-partylist" did not match any elements

SUGGESTION:

- The selector "app-partylist" in app.component.ts did not match any elements in index.html.

- Change the selector to conform with the parent-template:

**selector: 'app-partylist'** should be **selector: 'app-kieskompas’**

***ERROR 3***

EXCEPTION: Error in ./AppComponent class AppComponent - inline template:25:6 caused by: No provider for TemplateRef!

SOLUTION:

- Change **app.component.html** --> **ngIf** should be **\*ngFor**

***ERROR 4***

EXCEPTION: Error in ./AppComponent class AppComponent - inline template:16:40 caused by: Cannot read property 'name' of undefined

SUGGESTION:

Define **categories** in app.component.ts and categories.ts

HOW?

Use mocks data instead in**app.component.ts**:

Step 1. Define in categories.ts your Category Type, and export the mock Categories:

|  |
| --- |
| **export class** Category {  **name**: String;  **selected**: Boolean; }  **export const** mockCategories = [  {name: 'Alles', selected: **true**},  ….  ]; |

Step 2. Update **app.component.ts**:

* Add imports from categories.ts
* Remove Category class in app.component.ts
* Assign mockCategories to the array Category[]

**import** {mockCategories ,Category} **from './mocks/categories'**;

**categories**: Category [] = mockCategories; // use mocks data instead

Step 3. Update app.component.html

**ngFor** should be **\*ngFor in app.component.html**

***ERROR 5***

Unhandled Promise rejection: Template parse errors:

Can't bind to 'ngForIn' since it isn't a known property of

'li'. (" <a data-type="all" href="#0">All</a>

</li>

<li [ERROR ->]\*ngFor="let category in categories">

<a class="selected" href="#0">{{category.name}}<"): AppComponent@15:16

SUGGESTION:

Use:  **\*ngFor="let category of categories"**

Instead of:  **\*ngFor="let category in categories"**

## TODO: Features to implement in your app

**1. Display all parties on the main area**

1. Replace the Party class in **app.components.ts** by an *import***:**

import { mockParties, Party } from './mocks/parties';

2. Assign the mockParties in the **app.components.ts:**

parties: Party[] = mockParties;

3. SOLVE EXCEPTION

Failed to load resource: the server responded with a status of 404 (Not Found) party.cover: GET http://localhost:4200/party.logo 404 (Not Found)

SUGGESTION:

Go to the **app.component.html:**

<li \*ngFor="let party of parties"><img src="party.logo" click="clicked"/></li>

Note that you should use property bindings here!

**2a. On click on a category, update the list of parties**

SUGGESTION:

A filter should be applied to the selected category of the party!

Change the following in **app.component.html:**

|  |
| --- |
| <li \*ngFor="let category of categories">  <a class="selected" href="#0" **(**click**)**="clicked(category)">{{category.name}}</a>  </li> |

Replace the clicked method in **app.components.ts** with the following snippet**:**

|  |
| --- |
| private static sortParties(party1: Party, party2: Party): number {  if (party1.afkorting > party2.afkorting) {  return 1;  } else if (party1.afkorting < party2.afkorting) {  return -1;  } else {  return 0;  }  }  private filterBooks(category: Category): void {  if (category.name === 'Alles') {  this.parties = mockParties;  return;  }  this.parties = mockParties.filter(party => party.categorie === category.name).sort(AppComponent.sortParties);  }  clicked(selectedCategory: Category): void {  this.categories = this.categories.map(category => {  category.selected = category === selectedCategory;  return category;  });  this.filterBooks(selectedCategory);  } |

**2b. On click on a category change the class to selected**

SUGGESTION

The css class should be applied to the anchor: use **[class.selected] or [ngClass]**

Change the following in **app.component.html:**

|  |
| --- |
| <li \*ngFor="let category of categories">  <a **[ngClass]="selCat==category.name ? 'selected' : ''"** href="#0" (click)="clicked(category)">{{category.name}}</a>  </li> |

Change the following in your **app.components.ts:**

|  |
| --- |
| clicked(cat : Category) {  **if** (cat.**name** == **'All'**) {  **this.selCat = cat.name;**  **this**.**parties** = ***mockParties***;  } **else** {  **this**.**parties** = ***mockParties***.filter(b => b.**category** === cat.**name**);  **this**.**selCat** = cat.**name**;  } } |

**3. On click on the filter button, open the sidebar filter component**

SUGGESTION:

Change the following in **app.component.html** in the gallery section.

<section class="gallery" [class.filter-is-visible]="!navClosed">

and update the section: *Sidebar navigation*

<div [ngClass]="!navClosed ? 'filter filter-is-visible': 'filter'">

Define an onclick event in your **app.components.ts**:

**(click)**="toggleSideBar**()**"

**4. Implement the search**

Add the following in **app.module.ts:**

|  |
| --- |
| **import { FormsModule } from '@angular/forms';**  @NgModule({  imports: [ BrowserModule, **FormsModule** ], // Inject built-in modules  declarations: [ AppComponent ], // Inject your own modules  bootstrap: [ AppComponent ] // Module you need to bootstrap  }) |

Change the following in **app.component.html** in the section: *Sidebar navigation*

|  |
| --- |
| <form (submit)="search()">  <input class="form-control" type="text" id="search" required  [(ngModel)]="searchString" name="searchString" placeholder="Title or category"> </form> |

Add the instance variable “searchString” to the AppComponent class, and implement the search method in **app.components.ts**:

|  |
| --- |
| search() {  console.log(**this**.searchString);   **this**.parties = mockParties.filter(b => b.trefwoorden.find(  trefwoord => trefwoord.toLowerCase().includes(  **this**.searchString.toLowerCase()))  || b.titel.toLowerCase().includes(**this**.searchString.toLowerCase())  ).sort(AppComponent.*sortParties*); } |

**5. Define a service with promises (for retrieving the mockParties and categories)**

Create a new folder: **app/services**

Add a service in this folder: **app.service.ts**

|  |
| --- |
| **import** {Injectable} **from** '@angular/core'; **import** {mockParties, Party} **from** '../mocks/parties'; **import** {mockCategories, Category} **from** '../mocks/categories';  @Injectable() **export class** AppService {  getParties(): Promise<Party[]> {  **return** Promise.resolve(mockParties);  }   getCategories(): Promise<Category[]> {  **return** Promise.resolve(mockCategories);  } } |

To make this AppService injectable, we need to create a provider for it first. This can be done either in the module and the component

Import and define the service as a provider in **app.module.ts**

|  |
| --- |
| @NgModule({  imports: [BrowserModule, FormsModule], // Inject built-in modules  declarations: [AppComponent],  providers: [AppService],  bootstrap: [AppComponent] // Module you need to bootstrap }) |

Import this new service in **app.component.ts:**

|  |
| --- |
| **import** { AppService } **from './services/app.service'**; |

Inject an instance of the Service via the constructor of the AppComponent:

|  |
| --- |
| **constructor**(**private** appService: AppService) {  // empty constructor } |

Redefine parties and categories in app.component.ts, and implement the OnInit interface:

|  |
| --- |
| **export class** AppComponent **implements** OnInit {  **parties**: Party [] ;  **categories**: Category[]; |

Get the mockdata via the lifecycle hook **ngOnInit()** :

Make use of the AppService:

|  |
| --- |
| **public** ngOnInit() {  **this**.appService.getParties().then((parties) => {  **this**.parties = parties.sort(AppComponent.*sortParties*);  });   **this**.appService.getCategories().then(categories => {  **this**.categories = categories;  }); } |

**6. Architecture – refactor application into components**

**Which components do you recognize?**

* HeaderComponent
* PartyListComponent
* MenuComponent
* SideBarComponent

Of these components, the header component contains only static HTML. So we can leave that component as it is, inside the **app.component.html.**

**6a. Steps to create a *PartyListComponent*:**

* Create a new folder: app/components/party
* Add new template for the partylist in this folder:

partylist.component.html

* + Copy the html for showing a partylist to: Partylist.component.html
  + Create a new PartyListComponent in this folder

partylist.component.ts

* + Replace the html in **app.component.html**

|  |
| --- |
| <app-party-list [parties]="parties" [sidebarVisible]="sidebarVisible"></app-party-list> |

* + Import **and** declare the PartyListComponent in app.module.ts

The PartyListComponent:

|  |
| --- |
| @Component({  moduleId: module.id,  selector: 'app-party-list',  templateUrl: 'partylist.component.html' }) **export class** PartyListComponent {   @Input() parties: Party[];  @Input() sidebarVisible: **boolean**;   clicked(party: Party) {  console.log(`Klik.. ${party.titel}`);  } } |

**6b. Steps to create a *MenuComponent*:**

* Create a new folder: app/components/menu
* Add new template for the menu in this folder:

menu.component.html

* + Copy the html for showing the menu to: menu.component.html
  + Create a new MenuComponent in this folder

menu.component.ts

* + Replace the html in **app.component.html**

|  |
| --- |
| <app-menu [categories]="categories" (categoryChanged)="clicked($event)"></app-menu> |

* + Import **and** declare the MenuComponent in app.module.ts

The MenuComponent:

|  |
| --- |
| @Component({  moduleId: module.id,  selector: 'app-menu',  templateUrl: 'menu.component.html' }) **export class** MenuComponent {   @Input() categories: Category[];  @Output() categoryChanged: EventEmitter<Category> = **new** EventEmitter<Category>();   changeCategory(category): **void** {  **this**.categoryChanged.emit(category);  category.selected = **true**;  }  } |

**6c. Steps to create a *SidebarComponent*:**

* Create a new folder: app/components/sidebar
* Add new template for the menu in this folder:

sidebar.component.html

* + Copy the html for showing the sidebar to: sidebar.component.html
  + Create a new SidebarComponent in this folder

sidebar.component.ts

* + Replace the html in **app.component.html**

|  |
| --- |
| <app-sidebar (searchParty)="handleSearch($event)" (sidebarToggled)="handleSidebarToggle()" [sidebarVisible]="sidebarVisible"></app-sidebar> |

* + Import **and** declare the SidebarComponent in app.module.ts

The SidebarComponent:

|  |
| --- |
| @Component({  moduleId: module.id,  selector: 'app-sidebar',  templateUrl: 'sidebar.component.html' }) **export class** SidebarComponent {   searchString: **string**;   @Output() searchParty: EventEmitter<**string**> = **new** EventEmitter<**string**>();  @Output() sidebarToggled: EventEmitter<**any**> = **new** EventEmitter();  @Input() sidebarVisible: **boolean**;   search() {  **this**.searchParty.emit(**this**.searchString);  }   toggleSideBar() {  **this**.sidebarToggled.emit();  }  } |

**7. Bonus exercise – reimplement the service using the HttpModule**

Of course no one would implement the service using a promise and resolve it immediately.

A lot more useful implementation would be to:

* Reimplement the service using the Angular HttpModule
* Convert const mockParties: Party [] to a JSON file, and place it in the assets folder
* Find out the URL to retrieve the JSON-file
* Use http.get() to return Observable<Party[]>
* Convert the response using rxjs/add/operator/map