
we build better, not more

# Angular:
# Routing and navigation, Forms

Petar Nacevski


# About me

- Full stack developer at MCA
- pna@mca.dev
- www.linkedin.com/in/petar-nacevski-94490b248

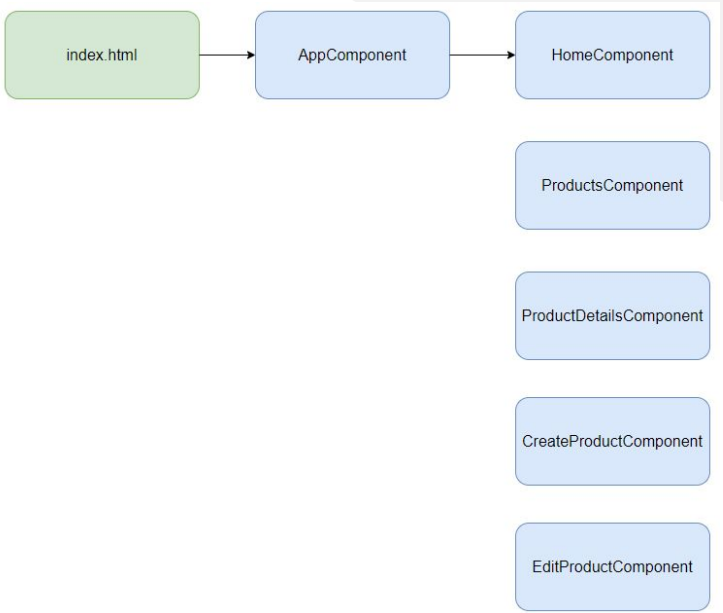
MCA

# Routing and Navigation

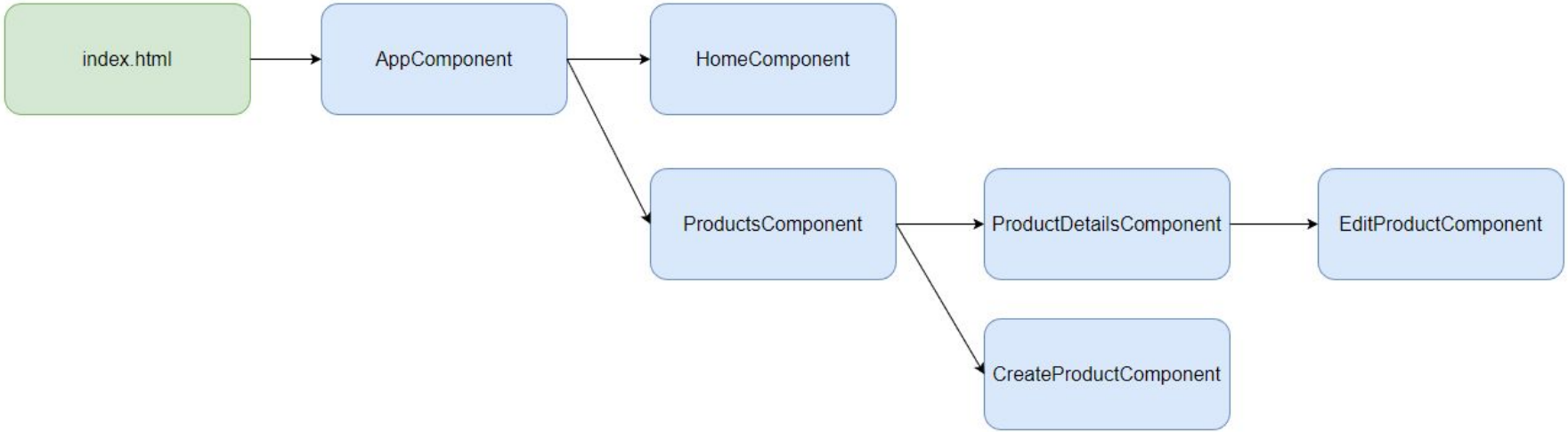

# Routing and Navigation

- How does routing work?
- Tying routes to actions
- Configuring routes
- Placing the view

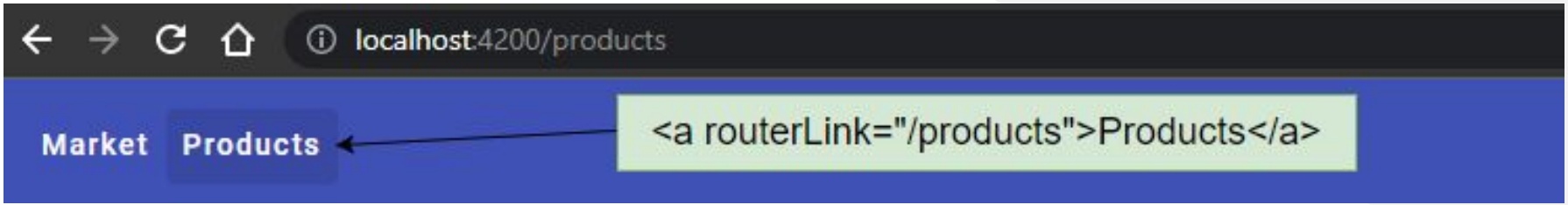
# Application Architecture (currently)



# Application Architecture (final)



# Tying routes to actions



# Configuring routes

App.Module.ts

```
@NgModule({
  imports: [
    RouterModule.forRoot([
      { path: 'products/:id', component: ProductDetailsComponent },
      { path: 'products', component: ProductsComponent },
      { path: 'home', component: HomeComponent },
      { path: '', redirectTo: 'home', pathMatch: 'full' },
      { path: '**', redirectTo: 'home', pathMatch: 'full' }
    ])
  ],
  declarations: [ ···
  ],
  providers: [ ···
  ],
  bootstrap: [ ···
  ]
})
export class AppModule { }
```

# Placing the views

App.component.ts

```
<mat-toolbar color="primary">
  <a mat-button routerLink="/">Market</a>
  <a mat-button routerLink="/products">Products</a>
</mat-toolbar>
<div class="content">
  <router-outlet></router-outlet>
</div>
```

# Passing parameters to a route

App.Module.ts

```
@NgModule({
  imports: [
    RouterModule.forRoot([
      { path: 'products/:id', component: ProductDetailsComponent },
      { path: 'products', component: ProductsComponent },
      { path: 'home', component: HomeComponent },
      { path: '', redirectTo: 'home', pathMatch: 'full' },
      { path: '**', redirectTo: 'home', pathMatch: 'full' }
    ])
  ],
  declarations: [ ···
  ],
  providers: [ ···
  ],
  bootstrap: [ ···
  ]
})
export class AppModule { }
```

# Passing parameters to a route

products.component.ts

```
<a [routerLink]="['/products', product.id]">{{ product.title }}</a>
```


# Reading Parameters from a Route

product-details.component.ts

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

    constructor(private route: ActivatedRoute) { }
```

# Reading Parameters from a Route

**Snapshot : Read the parameter one time**

```
this.route.snapshot.paramMap.get('id');
```

**Observable: Read emitted parameters as they change**

```
this.route.paramMap.subscribe(
  params => console.log(params.get('id'))
);
```

**Specified string is the route parameter name**

```
{ path: 'products/:id',
  component: ProductDetailComponent }
```

# Activating a route through code

product-details.component.ts

```
import { Router } from '@angular/router';
...
    constructor(private router: Router) { }

    onBack(): void {
      this.router.navigate(['/products']);
    }
```

MCA

# Protecting Routes with Guards

Limit access to a route

Restrict access to only certain users

Require confirmation before navigating away

# Guard use cases

**CanActivate**

- Guard navigation to a route

**CanDeactivate**

- Guard navigation from a route

**Resolve**

- Pre-fetch data before activating a route

**CanLoad**

- Prevent asynchronous routing

# Guard

product.guard.ts

```
export const canActivateTeam: CanActivateFn =
  (route: ActivatedRouteSnapshot, state: RouterStateSnapshot) => { ···
  };
```

# Guard

product.guard.ts DEPRECATED!!!

```
@Injectable({
  providedIn: 'root'
})
export class ProjectGuard implements canActivate {
  public canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot)
    : Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree {…
  }

}
```

DEPRECATED! >=v14

MCA

# Forms

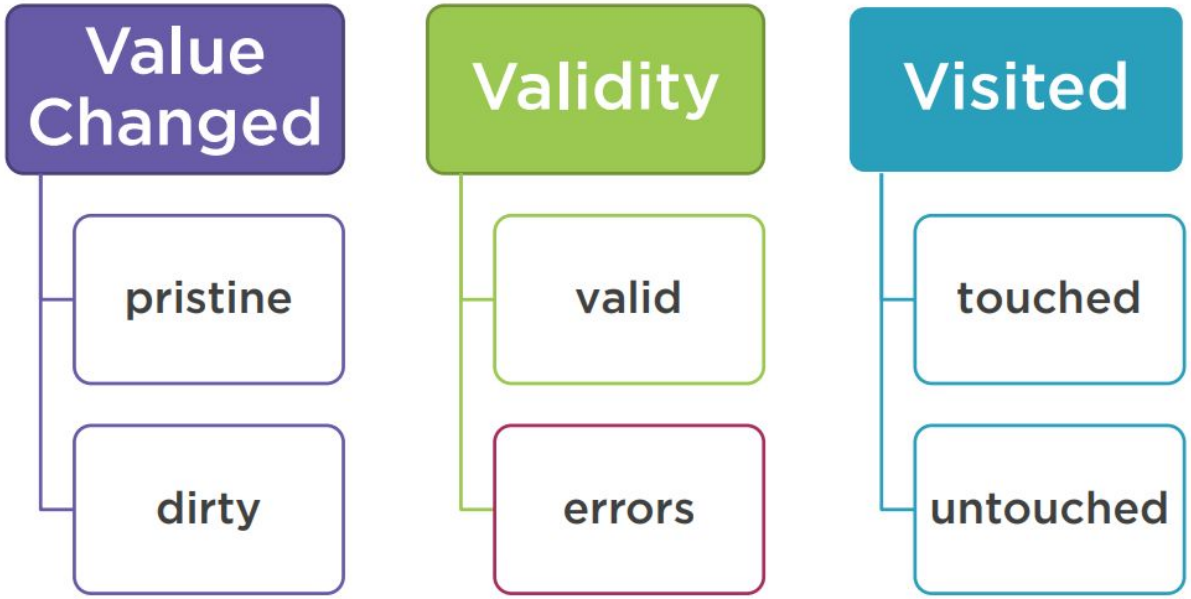

REPORT OF ADOPTION
ATTORNEY
CLERK OF THE COURT
ADOPT-215
ADOPT-200
Adoption Request

MCA

# Angular Forms

| Template-driven | Reactive |
| --- | --- |
| Generated form model | Manually created form model |
| HTML validation | Validation in the class |
| Two-way data binding | No two-way data binding |

# States


Value Changed
pristine
dirty
Validity
valid
errors
Visited
touched
untouched

# Template-driven Forms

app.module.ts

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

...
@NgModule({
  imports: [
    FormsModule
  ],
  providers: [ ···
  ],
  bootstrap: [ ···
  ]
})
export class AppModule { }
```


# Using ngForm

**form.component.ts**

```
> <form #form="ngForm"> ···
  </form>
```

# Using ngModel

form.component.ts

```
<form #form="ngForm">
  <input type="text" ngModel name="title"/>
  <input type="number" ngModel name="price"/>
</form>
```

# Two way data binding

component.html

```
<form #form="ngForm">
  <input type="text" [(ngModel)]="titleProperty" name="title"/>
</form>

{{ titleProperty }}
```

component.ts

```
export class CreateProductComponent {

  public titleProperty = '';
```

# Template-driven Form with Angular Material form-fields

form.component.ts

```
<form #form="ngForm">
  <mat-form-field class="full-width" appearance="outline">
    <mat-label>Title</mat-label>
    <input matInput name="title" ngModel/>
  </mat-form-field>
</form>
```

# Validations

form.component.ts

```
<form #form="ngForm">
  <mat-form-field class="full-width" appearance="outline">
    <mat-label>Title</mat-label>
    <input matInput name="title" ngModel required/>
  </mat-form-field>
</form>
```


# Built in validators

https://angular.io/api/forms/Validators

```
class Validators {
  static min(min: number): ValidatorFn
  static max(max: number): ValidatorFn
  static required(control: AbstractControl<any, any>): ValidationErrors | null
  static requiredTrue(control: AbstractControl<any, any>): ValidationErrors | null
  static email(control: AbstractControl<any, any>): ValidationErrors | null
  static minLength(minLength: number): ValidatorFn
  static maxLength(maxLength: number): ValidatorFn
  static pattern(pattern: string | RegExp): ValidatorFn
  static nullValidator(control: AbstractControl<any, any>): ValidationErrors | null
  static compose(validators: ValidatorFn[]): ValidatorFn | null
  static composeAsync(validators: AsyncValidatorFn[]): AsyncValidatorFn | null
}
```

# Displaying validation error messages

form.component.ts

```
<form #form="ngForm">
  <mat-form-field class="full-width" appearance="outline">
    <mat-label>Title</mat-label>
    <input matInput name="title" ngModel required #title="ngModel"/>
  </mat-form-field>
  <div *ngIf="!!title.errors && (title.touched || !title.pristine)"class="errors">
    <mat-error *ngIf="title.errors['required']">Title is a required field.</mat-error>
  </div>
</form>
```

# Custom Validators - Template-driven Forms

*ng generate directive {relative-path-from-app-folder}/{validator-name}*

validator.directive.ts

```
@Directive({
  selector: '[price]',
  providers: [
    { provide: NG_VALIDATORS, useExisting: PriceValidatorDirective, multi: true }
  ]
})
export class PriceValidatorDirective implements Validator {

  validate(control: FormControl) {…
  }
}
```

# Custom Validators - Template-driven Forms

form.component.ts

```
<mat-form-field class="full-width" appearance="outline">
  <mat-label>Price</mat-label>
  <input type="number" matInput name="price" ngModel #price="ngModel" required price/>
</mat-form-field>
<div *ngIf="!!price.errors && (price.touched || !price.pristine)" class="errors">
  <mat-error *ngIf="price.errors['required']">Price is a required field.</mat-error>
  <mat-error *ngIf="price.errors['price']">Price cannot be 0 or lower.</mat-error>
</div>
```

```
(directive) AppModule.PriceValidatorDirective
```

# Submit form (Template-driven Forms)

component.html

```html
<form class="form" #form="ngForm" (ngSubmit)="submit(form.value)">

  <div class="actions">
    <button mat-stroked-button>Submit</button>
  </div>
</form>
```

component.ts

```ts
public submit(value: ProductRequest): void {

  // POST
  console.log('Create product', value);

  this.router.navigateByUrl(`/products`);
}
```


# Reactive Forms

app.module.ts

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

...
@NgModule({
  imports: [
    ReactiveFormsModule
  ],
  providers: [

  ],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule { }
```

# Creating FormGroup & FormControl

component.ts

```ts
export class EditProductComponent implements OnInit {
  public formGroup!: FormGroup;
  public descriptionControl!: FormControl;

  public ngOnInit(): void {
    this.descriptionControl = new FormControl('', []);

    this.formGroup = new FormGroup({
      title: new FormControl('', []),
      description: this.descriptionControl
    })
  }
```

# Using FormGroup & FormControl

**component.ts**

```
<form [formGroup]="formGroup">
    <input type="number" matInput formControlName="title" />
    <input type="number" matInput [formControl]="descriptionControl" />
</form>
```

MCA

# FormBuilder

Creates a form model from a configuration

Shortens boilerplate code

Provided as a service

# FormBuilder

component.ts

```ts
export class EditProductComponent implements OnInit {
  public formGroup?: FormGroup;

  public constructor(private formBuilder: FormBuilder) { }

  public ngOnInit(): void {

    this.formGroup = this.formBuilder.group({
      title: [{value: '', disabled: false}, []],
      description: ['', []],
    });
  }
```

# Validators (Reactive Forms)

custom-validators.ts

```
export class EditProductComponent implements OnInit {
  public formGroup?: FormGroup;

  public constructor(private formBuilder: FormBuilder) {}
  public ngOnInit(): void {

    this.formGroup = this.formBuilder.group({
      title: ['', [Validators.required]]
    })
  }

}
```

# Custom Validators (Reactive Forms)

custom-validators.ts

```
export class CustomValidators {

  public static priceValidator(control: AbstractControl) : ValidationErrors | null {···
  }

}
```

component.ts

```
this.formGroup = this.formBuilder.group({
  title: ['', [Validators.required]],
  price: [null, [Validators.required, CustomValidators.priceValidator]]
})
```

# Displaying validation error messages

component.html

```
<form [formGroup]="formGroup" *ngIf="formGroup">
  <mat-form-field class="full-width" appearance="outline">
    <mat-label>Price</mat-label>
    <input type="number" matInput formControlName="price" />
  </mat-form-field>
  <div *ngIf="!!formGroup.controls['price'].errors && (formGroup.controls['price'].touched || !formGroup.controls['price'].pristine)" class="errors">
    <mat-error *ngIf="formGroup.controls['price'].errors['required']">Price is a required field.</mat-error>
    <mat-error *ngIf="formGroup.controls['price'].errors['price']">Price cannot be 0 or lower.</mat-error>
  </div>
</form>
```

# Submit form (Reactive Forms)

component.html

```
<button mat-stroked-button (click)="submit()">Submit</button>
```

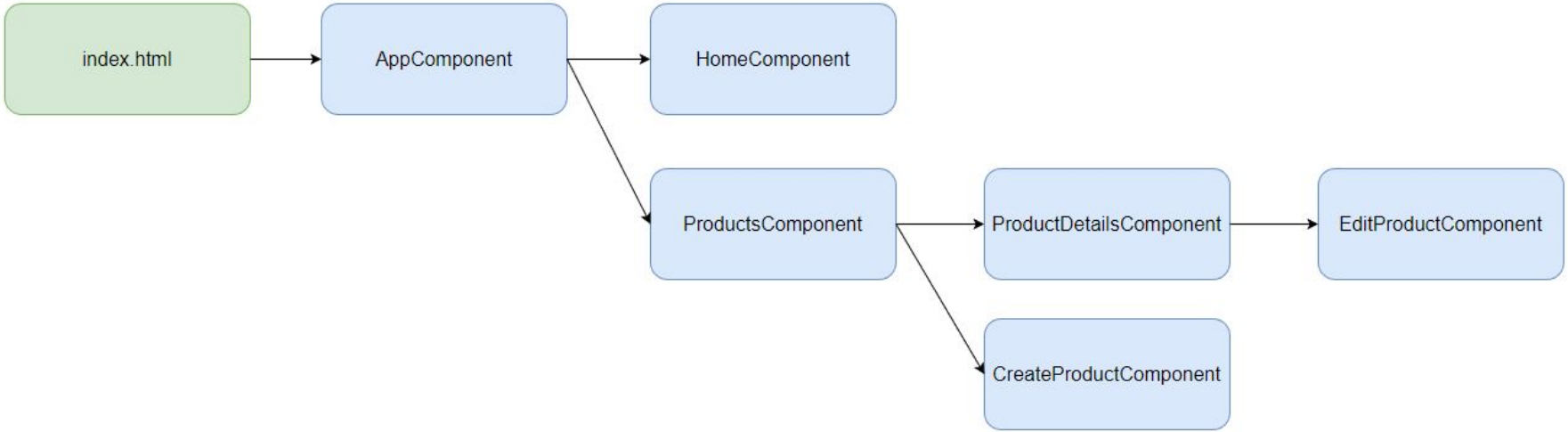
component.ts

```
public submit(): void {
  const value: ProductRequest = this.formGroup?.value;

  // POST call
  console.log('Create product', value);

  this.router.navigateByUrl(`/products/${this.product!.id}`);
}
```

# Application Architecture (final)




# Questions?