Angular Reactive Forms

INTRODUCTION

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/


ONLINE REGISTRATION FORM
First Name
Last Name
Gender
Date of Birth
Age
Country
Place of Birth
Zip Code
Phone
email
Current adress
Reason for the application
SELECT YOUR COUNTRY OF ORIGIN
Botswana
Brazil
Britain
Brunei
Bulgaria
Burkina
Burma
Burundi
Burma
Burundi
Cambodia
Cameroon
Canada
search
.com
123/?

Module Overview

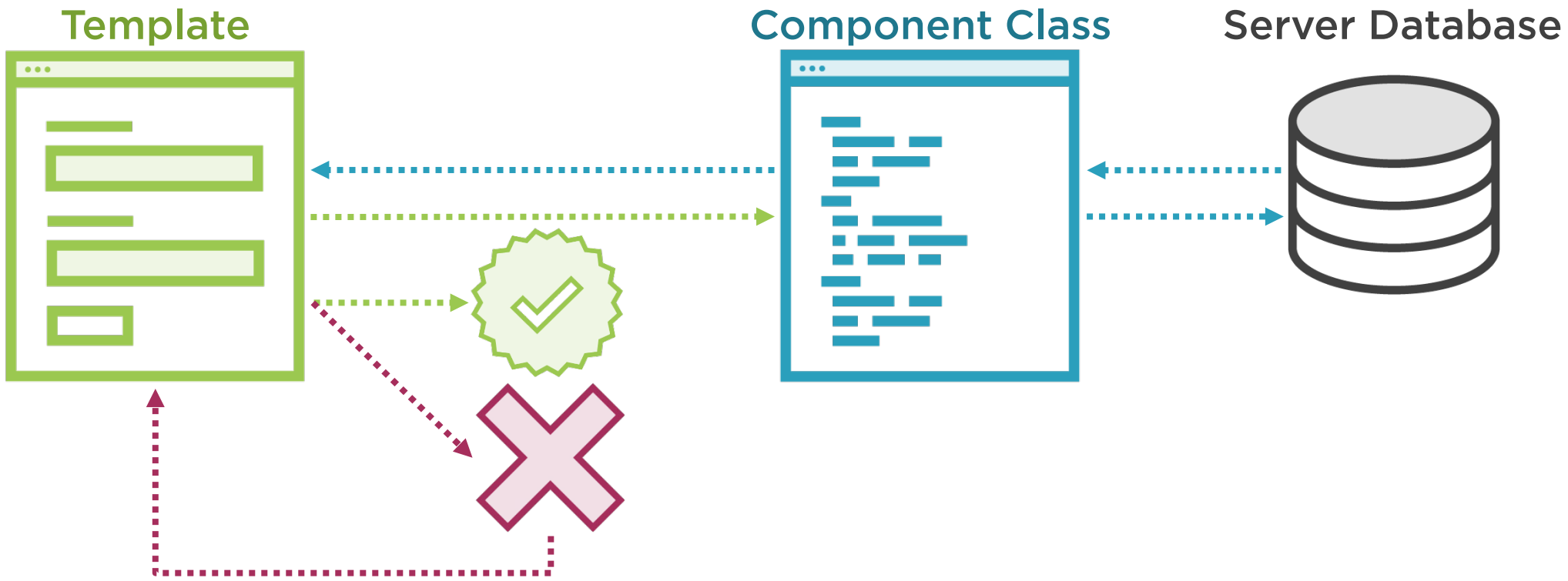
Angular Forms

Getting the Most from This Course

Demo Form and Sample Application

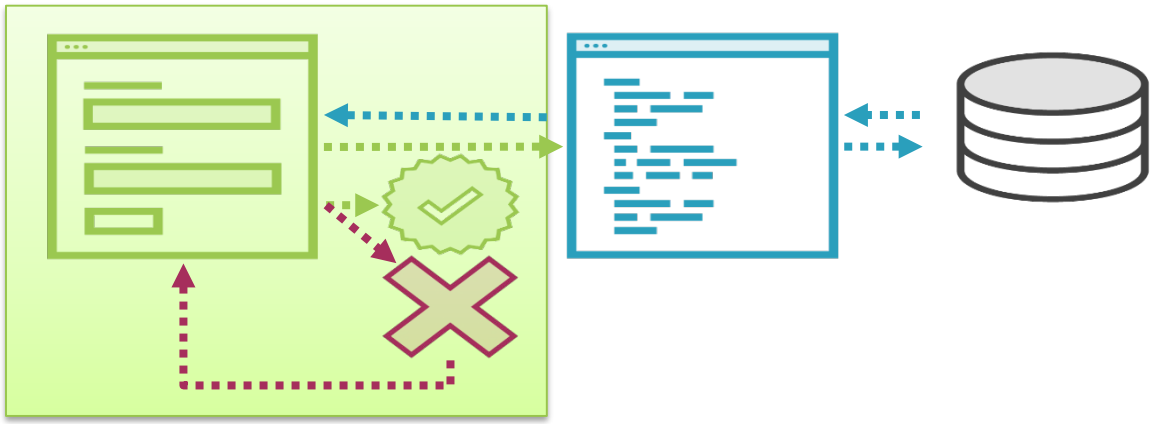
Course Outline

Angular Forms

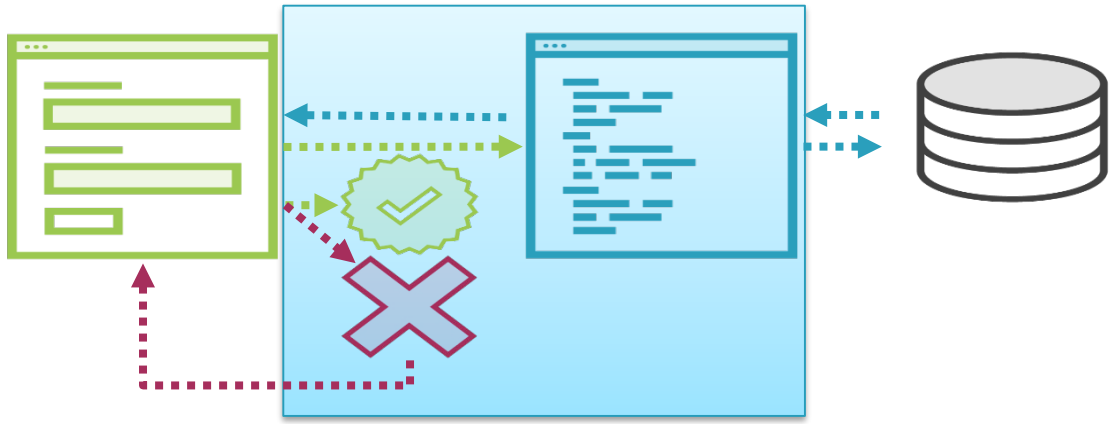


Angular Forms

Template-driven



Reactive
(Model-driven)



Angular Forms

Template-driven

Easy to use

Similar to AngularJS

Two-way data binding ->
Minimal component code

Automatically tracks form and
input element state

Reactive

More flexible ->
more complex scenarios

Immutable data model

Easier to perform an action
on a value change

Reactive transformations ->
DebounceTime or DistinctUntilChanged

Easily add input elements dynamically

Easier unit testing

Prerequisites

Required

- Angular modules (NgModule)
- Components
- Templates
- Binding
- Services
- Routing

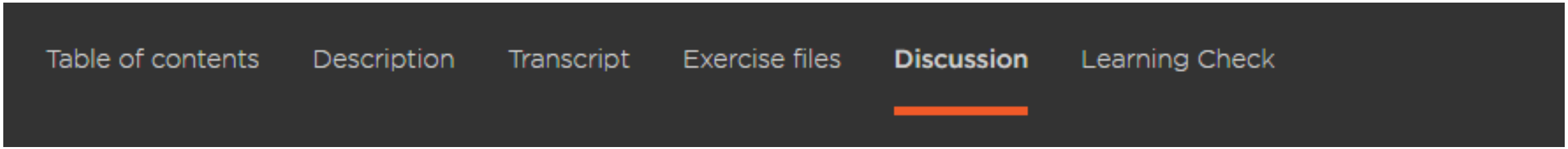
Suggested

- Angular: Getting Started
- Angular: First Look

Not Required

- Prior Angular forms knowledge

Thoughts? Comments? Questions?



@deborahkurata

Blog Post

http://blogs.msmvps.com/deborahk/angular-2-reactive-forms-problem-solver/

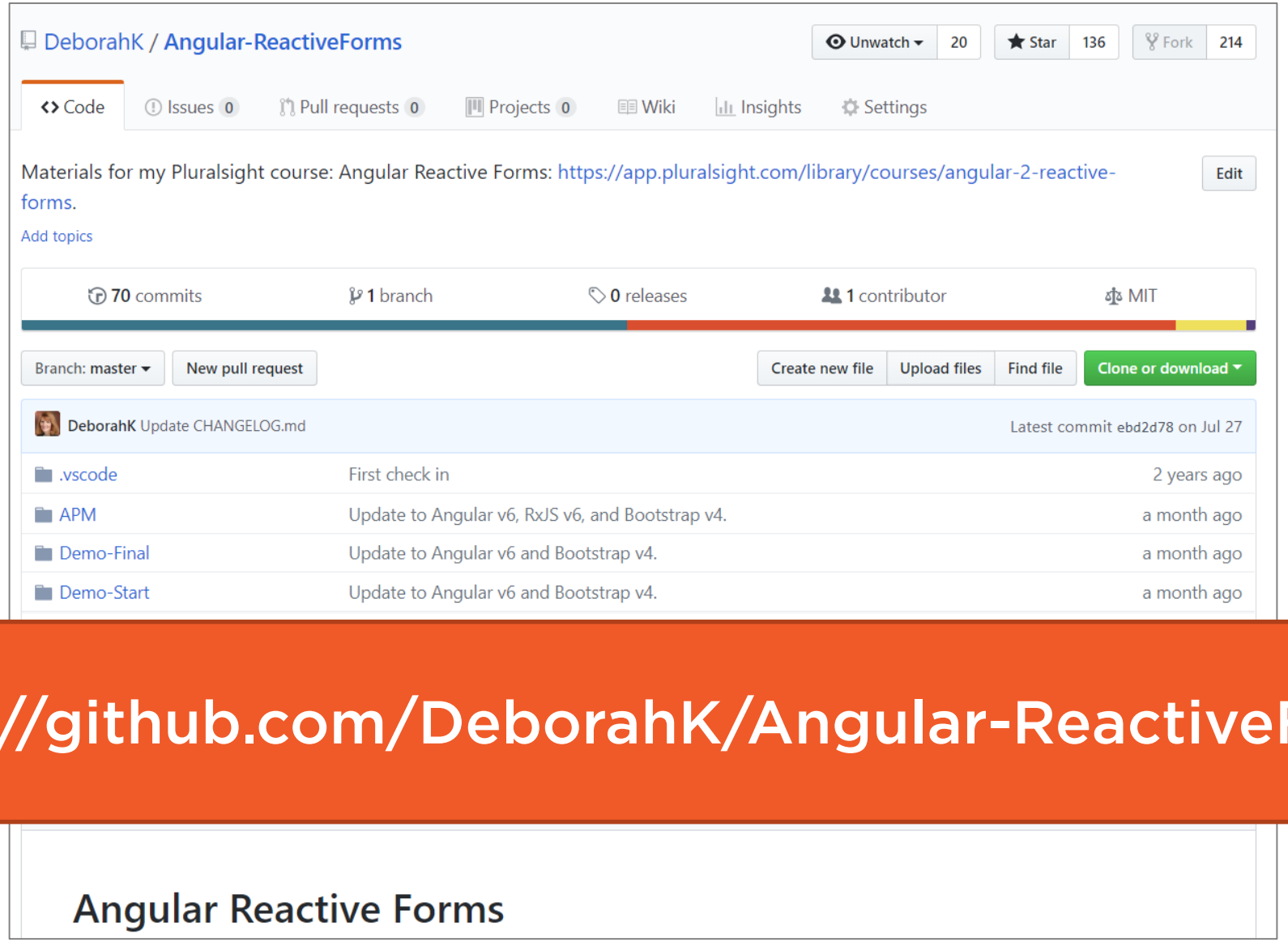
Checklist

Review module concepts

Code along assistance

Revisit as you build

GitHub Repository



https://github.com/DeborahK/Angular-ReactiveForms

Demo Form

Sign Up!

First Name — First Name (required)

Last Name — Last Name (required)

Email — Email (required)

Send me your catalog

Address Type — Home Work Other

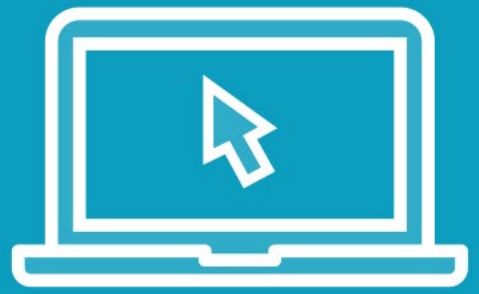
Street Address 1 — Street address

Street Address 2 — Street address (second line)

City, State, Zip Code — City | Select a State... | Zip Code

Save

Demo



Sample Application in Action

- Reactive form
- Routing
- Data Access Service

Course Outline

- Template-driven vs. Reactive Forms
- Building a Reactive Form
- Validation
- Reacting to Changes
- Dynamically Duplicate Input Elements
- Reactive Form in Context
- Create, Read, Update and Delete (CRUD) Using HTTP

Template-driven vs. Reactive Forms

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/


Template-driven
Reactive

Module Overview

Angular Form Building Blocks

- FormGroup
- FormControl

Template Syntax for Forms

Template-driven Forms

Complex Scenarios

Sign Up!

First Name — First Name (required)

Last Name — Last Name (required)

Email — Email (required)

☑ Send me your catalog

Address Type — ◉ Home ○ Work ○ Other

Street Address 1 — Street address

Street Address 2 — Street address (second line)

City, State, Zip Code — City | Select a State... | Zip Code

Save

Sign Up!

First Name

First Name (required)

Please enter your first name.

Last Name

Last Name (required)

Email

Email (required)

Send me your catalog

Address Type

Home Work Other

Street Address 1

Street address

Street Address 2

Street address (second line)

City, State, Zip Code

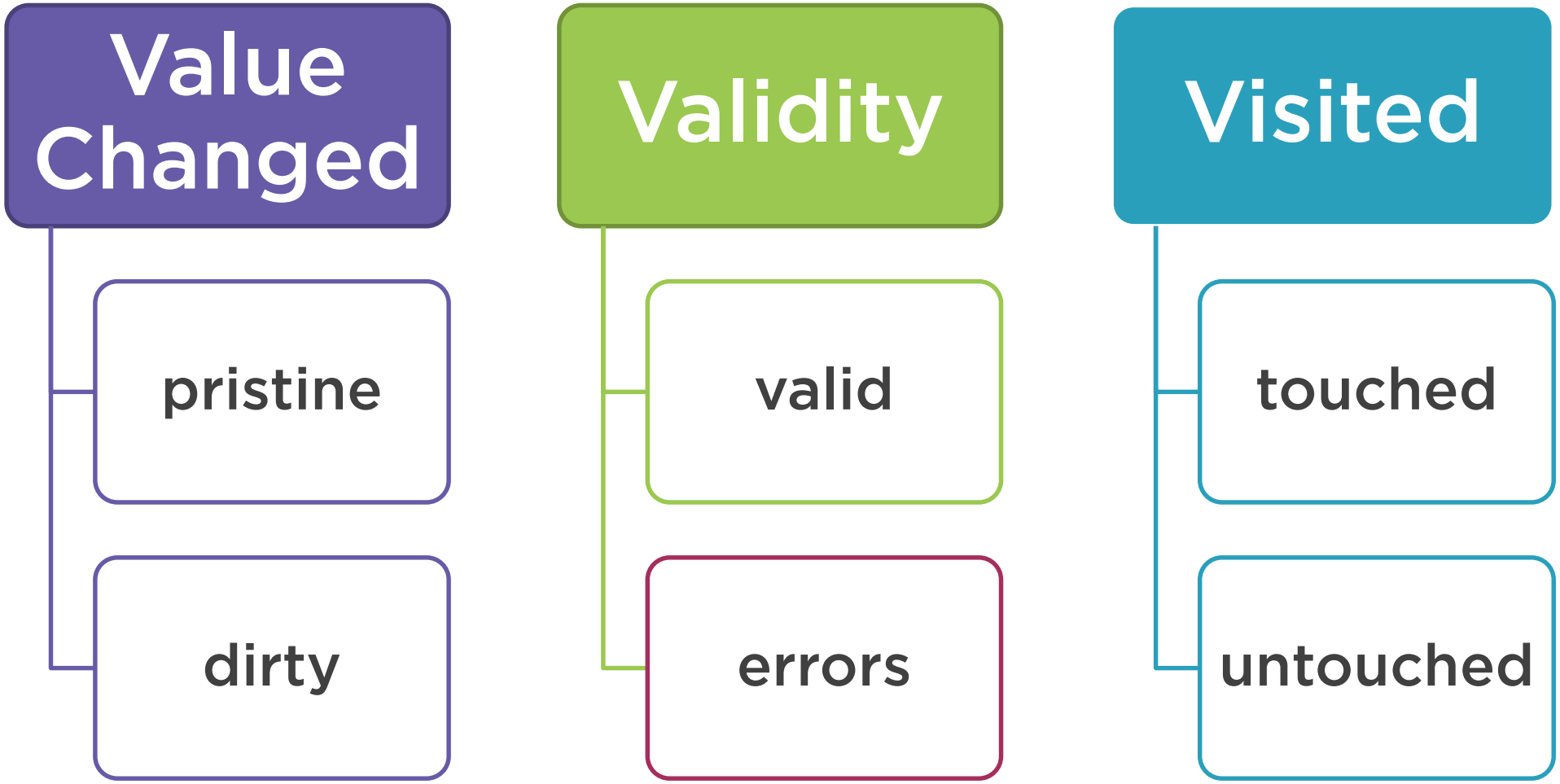
City

Select a State...

Zip Code

Save

State


Value Changed
pristine
dirty
Validity
valid
errors
Visited
touched
untouched

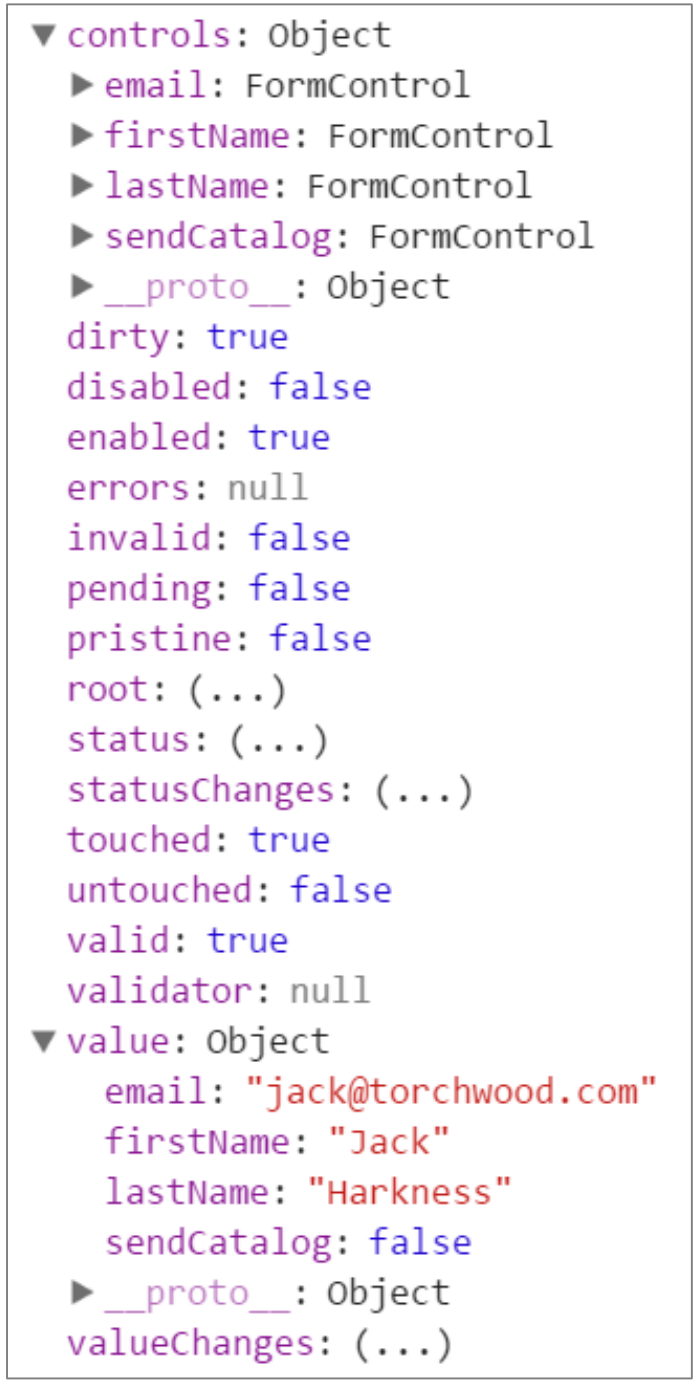
Form Building Blocks

FormControl

```
<input id="firstNameId"
       type="text"
       placeholder="First Name"
       required
       minlength="3" />
```

FormGroup

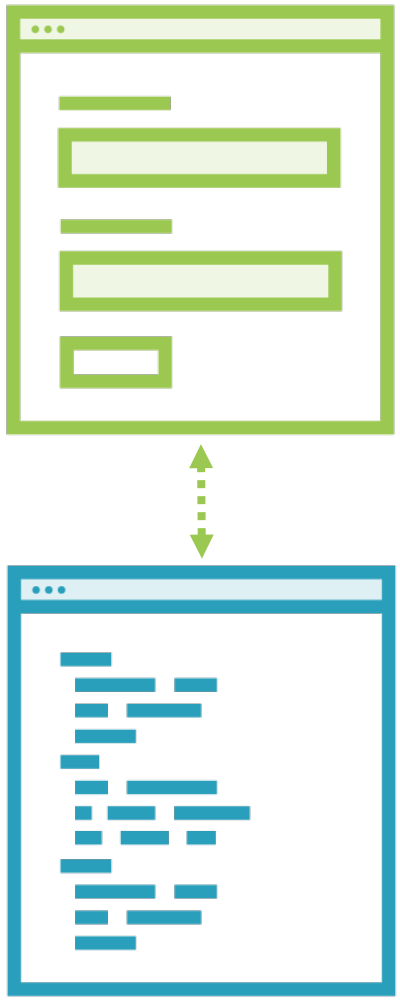
```
<form (ngSubmit)="save()">
    ...
</form>
```



Form Model

- Retains form state
- Retains form value
- Retains child controls
  - FormControls
  - Nested FormGroups

Template-driven Forms



```
▼ controls: Object
  ▶ email: FormControl
  ▶ firstName: FormControl
  ▶ lastName: FormControl
  ▶ sendCatalog: FormControl
  ▶ __proto__: Object
  dirty: true
  disabled: false
  enabled: true
  errors: null
  invalid: false
  pending: false
  pristine: false
```

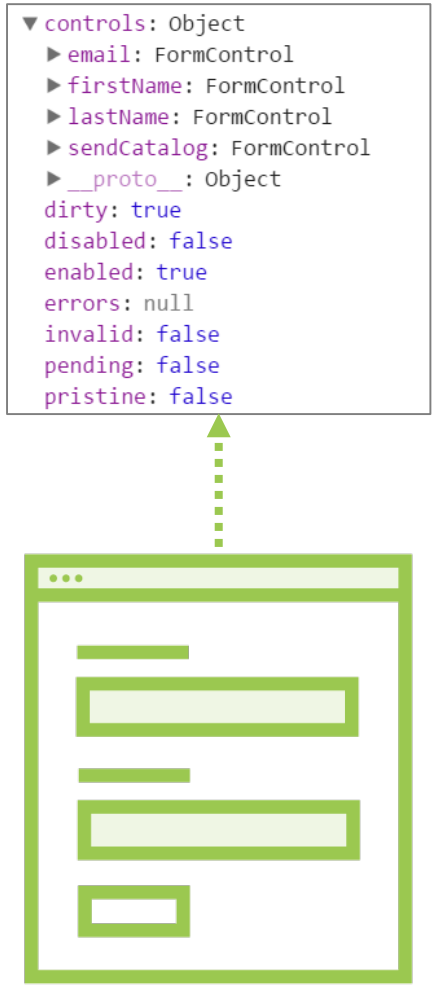
Template

- Form element
- Input element(s)
- Data binding
- Validation rules (attributes)
- Validation error messages
- Form model automatically generated

Component Class

- Properties for data binding (data model)
- Methods for form operations, such as submit

Reactive Forms



Component Class

- Form model
- Validation rules
- Validation error messages
- Properties for managing data (data model)
- Methods for form operations, such as submit

Template

- Form element
- Input element(s)
- Binding to form model

Directives

Template-driven (FormsModule)

- ngForm
- ngModel
- ngModelGroup

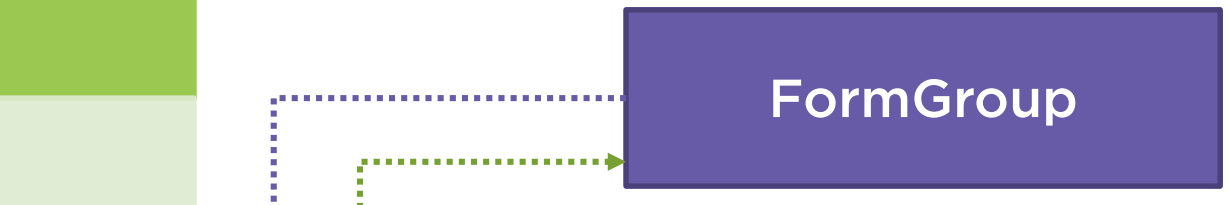
```
<form (ngSubmit)="save()">
</form>
```

FormGroup

Directives

Template-driven (FormsModule)

- ngForm
- ngModel
- ngModelGroup




```
<form (ngSubmit)="save()"
      #signupForm="ngForm">
</form>
```

Directives

Template-driven (FormsModule)

- ngForm
- ngModel
- ngModelGroup

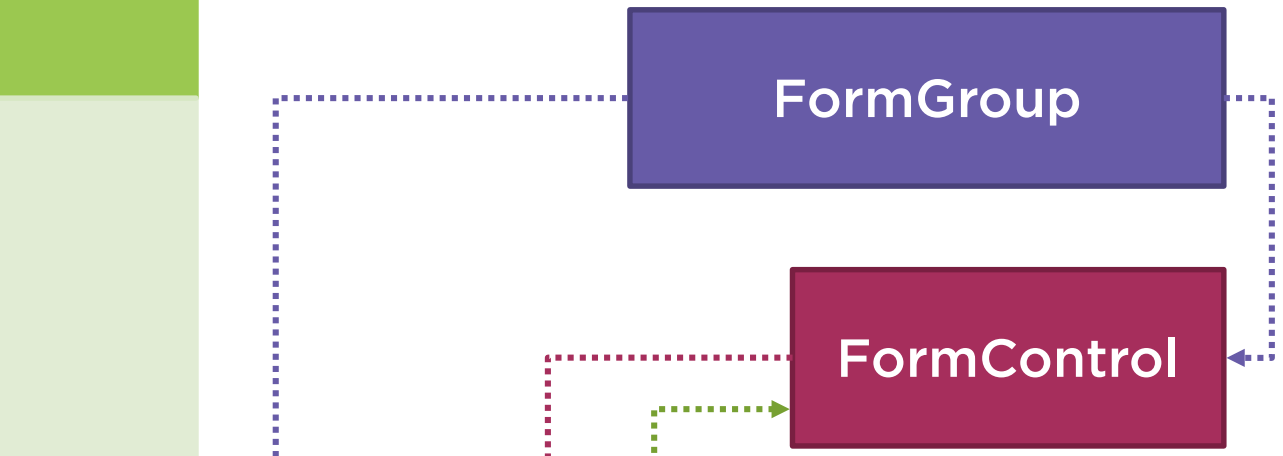
FormGroup

```
<form (ngSubmit)="save()"
      #signupForm="ngForm">
 <button type="submit"
         [disabled]="!signupForm.valid">
   Save
  </button>
</form>
```

Directives

Template-driven (FormsModule)

- ngForm
- ngModel
- ngModelGroup




```
<form (ngSubmit)="save()">
 <input id="firstNameId" type="text"
        [(ngModel)]="customer.firstName"
        name="firstName"
        #firstNameVar="ngModel" />
</form>
```

Directives

Template-driven (FormsModule)	Reactive (ReactiveFormsModule)
• ngForm • ngModel • ngModelGroup	• formGroup • formControl • formControlName • formGroupName • formArrayName

HTML Form

customer.component.html

```
<form>
  <div>
    <label for="firstNameId">First Name</label>
    <input id="firstNameId" type="text"
           placeholder="First Name (required)"
           required
           minlength="3" />
  </div>
  ...
  <button type="submit">Save</button>
</form>
```

Template-driven Form

customer.component.html

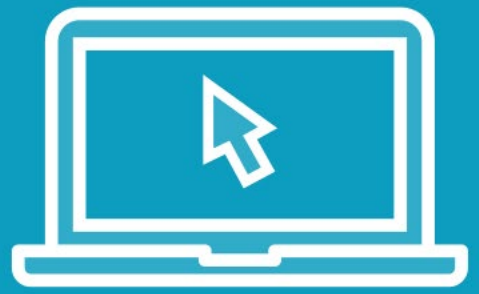
```
<form (ngSubmit)="save()">
  <div>
    <label for="firstNameId">First Name</label>
    <input id="firstNameId" type="text"
           placeholder="First Name (required)"
           required
           minlength="3"
           [(ngModel)]="customer.firstName"
           name="firstName"
           #firstNameVar="ngModel"
           [ngClass]="{'is-invalid': firstNameVar.touched && !firstNameVar.valid }" />
    <span *ngIf="firstNameVar.errors">
      Please enter your first name.
    </span>
  </div>
  ...
  <button type="submit">Save</button>
</form>
```

Reactive Form

customer.component.html

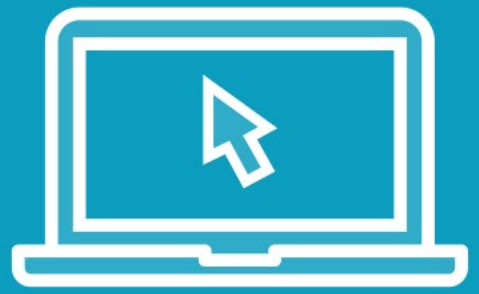
```
<form (ngSubmit)="save()" [formGroup]="signupForm">
  <div>
    <label for="firstNameId">First Name</label>
    <input id="firstNameId" type="text"
           placeholder="First Name (required)"
           formControlName="firstName"
           [ngClass]="{'is-invalid': formError.firstName }" />
    <span *ngIf="formError.firstName">
      {{formError.firstName}}
    </span>
  </div>
  ...
  <button type="submit">Save</button>
</form>
```

Demo



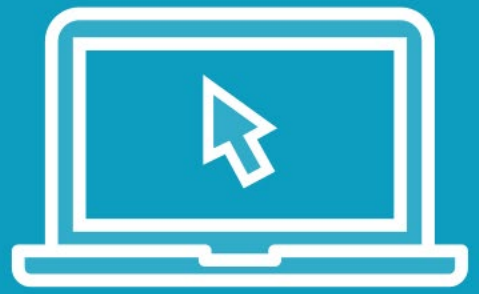
Template-driven Form

Demo



Template-driven Form: Template

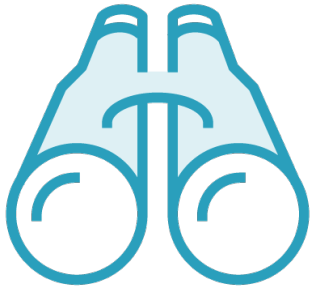
Demo



Template-driven Form: Component

Complex Scenarios

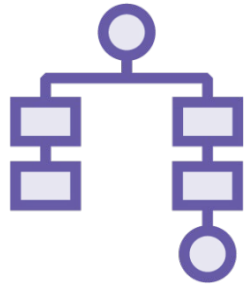
Dynamically add input elements



Watch what the user types

Wait validation until typing stops

Different validation for different situations



Immutable data structures

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

Summary

Angular Form Building Blocks

- FormGroup
- FormControl

Template Syntax for Forms

Template-driven Forms

Complex Scenarios

Building a Reactive Form

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/


GIVE
ME
MORE

Module Overview

The Component Class

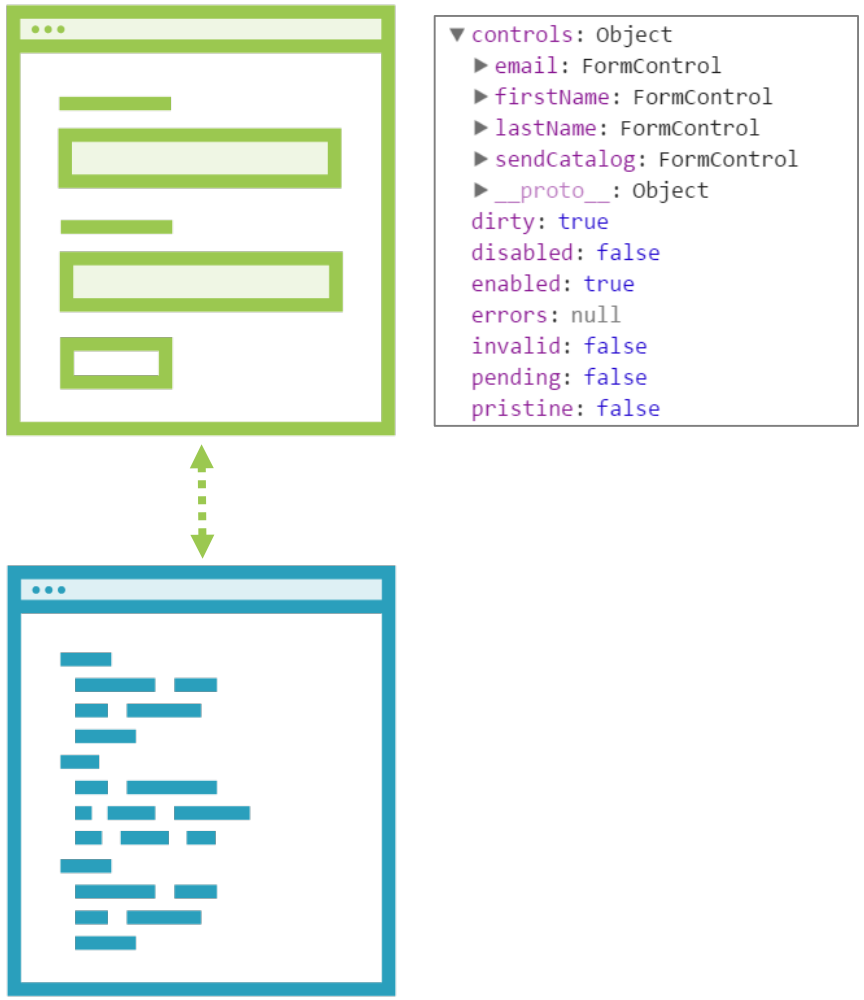
The Angular Module

The Template

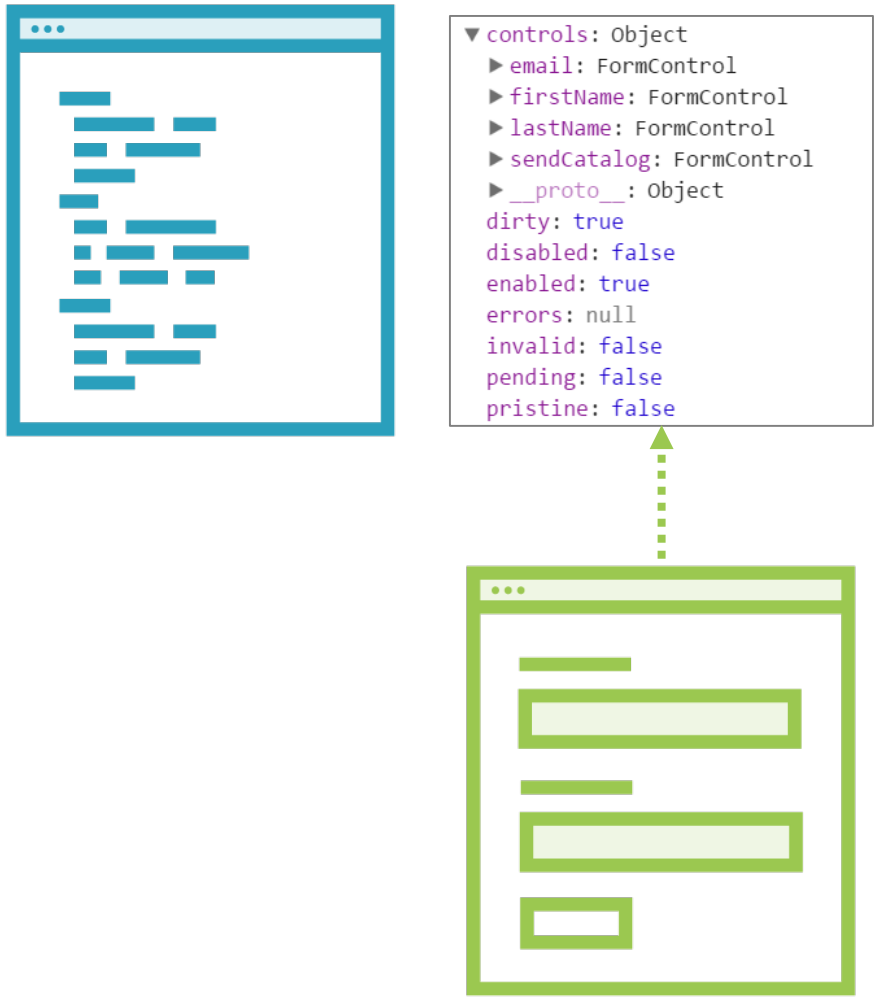
Using `setValue` and `patchValue`

Simplifying with FormBuilder

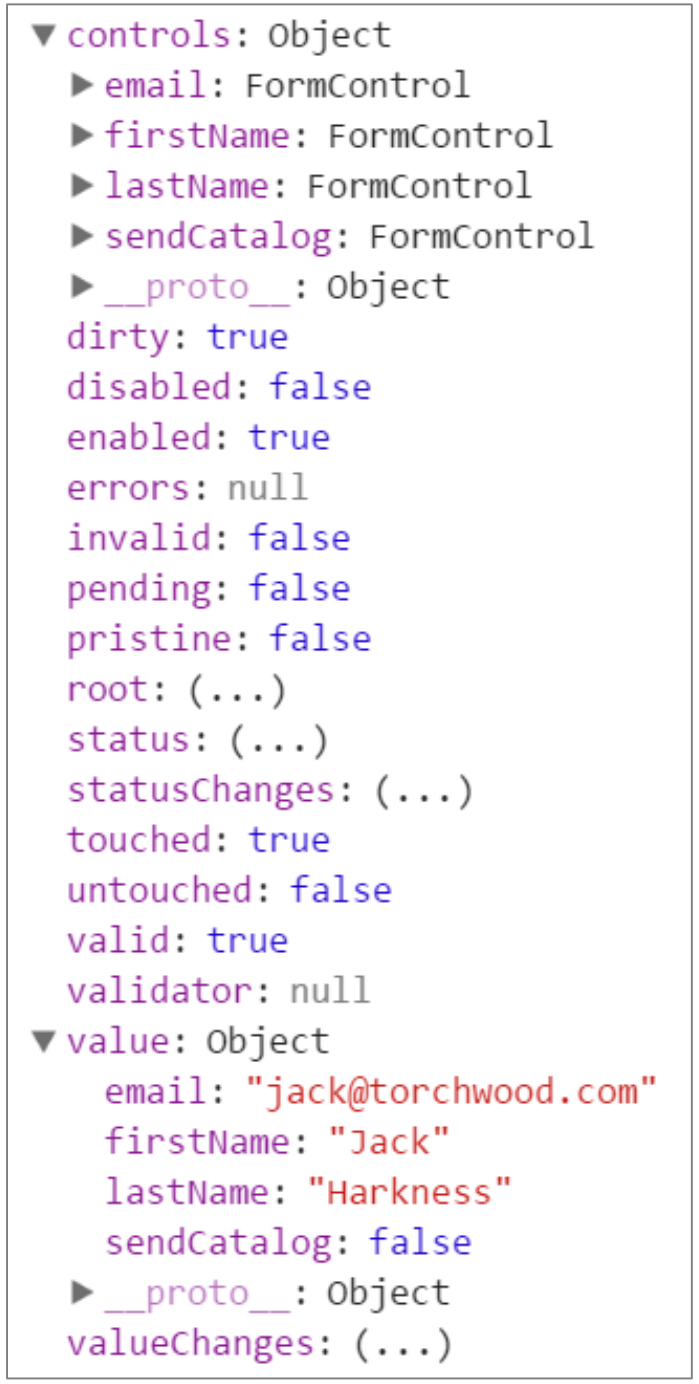
Template-driven vs. Reactive Forms



```
▼ controls: Object
  ▶ email: FormControl
  ▶ firstName: FormControl
  ▶ lastName: FormControl
  ▶ sendCatalog: FormControl
  ▶ __proto__: Object
  dirty: true
  disabled: false
  enabled: true
  errors: null
  invalid: false
  pending: false
  pristine: false
```



```
▼ controls: Object
  ▶ email: FormControl
  ▶ firstName: FormControl
  ▶ lastName: FormControl
  ▶ sendCatalog: FormControl
  ▶ __proto__: Object
  dirty: true
  disabled: false
  enabled: true
  errors: null
  invalid: false
  pending: false
  pristine: false
```



Form Model

- Root FormGroup
- FormControl for each input element
- Nested FormGroups as desired
- FormArrays

Creating a FormGroup

customer.component.ts

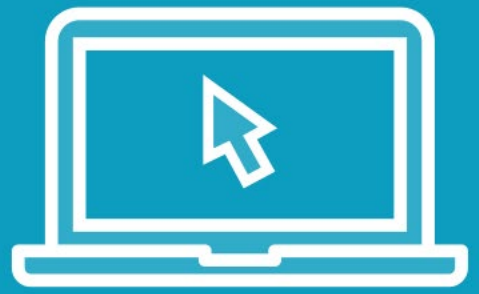
```
...
import { FormGroup } from '@angular/forms';


...
export class CustomerComponent implements OnInit {
  customerForm: FormGroup;
  customer: Customer = new Customer();

  ngOnInit(): void {
    this.customerForm = new FormGroup({ });
  }
}
```

Creating FormControls

customer.component.ts

```
...
import { FormGroup, FormControl } from '@angular/forms';


...
export class CustomerComponent implements OnInit {
  ...
  ngOnInit(): void {
    this.customerForm = new FormGroup({
        firstName: new FormControl(),
        lastName: new FormControl(),
        email: new FormControl(),
        sendCatalog: new FormControl(true)});
  }
}
```

Demo



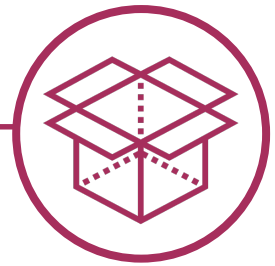
Building the Form Model

AppComponent

Customer-Component

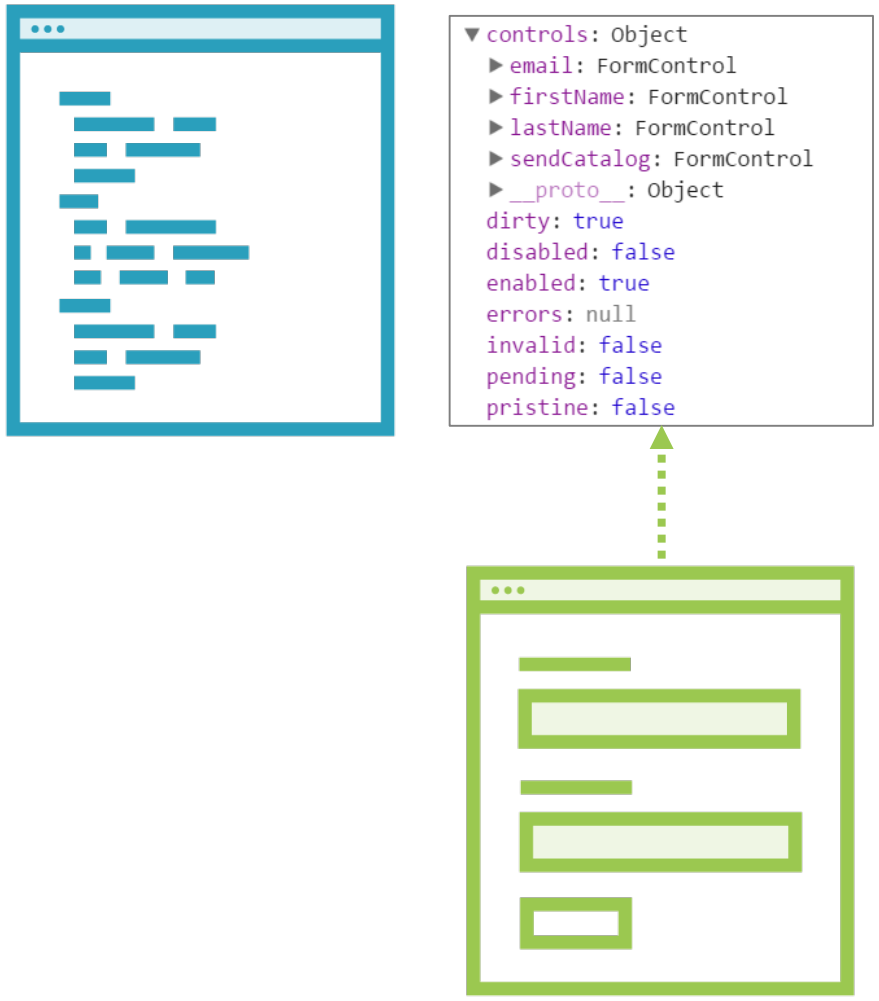
BrowserModule

Reactive - FormsModule



Angular Module

Binding to the Form Model



Reactive Forms Directives

Reactive Forms

- formGroup
- formControl
- formControlName
- formGroupName
- formArrayName

formGroup

customer.component.html

```
<form (ngSubmit)="save()" [formGroup]="customerForm">
    ...
</form>
```

formControlName

customer.component.html

```
<form (ngSubmit)="save()" [formGroup]="customerForm">
  <div>
    <label for="firstNameId">First Name</label>
    <input id="firstNameId" type="text"
           placeholder="First Name (required)"
           formControlName="firstName" />
    <span ... >
       ...
    </span>
  </div>
    ...
</form>
```

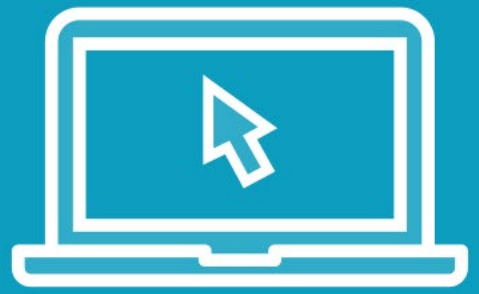
Accessing the Form Model Properties

```
customerForm.controls.firstName.valid

customerForm.get('firstName').valid
```

```
firstName = new FormControl();

ngOnInit(): void {
  this.customerForm = new FormGroup({
      firstName: this.firstName,
      ...
  });
}
```

```
firstName.valid
```

Demo



Binding the Template to the Form Model

Using `setValue` and `patchValue`

```
this.customerForm.setValue({
    firstName: 'Jack',
    lastName: 'Harkness',
    email: 'jack@torchwood.com'
});
```

```
this.customerForm.patchValue({
    firstName: 'Jack',
    lastName: 'Harkness'
});
```

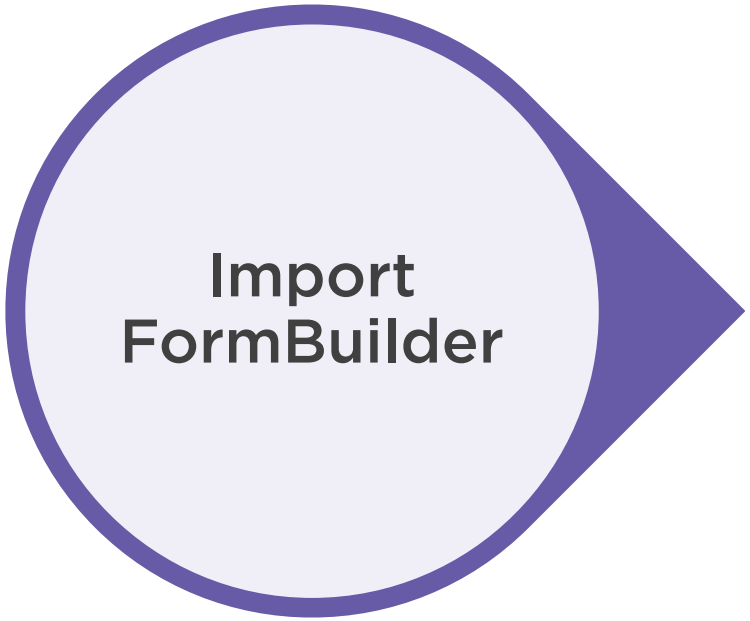
FormBuilder

Creates a form model from a configuration

Shortens boilerplate code

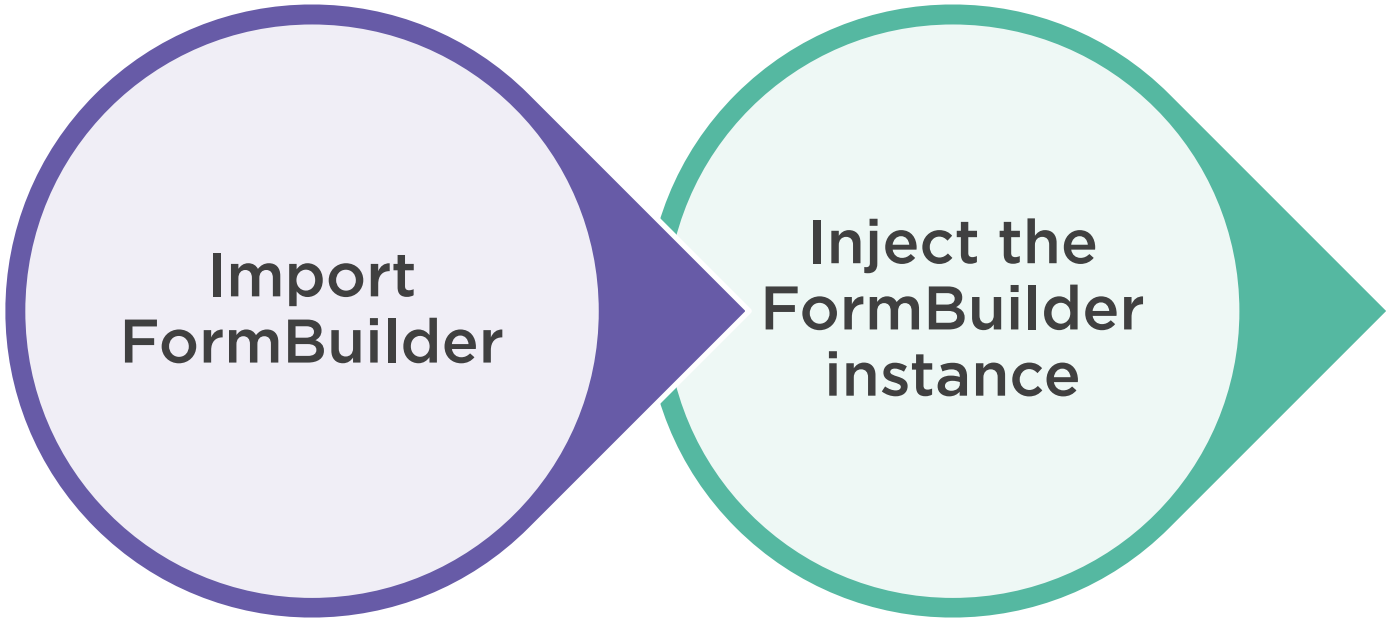
Provided as a service

FormBuilder Steps



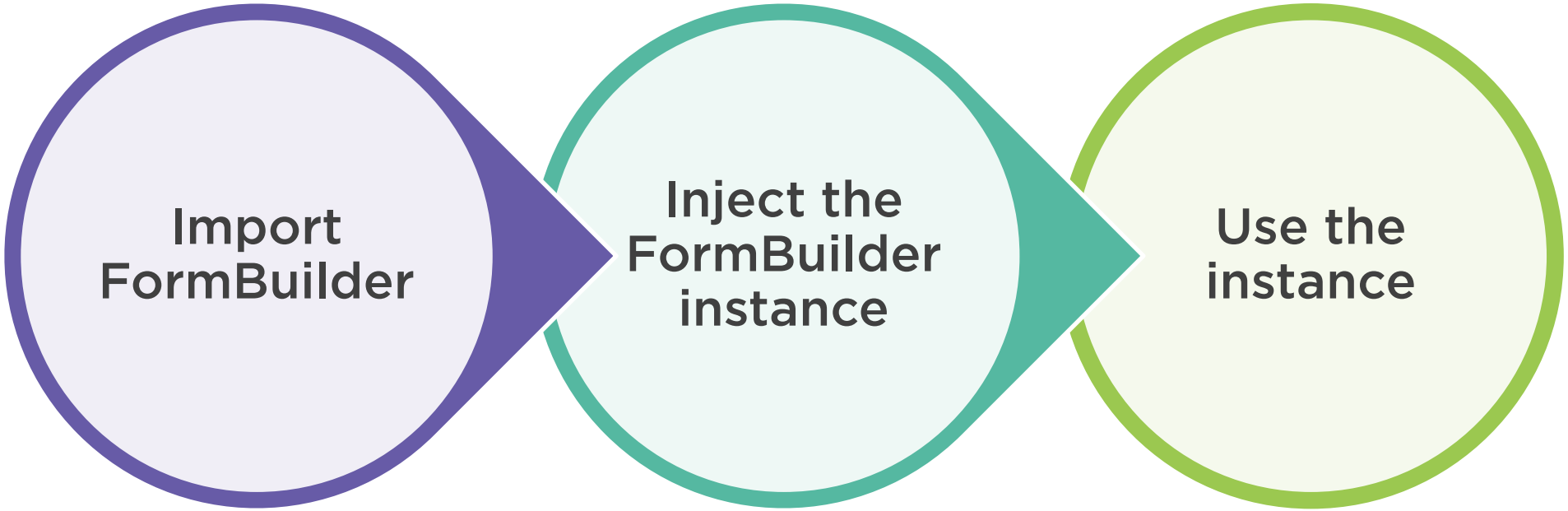

```
import { FormBuilder } from '@angular/forms';
```

FormBuilder Steps




```
constructor(private fb: FormBuilder) { }
```

FormBuilder Steps




```
this.customerForm = this.fb.group({
        firstName: null,
        lastName: null,
        email: null,
        sendCatalog: true
});
```

FormBuilder's FormControl Syntax

```
this.customerForm = this.fb.group({
            firstName: '',
            sendCatalog: true
});
```

```
this.customerForm = this.fb.group({
            firstName: {value: 'n/a', disabled: true},
            sendCatalog: {value: true, disabled: false}
  });
```

```
this.customerForm = this.fb.group({
            firstName: [''],
            sendCatalog: [{value: true, disabled: false}]
  });
```

Checklist: Component Class

- Create a property for the root FormGroup
- Create the FormGroup instance
- Pass in each FormControl instance

```
ngOnInit(): void {
  this.customerForm = new FormGroup({
        firstName: new FormControl(),
        lastName: new FormControl(),
        email: new FormControl(),
        sendCatalog: new FormControl(true)
  });
}
```

Checklist: FormBuilder

- Import FormBuilder
- Inject the FormBuilder instance
- Use that instance

```
ngOnInit(): void {
  this.customerForm = this.fb.group({
              firstName: '',
              lastName: '',
              email: '',
              sendCatalog: true
  });
}
```

Checklist: Angular Module

Import ReactiveFormsModule

Add ReactiveFormsModule to the imports array

```
@NgModule({
  imports: [
    BrowserModule,
    ReactiveFormsModule
  ],
  declarations: [
    AppComponent,
    CustomerComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Checklist: Template

Bind the form element to the FormGroup property

```
<form (ngSubmit)="save()"
      [formGroup]="customerForm">
```

Bind each input element to its associated FormControl

```
<input id="firstNameId"
       type="text"
       placeholder="First Name (required)"
       formControlName="firstName" />
```

Summary

The Component Class

The Angular Module

The Template

Using `setValue` and `patchValue`

Simplifying with FormBuilder

Validation

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/


RIGHT
WRONG
IT DEPENDS

Module Overview

Setting Built-in Validation Rules

Adjusting Validation Rules at Runtime

Custom Validators

Custom Validators with Parameters

Cross-field Validation

Creating the Root FormGroup

```
ngOnInit(): void {
  this.customerForm = this.fb.group({
        firstName: '',
        lastName: '',
        email: '',
        sendCatalog: true
  });
}
```

Creating the FormControls

```
this.customerForm = this.fb.group({
      firstName: '',
      sendCatalog: true
});
```

```
this.customerForm = this.fb.group({
      firstName: {value: 'n/a', disabled: true},
      sendCatalog: {value: true, disabled: false}
});
```

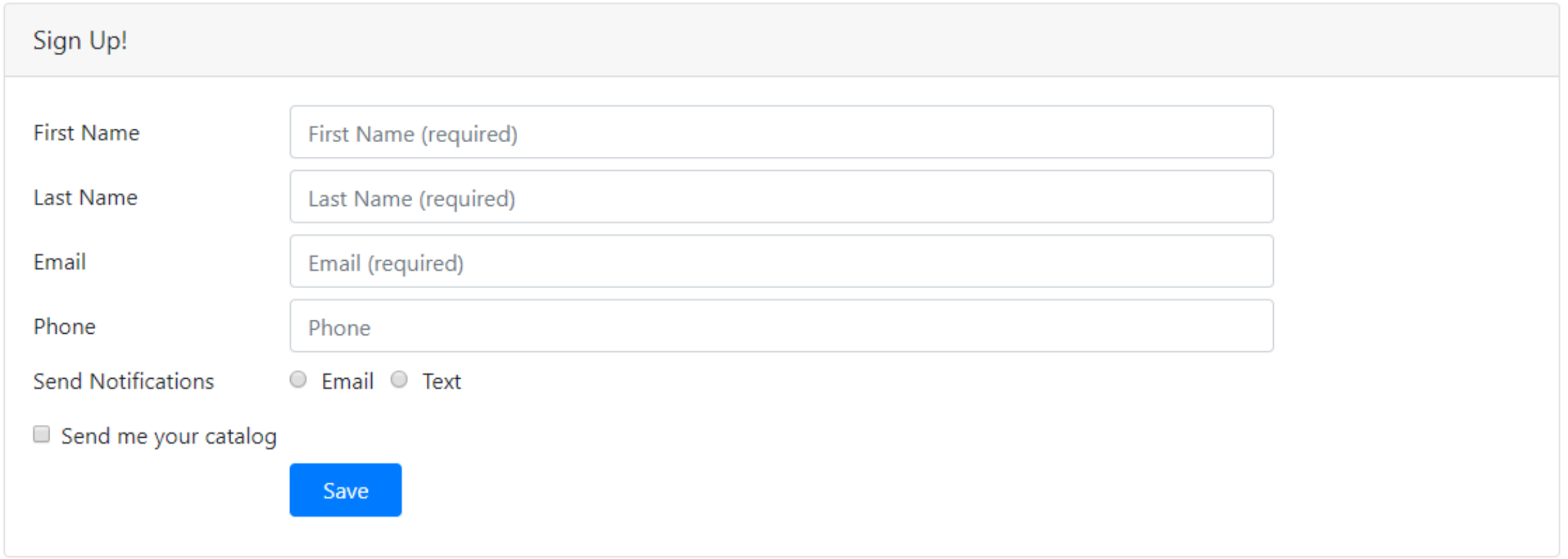
```
this.customerForm = this.fb.group({
      firstName: [''],
      sendCatalog: [true]
});
```

Setting Built-in Validation Rules

```
this.customerForm = this.fb.group({
      firstName: ['', Validators.required],
      sendCatalog: true
});
```

```
this.customerForm = this.fb.group({
      firstName: ['',
         [Validators.required, Validators.minLength(3)]],
      sendCatalog: true
});
```

Adjusting Validation Rules at Runtime



Adjusting Validation Rules at Runtime

```
myControl.setValidators(Validators.required);
```

```
myControl.setValidators([Validators.required,
                         Validators.maxLength(30)]);
```

```
myControl.clearValidators();
```

```
myControl.updateValueAndValidity();
```

Custom Validator

```
function myValidator(c: AbstractControl): {[key: string]: boolean} | null {

    if (somethingIsWrong) {
        return { 'myvalidator': true };
    }
    return null;

}
```

Custom Validator

Sign Up!

First Name — First Name (required)

Last Name — Last Name (required)

Email — Email (required)

Phone — Phone

Send Notifications — Email / Text

Rating

Send me your catalog

Save | Test Data

Custom Validator

```
function myValidator(c: AbstractControl): {[key: string]: boolean} | null {

    if (somethingIsWrong) {
        return { 'myvalidator': true };
    }
    return null;

}
```

Custom Validator with Parameters

```
function myValidator(param: any): ValidatorFn {

  return (c: AbstractControl): {[key: string]: boolean} | null => {

    if (somethingIsWrong) {
        return { 'myvalidator': true };
    }
    return null;

  };
}
```

Cross-field Validation

Sign Up!

First Name — First Name (required)

Last Name — Last Name (required)

Email — Email (required)

Confirm Email — Confirm Email (required)

Phone — Phone

Send Notifications — Email / Text

Rating

Send me your catalog

Save

Cross-field Validation: Nested FormGroup

```
this.customerForm = this.fb.group({
    firstName: ['', [Validators.required, Validators.minLength(3)]],
    lastName: ['', [Validators.required, Validators.maxLength(50)]],
    availability: this.fb.group({
                        start: ['', Validators.required],
                        end: ['', Validators.required]
  })
});
```

```
<div formGroupName="availability">
  ...
  <input formControlName="start"/>
  ...
  <input formControlName="end"/>
</div>
```

Cross-field Validation

Sign Up!

First Name — First Name (required)

Last Name — Last Name (required)

Email — Email (required)

Confirm Email — Confirm Email (required)

Phone — Phone

Send Notifications — Email / Text

Rating

Send me your catalog

Save

Cross-field Validation: Custom Validator

```
function dateCompare(c: AbstractControl): {[key: string]: boolean} | null {
    let startControl = c.get('start');
    let endControl = c.get('end');
    if (startControl.value !== endControl.value) {
        return { 'match': true };
    }
    return null;
}
```

```
this.customerForm = this.fb.group({
    firstName: ['', [Validators.required, Validators.minLength(3)]],
    lastName: ['', [Validators.required, Validators.maxLength(50)]],
    availability: this.fb.group({
                        start: ['', Validators.required],
                        end: ['', Validators.required]
    }, { validator: dateCompare })
});
```

Checklist: Setting Built-in Validation Rules

Import Validators

Pass in the validator or array of validators

```
ngOnInit(): void {
 this.customerForm = this.fb.group({
   firstName: ['', Validators.required],
   lastName: ['', [Validators.required,
                   Validators.maxLength(30)]],
   email: '',
   sendCatalog: true
 });
}
```

Checklist: Adjusting Validation Rules

Determine when to make the change

Use setValidators or clearValidators

Call updateValueAndValidity

```
setNotification(notifyVia: string): void {
  const p = this.myForm.get('phone');
  if (notifyVia === 'text') {
   p.setValidators(Validators.required);
  } else {
   p.clearValidators();
  }
   p.updateValueAndValidity();
}
```

Checklist: Custom Validators

Build a validator function

```
function myValidator(c: AbstractControl):
            {[key: string]: boolean} | null {
    if (somethingIsWrong) {
        return { 'thisValidator': true };
    }
    return null;
}
```

Use it like any other validator

```
this.customerForm = this.fb.group({
   firstName: ['', myValidator],
   sendCatalog: true
});
```

Checklist: Custom Validators with Parameters

Wrap the validator function in a factory function

```
function myValidator(param: any): ValidatorFn {
    return (c: AbstractControl):
           {[key: string]: boolean} | null => {
    if (c.value === param) {
        return { 'thisvalidator': true };
    }
    return null;
}}
```

Use it like any other validator

```
this.customerForm = this.fb.group({
   firstName: ['', myValidator('test')]
});
```

Checklist: Cross-field Validation: FormGroup

Create a nested FormGroup

Add FormControls to that FormGroup

```
this.customerForm = this.fb.group({
  name: ['', Validators.required],
  availability: this.fb.group({
        start: ['', Validators.required],
        end: ['', Validators.required]
  })
});
```

Update the HTML

```
<div formGroupName="availability">
  <input formControlName="start"/>
  <input formControlName="end"/>
</div>
```

Checklist: Cross-field Validation: Validator

Build a custom validator

```
function dateCompare(c: AbstractControl) {
  if (c.get('start').value !==
      c.get('end').value) {
        return { 'match': true };
  }
  return null;
}
```

Use the validator

```
this.customerForm = this.fb.group({
  name: ['', Validators.required],
  availability: this.fb.group({
        start: [null, Validators.required],
        end: [null, Validators.required]
  }, { validator: dateCompare })
});
```

Summary

- Setting Built-in Validation Rules
- Adjusting Validation Rules at Runtime
- Custom Validators
- Custom Validators with Parameters
- Cross-field Validation

Reacting to Changes

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/

Module Overview

Watching

Reacting

Reactive Transformations

Watching

```
(property) AbstractControl.valueChanges: Observable<any>
```

A multicasting observable that emits an event every time the value of the control changes, in the UI or programmatically.

```
const phoneC

phoneControl.valueChanges.subscribe();
```

```
const phoneControl =
this.customerForm.get('phone');

phoneControl.statusChanges.subscribe();
```

Watching

```
(property) AbstractControl.valueChanges: Observable<any>
```

A multicasting observable that emits an event every time the value of the control changes, in the UI or programmatically.

```
const phoneC

phoneControl.valueChanges.subscribe();
```

```
const phoneControl = this.customerForm.get('phone');

phoneControl.statusChanges.subscribe();
```

Watching

```
this.myFormControl.valueChanges.subscribe(value =>
   console.log(value));
```

```
this.myFormGroup.valueChanges.subscribe(value =>
   console.log(JSON.stringify(value)));
```

```
this.customerForm.valueChanges.subscribe(value =>
   console.log(JSON.stringify(value)));
```

Watching

Sign Up!

First Name — First Name (required)

Last Name — Last Name (required)

Email — Email (required)

Confirm Email — Confirm Email (required)

Phone — Phone

Send Notifications — Email / Text

Rating

Send me your catalog

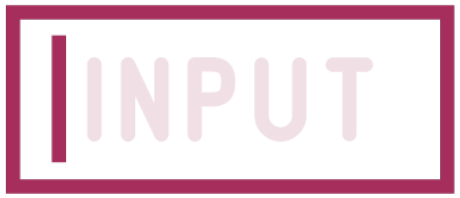
Save

Reacting

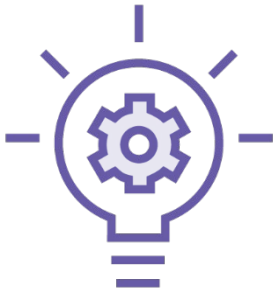
Adjust validation rules

Handle validation messages

Modify user interface elements

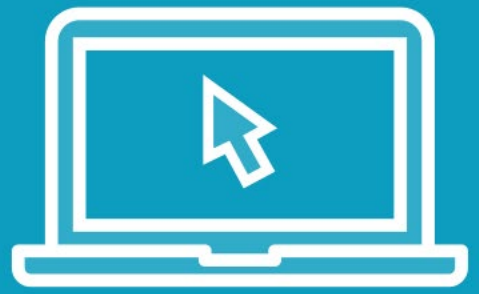


Provide automatic suggestions



...

Demo

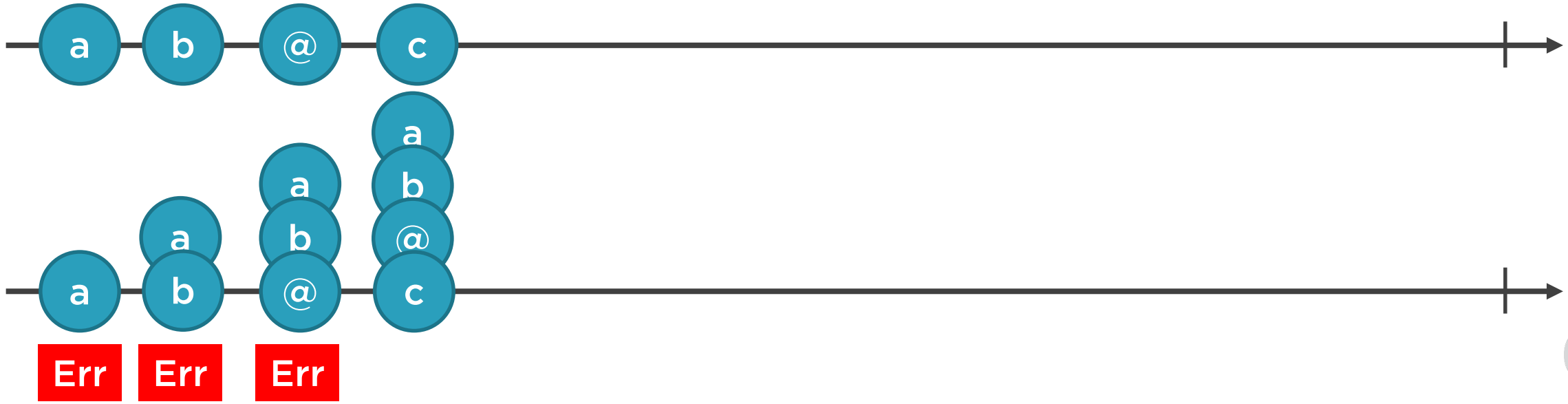


Displaying Validation Messages

Reactive Transformations

debounceTime

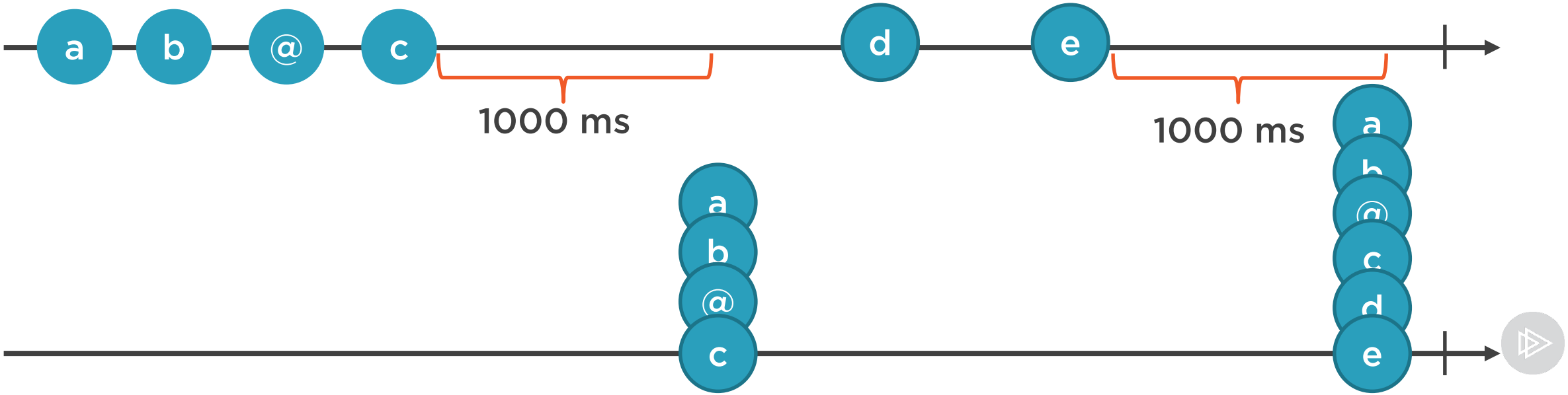
- Ignores events until a specific time has passed without another event
- `debounceTime(1000)` waits for 1000 milliseconds (1 sec) of no events before emitting another event



Reactive Transformations

debounceTime

- Ignores events until a specific time has passed without another event
- `debounceTime(1000)` waits for 1000 milliseconds (1 sec) of no events before emitting another event



Reactive Transformations

throttleTime

- Emits a value, then ignores subsequent values for a specific amount of time

distinctUntilChanged

- Suppresses duplicate consecutive items

Checklist: Watching

Use the valueChanges Observable property

Subscribe to the Observable

```
this.myFormControl.valueChanges.subscribe(
    value => console.log(value)
);
```

Checklist: Reacting

```
this.myFormControl.valueChanges.subscribe(
  value => this.setNotification(value)
);
```

- Change validation rules
- Handle validation messages
- Adjust user-interface elements
- Provide automatic suggestions
- And more

Checklist: Reactive Transformations

import the operator

```
import { debounceTime } from 'rxjs/operators';
```

Use the operator

```
this.myFormControl.valueChanges.pipe(
  debounceTime(1000)
).subscribe(
  value => console.log(value)
);
```

Summary

Watching

Reacting

Reactive Transformations

```
<div class="form-group">
  <label class="col-md-2 col-form-label"
         for="emailId">Email</label>
  <div class="col-md-8">
    <input class="form-control"
         id="emailId" type="email"
         placeholder="Email (required)"
         required
         email
         [(ngModel)]="customer.email"
         name="email"
         #emailVar="ngModel"
        [ngClass]="{'is-invalid':
            (emailVar.touched ||
             emailVar.dirty) &&
             !emailVar.valid }" />
    <span class="invalid-feedback">
      <span *ngIf="emailVar.errors?.required">
         Please enter your email address.
      </span>
      <span *ngIf="emailVar.errors?.email">
         Please enter a valid email address.
      </span>
    </span>
  </div>
</div>
```

Template-driven

```
<div class="form-group">
  <label class="col-md-2 col-form-label"
         for="emailId">Email</label>
  <div class="col-md-8">
    <input class="form-control"
           id="emailId" type="email"
           placeholder="Email (required)"
           formControlName = "email"
          [ngClass]="{'is-invalid':
                              emailMessage}" />
    <span class="invalid-feedback">
       {{ emailMessage }}
    </span>
   </div>
</div>
```

Reactive

Dynamically Duplicate Input Elements

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/

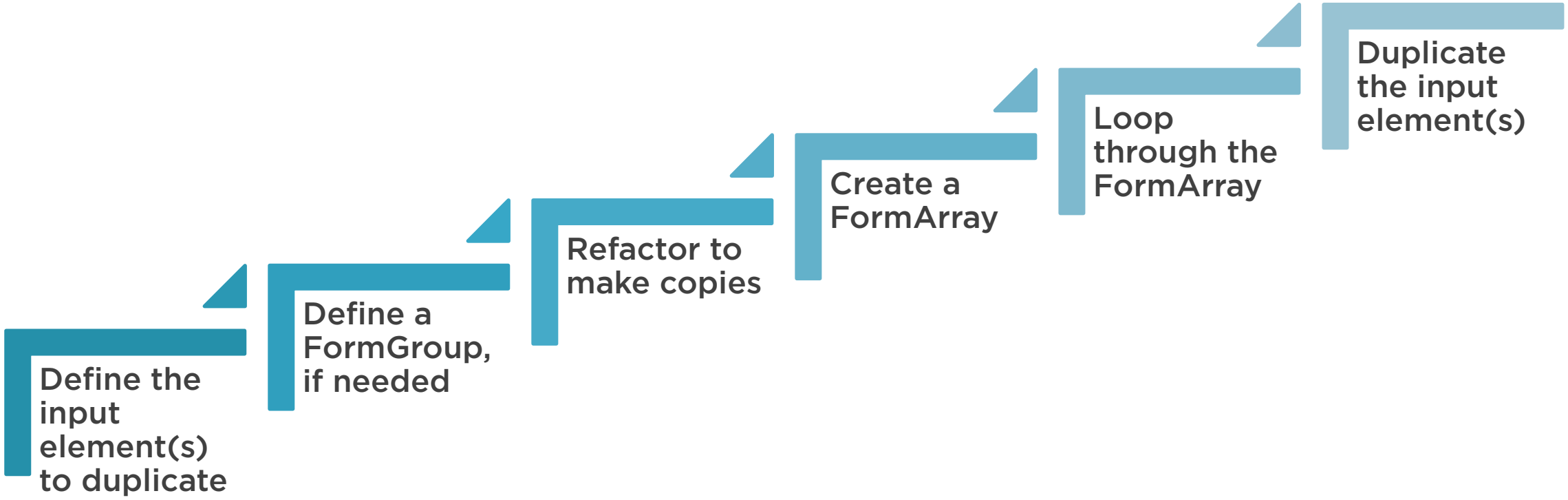
Module Overview

Steps

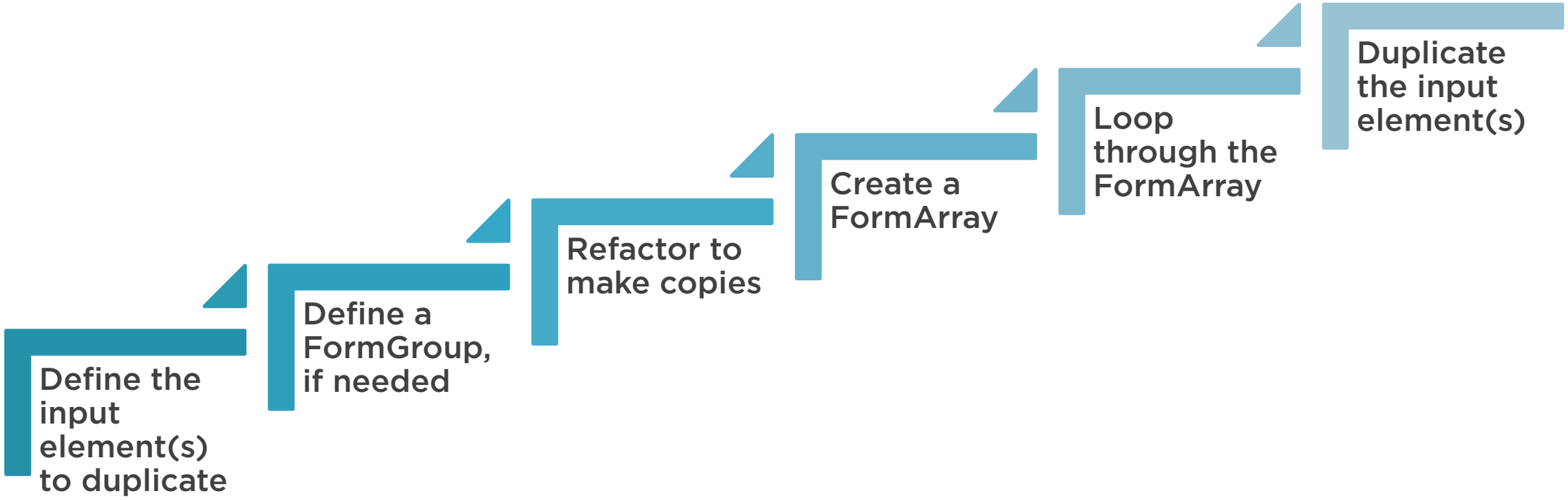
Perform Each Step

- FormArrays

Steps to Dynamically Duplicate Input Elements



Steps to Dynamically Duplicate Input Elements



Sign Up!

First Name — First Name (required)

Last Name — Last Name (required)

Email — Email (required)

☑ Send me your catalog

Address Type — ◉ Home ○ Work ○ Other

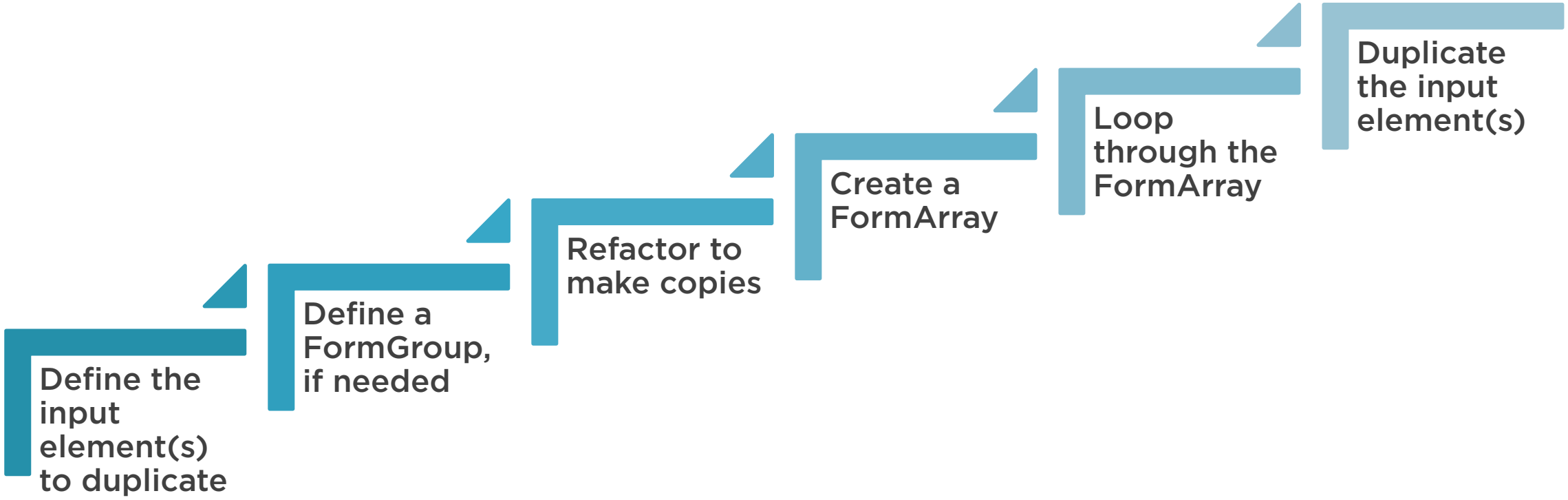
Street Address 1 — Street address

Street Address 2 — Street address (second line)

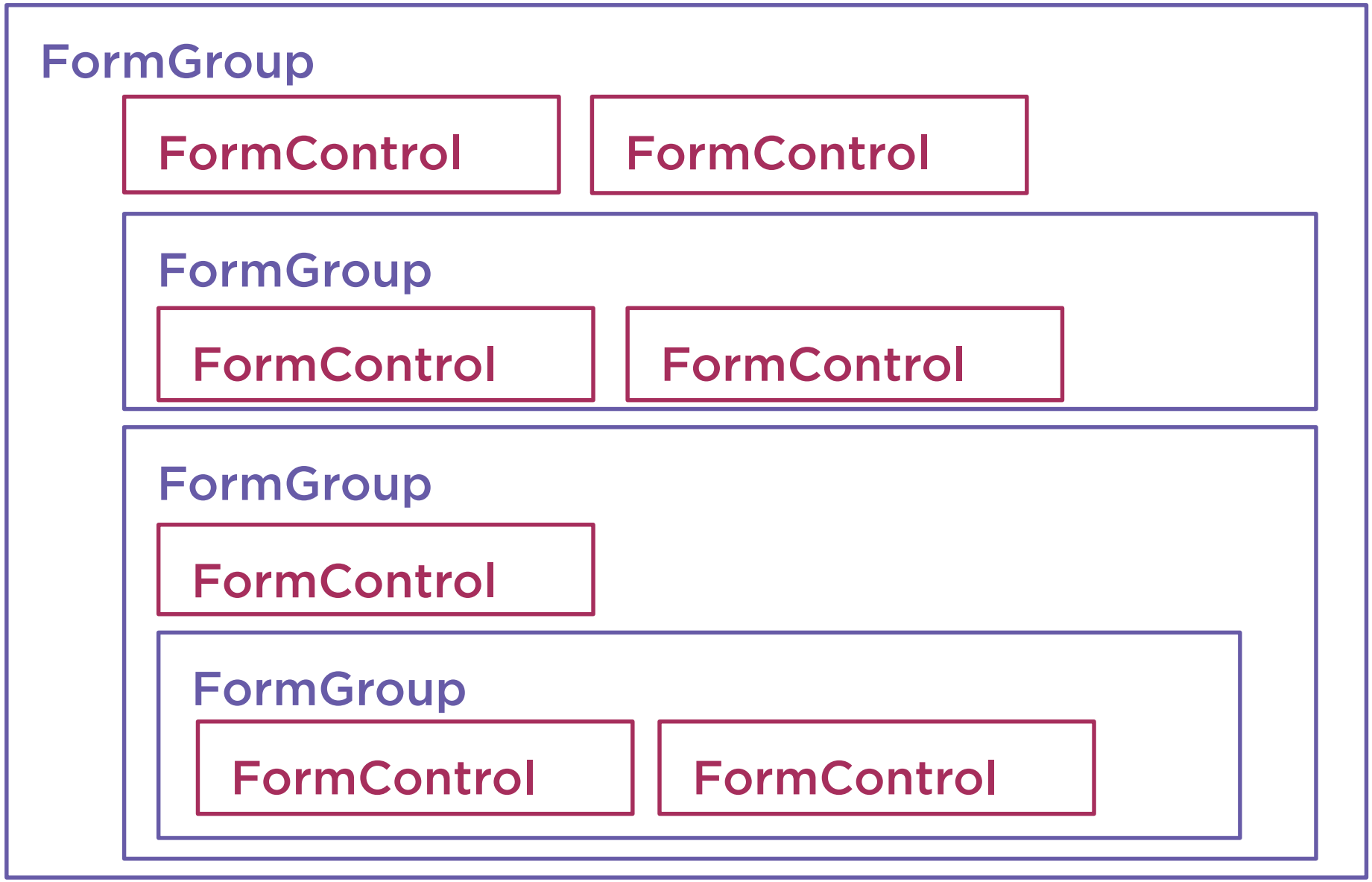
City, State, Zip Code — City | Select a State... | Zip Code

Save

Steps to Dynamically Duplicate Input Elements



FormGroup


FormGroup
FormControl
FormControl
FormGroup
FormControl
FormControl
FormGroup
FormControl
FormGroup
FormControl
FormControl

Benefits of a FormGroup

Match the value of the form model to the data model

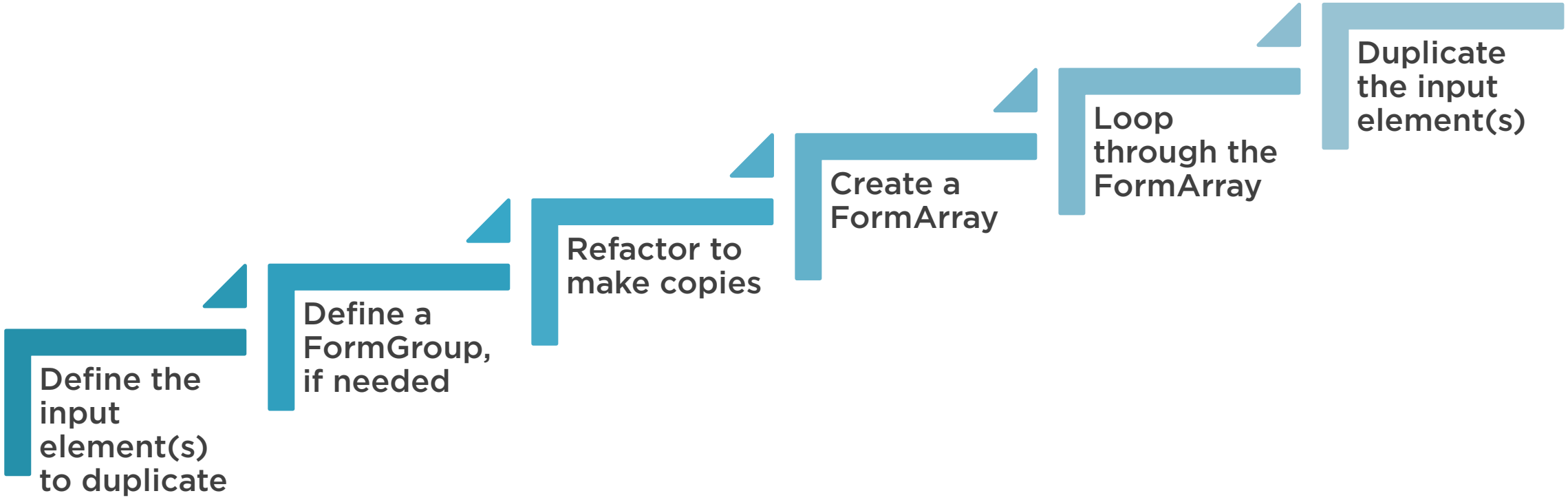
Check touched, dirty, and valid state

Watch for changes and react

Perform cross field validation

Dynamically duplicate the group

Steps to Dynamically Duplicate Input Elements

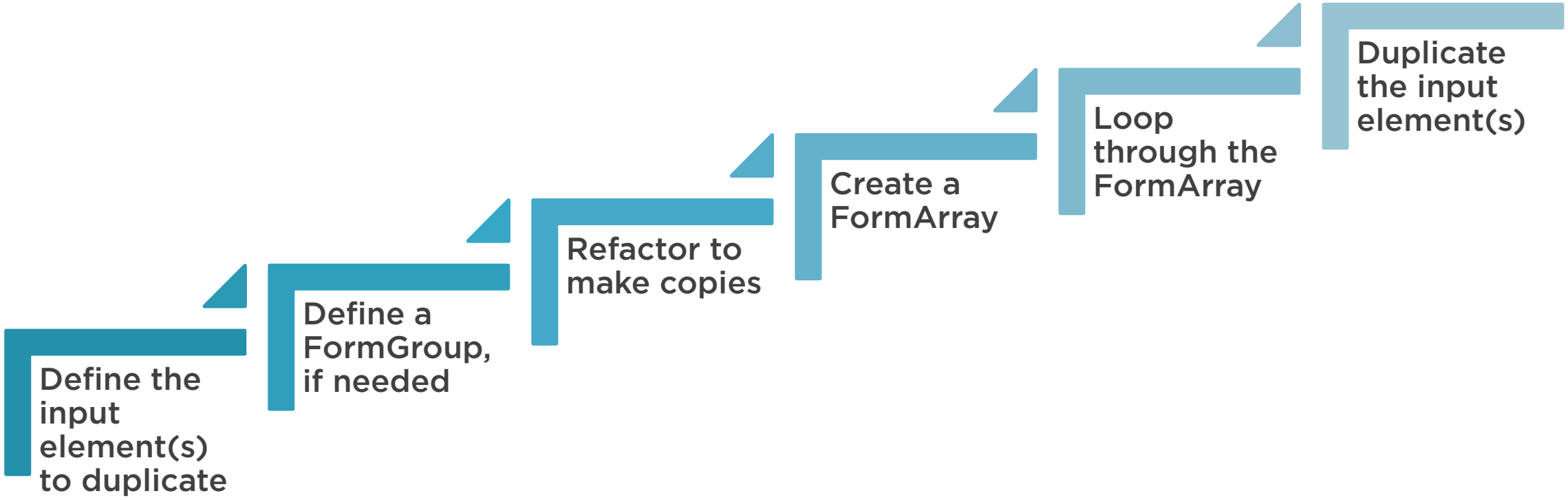


Creating a FormGroup in a Method

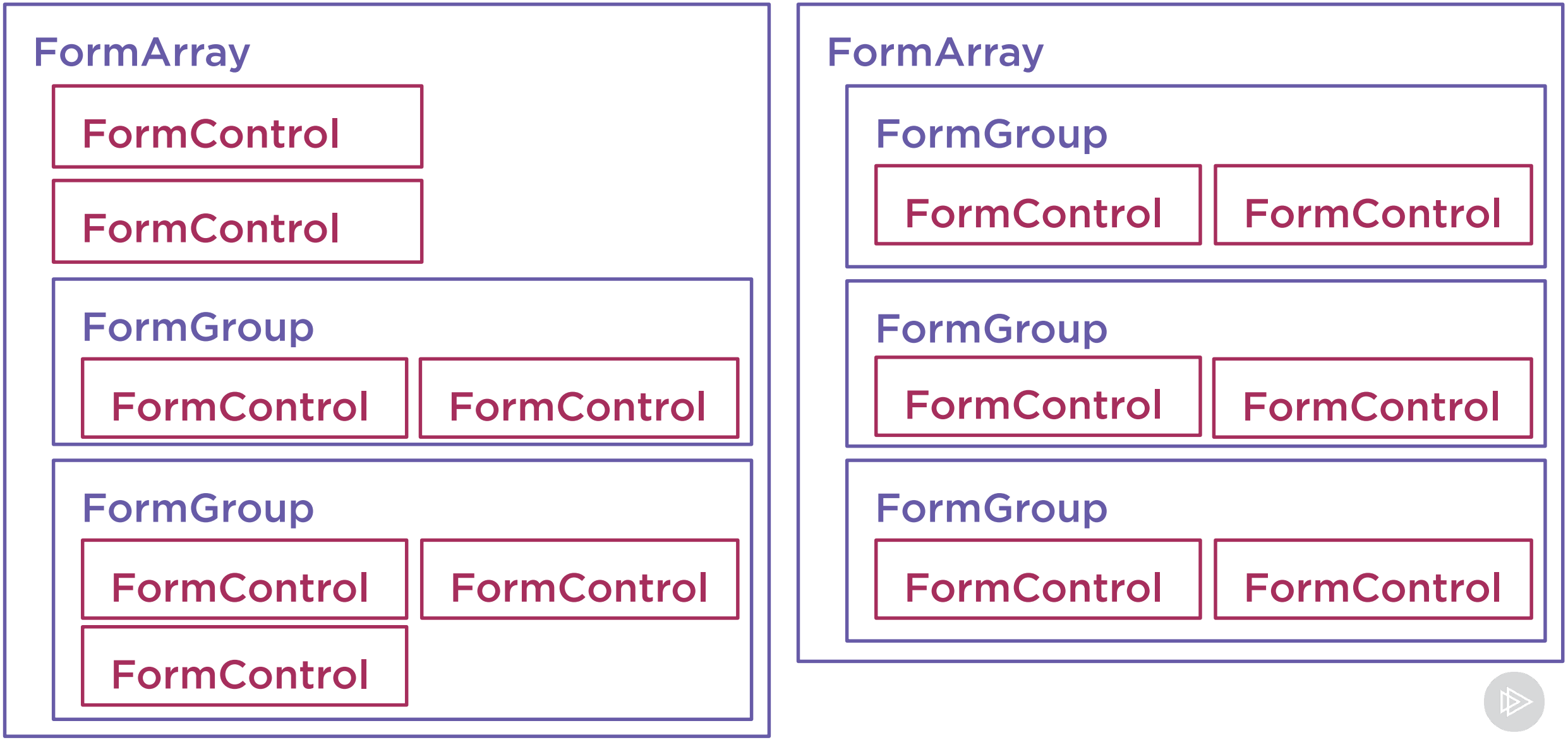
```
buildAddress(): FormGroup {
  return this.fb.group({
      addressType: 'home',
      street1: '',
      street2: '',
      city: '',
      state: '',
      zip: ''
  });
}
```

```
this.customerForm = this.fb.group({
        ...
        addresses: this.buildAddress()
});
```

Steps to Dynamically Duplicate Input Elements



FormArray

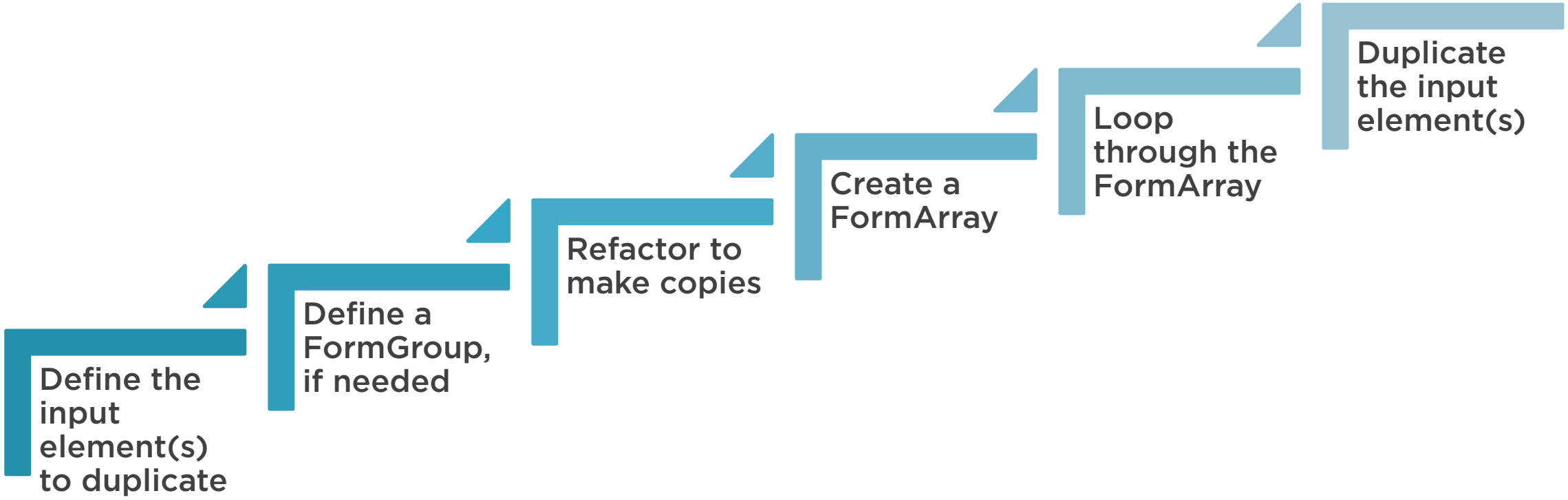

FormArray
FormControl
FormControl
FormGroup
FormControl
FormControl
FormGroup
FormControl
FormControl
FormControl
FormArray
FormGroup
FormControl
FormControl
FormGroup
FormControl
FormControl
FormGroup
FormControl
FormControl

Creating a FormArray

```
this.myArray = new FormArray([...]);
```

```
this.myArray = this.fb.array([...]);
```

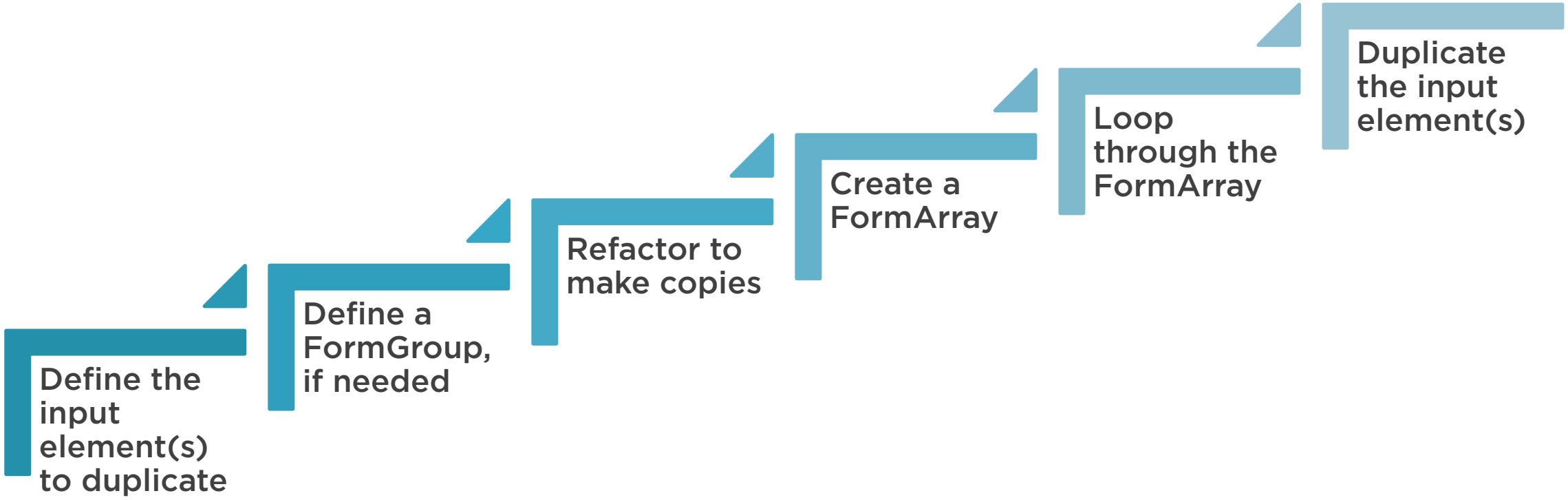
Steps to Dynamically Duplicate Input Elements



Looping Through a FormArray

```
<div formArrayName="addresses"
     *ngFor="let address of addresses.controls; let i=index">

  <div [formGroupName]="i">
    ...
  </div>

</div>
```

Steps to Dynamically Duplicate Input Elements

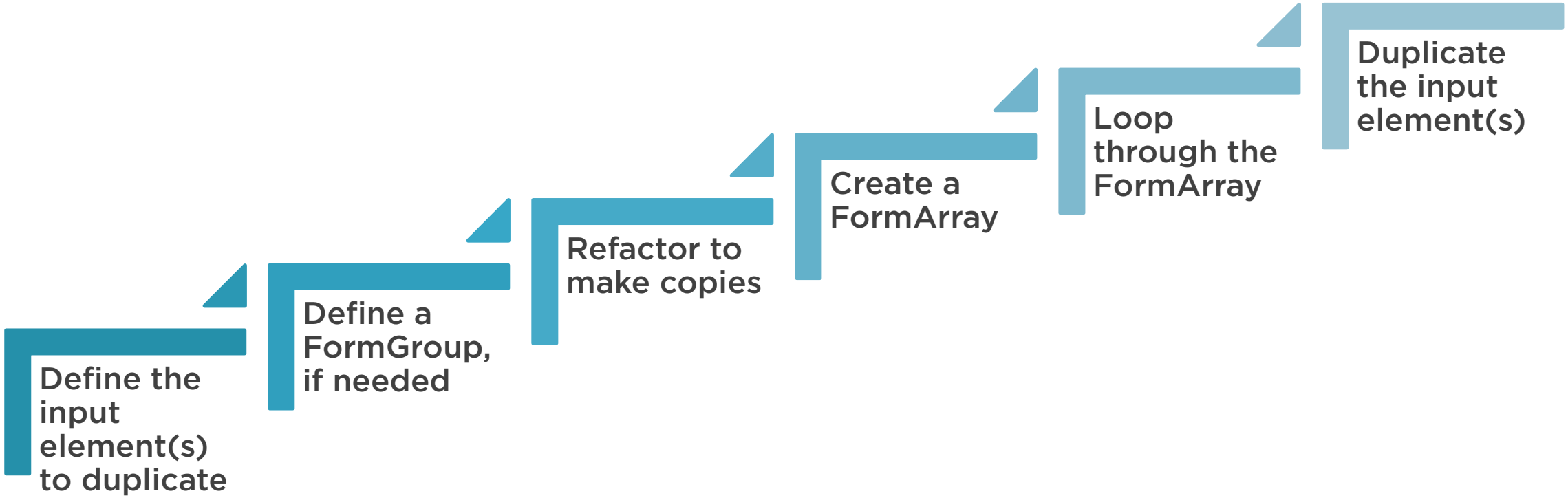


Duplicate the Input Elements

```
addAddress(): void {
 this.addresses.push(this.buildAddress());
}
```

```
<button class="btn btn-primary"
        type="button"
        (click)="addAddress()">
    Add Another Address
</button>
```

Checklist: Dynamically Duplicate Input Elements



Summary

Steps

Perform Each Step

- FormArrays

Reactive Form in Context

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/

Module Overview

Sample Application

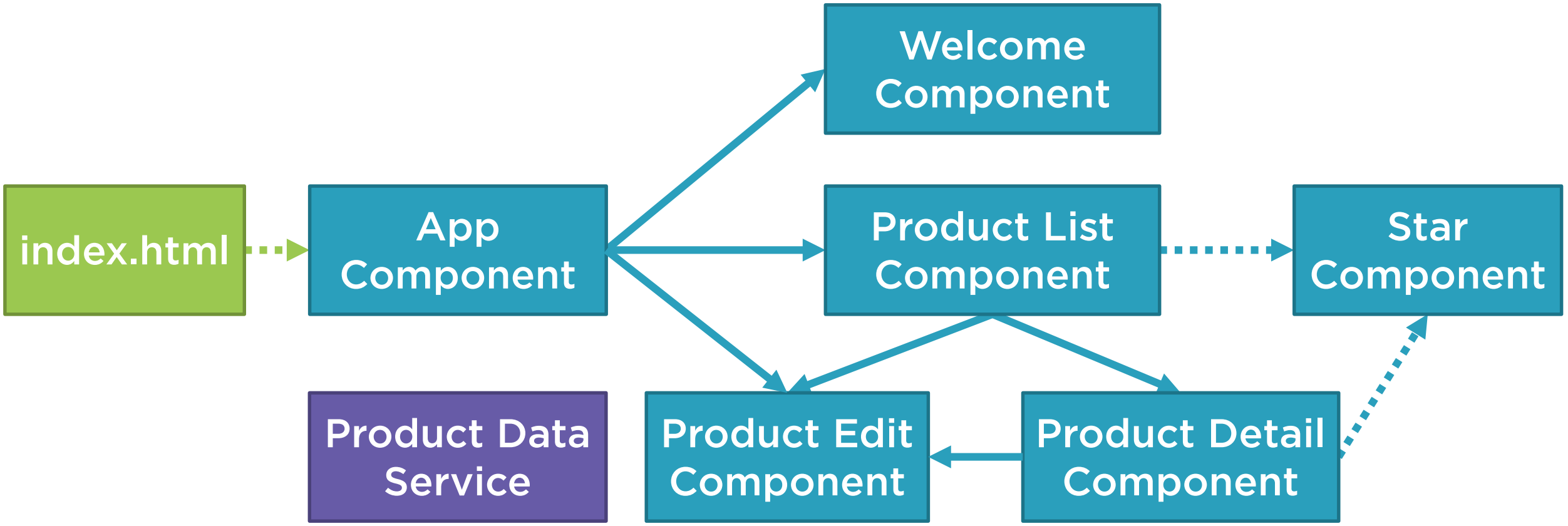
Routing to the Form

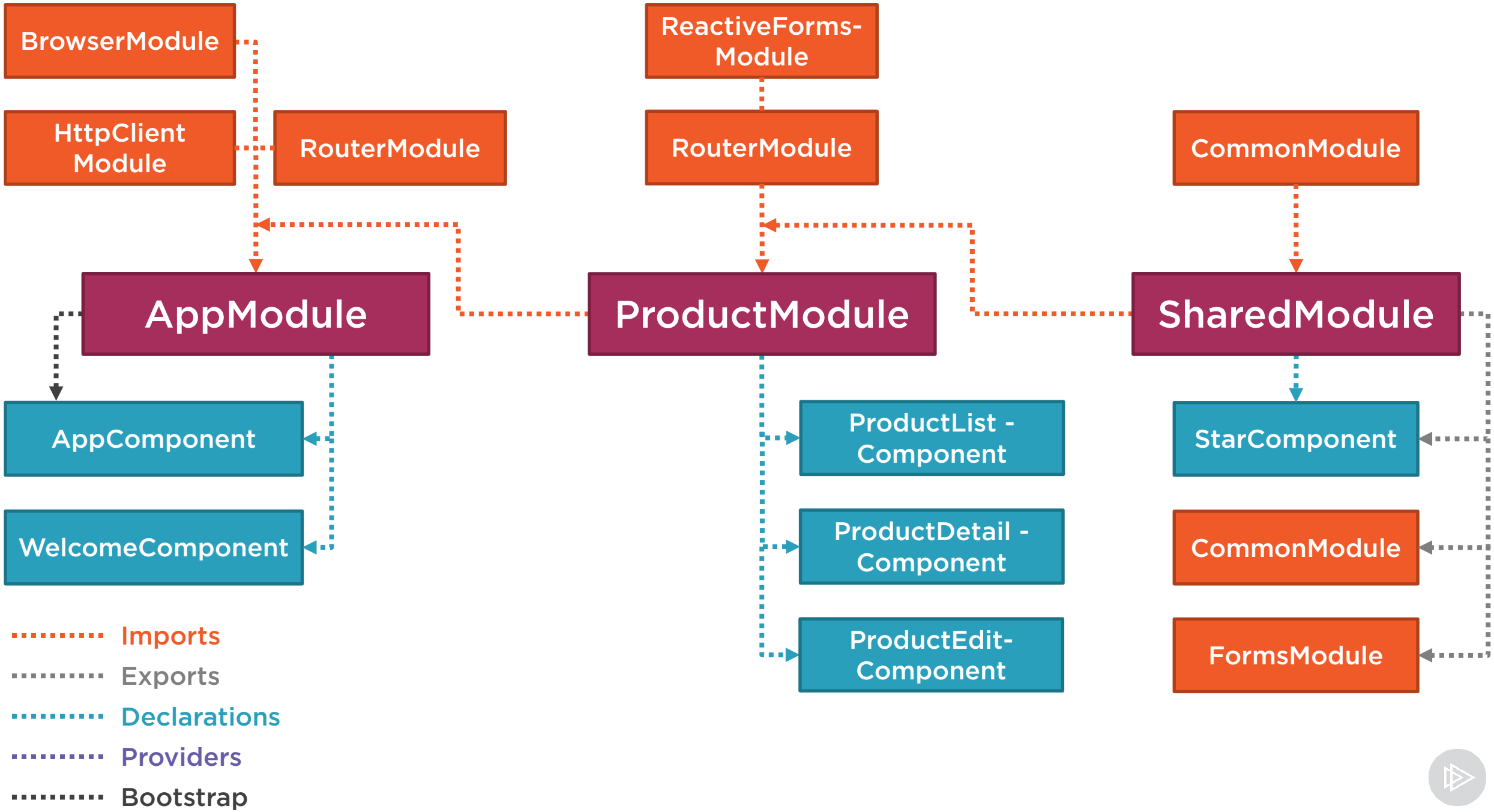
Reading a Route Parameter

Setting a `canDeactivate` Guard

Refactoring to a Custom Validation Class

APM Sample Application Architecture




BrowserModule
HttpClient Module
RouterModule
ReactiveForms-Module
RouterModule
CommonModule
AppModule
ProductModule
SharedModule
AppComponent
WelcomeComponent
ProductList - Component
ProductDetail - Component
ProductEdit- Component
StarComponent
CommonModule
FormsModule
Imports
Exports
Declarations
Providers
Bootstrap

Acme Product Management Home Product List Add Product

Product List

Filter by:

Show Image	Product	Code	Available	Price	5 Star Rating	
	Leaf Rake	GDN-0011	March 19, 2018	$19.95	★★★	Edit
	Garden Cart	GDN-0023	March 18, 2018	$32.99	★★★★	Edit
	Hammer	TBX-0048	May 21, 2018	$8.90	★★★★★	Edit
	Saw	TBX-0022	May 15, 2018	$11.55	★★★★	Edit
	Video Game Controller	GMG-0042	October 15, 2018	$35.95	★★★★	Edit

Acme Product Management Home Product List Add Product

Product Detail: Hammer

Name: Hammer
Code: TBX-0048
Description: Curved claw steel hammer
Availability: May 21, 2018
Price: $8.90
5 Star Rating: ★★★★★
Tags: tools,hamm ,construction

Back Edit

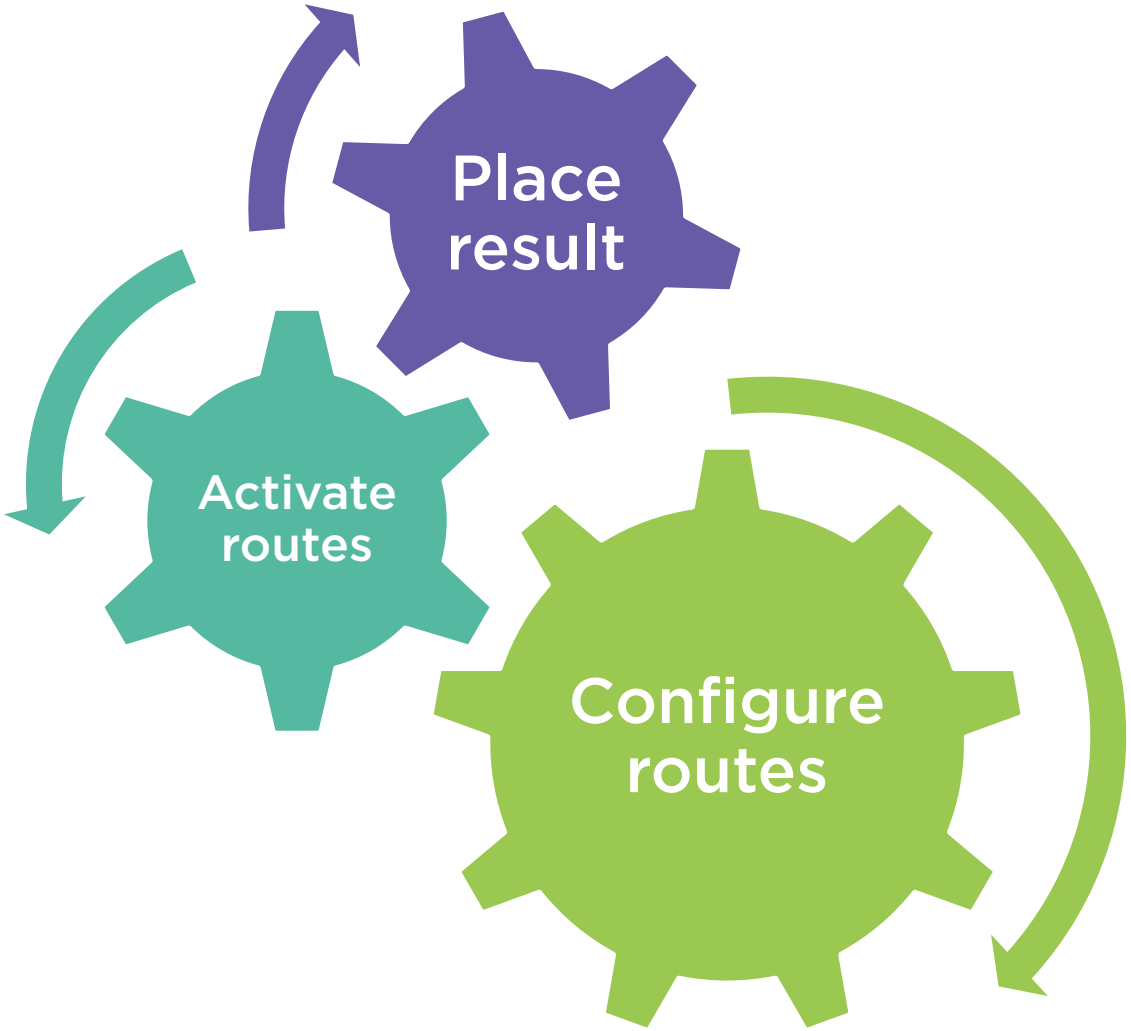
Acme Product Management Home Product List Add Product

Edit Product: Hammer

Product Name: Hammer
Product Code: TBX-0048
Star Rating (1-5): 4.8
Tag: tools Delete Tag
Tag: hammer Delete Tag
Tag: construction Delete Tag
Add Tag
Description: Curved claw steel hammer

Save Cancel Delete

Routing Steps



Configuring Routes

```
[
 { path: 'products', component: ProductListComponent },
 { path: 'products/:id', component: ProductDetailComponent },
 { path: 'products/:id/edit', component: ProductEditComponent }
]
```

Route Guards

```
[
 { path: 'products', component: ProductListComponent },
 { path: 'products/:id',
   canActivate: [ ProductDetailGuard ],
   component: ProductDetailComponent },
 { path: 'products/:id/edit',
   canDeactivate: [ ProductEditGuard ],
   component: ProductEditComponent }
]
```

Acme Product Management Home Product List Add Product

Product List

Filter by:

Show Image

Product	Code	Available	Price	5 Star Rating	
Leaf Rake	GDN-0011	March 19, 2018	$19.95	★★★☆	Edit
Garden Cart	GDN-0023	March 18, 2018	$32.99	★★★★☆	Edit
Hammer	TBX-0048	May 21, 2018	$8.90	★★★★★	Edit
Saw	TBX-0022	May 15, 2018	$11.55	★★★★☆	Edit
Video Game Controller	GMG-0042	October 15, 2018	$35.95	★★★★½	Edit

Acme Product Management Home Product List Add Product

Product Detail: Hammer

Name: Hammer
Code: TBX-0048
Description: Curved claw steel hammer
Availability: May 21, 2018
Price: $8.90
5 Star Rating: ★★★★★
Tags: tools,hammer,construction

Back Edit

Acme Product Management Home Product List Add Product

Edit Product: Hammer

Product Name: Hammer
Product Code: TBX-0048
Star Rating (1-5): 4.8
Tag: tools Delete Tag
Tag: hammer Delete Tag
Tag: construction Delete Tag

Add Tag

Description: Curved claw steel hammer

Save Cancel Delete

Tying Routes to Actions

app.component.ts

```
...

@Component({
    selector: 'pm-root',
    template: `
     <ul class='navbar-nav'>
       <li><a [routerLink]="['/welcome']">Home</a></li>
       <li><a [routerLink]="['/products']">Product List</a></li>
       <li><a [routerLink]="['/products', '0', 'edit']">
           Add Product</a>
       </li>
     </ul>
    `
})
```

Placing the Views

app.component.ts

```
...

@Component({
    selector: 'pm-root',
    template: `
     <ul class='navbar-nav'>
       <li><a [routerLink]="['/welcome']">Home</a></li>
       <li><a [routerLink]="['/products']">Product List</a></li>
       <li><a [routerLink]="['/products', '0', 'edit']">
           Add Product</a>
       </li>
     </ul>
     <router-outlet></router-outlet>
    `
})
```

Reading Parameters from a Route

```
{ path: 'products/:id/edit', component: ProductEditComponent }
```

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

```
constructor(private route: ActivatedRoute) {
   let id = +this.route.snapshot.paramMap.get('id');
   ...
}
```

```
constructor(private route: ActivatedRoute) {
   this.sub = this.route.paramMap.subscribe(
      params => {
         const id = +params.get('id');
         ...
      }
   );
}
```

Building a Guard

product-edit.guard.ts

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

@Injectable({
  providedIn: 'root'
})
export class ProductEditGuard
           implements CanDeactivate<ProductEditComponent> {

   canDeactivate(component: ProductEditComponent): boolean {
      ...
   }
}
```

Custom Validator with Parameters

```
function range(min:number, max:number): ValidatorFn {

    return (c: AbstractControl): {[key: string]: boolean} | null => {

    if (c.value < min || c.value > max) {
        return { 'range': true };
    }
    return null;

}
```

Checklist: Reactive Form in Context

- Add a route configuration
- Add user interface element(s) to activate the route
- Read the route parameter
- Set up a `canDeactivate` guard
- Refactor validators to a custom validation class for reuse

Summary

Sample Application

Routing to the Form

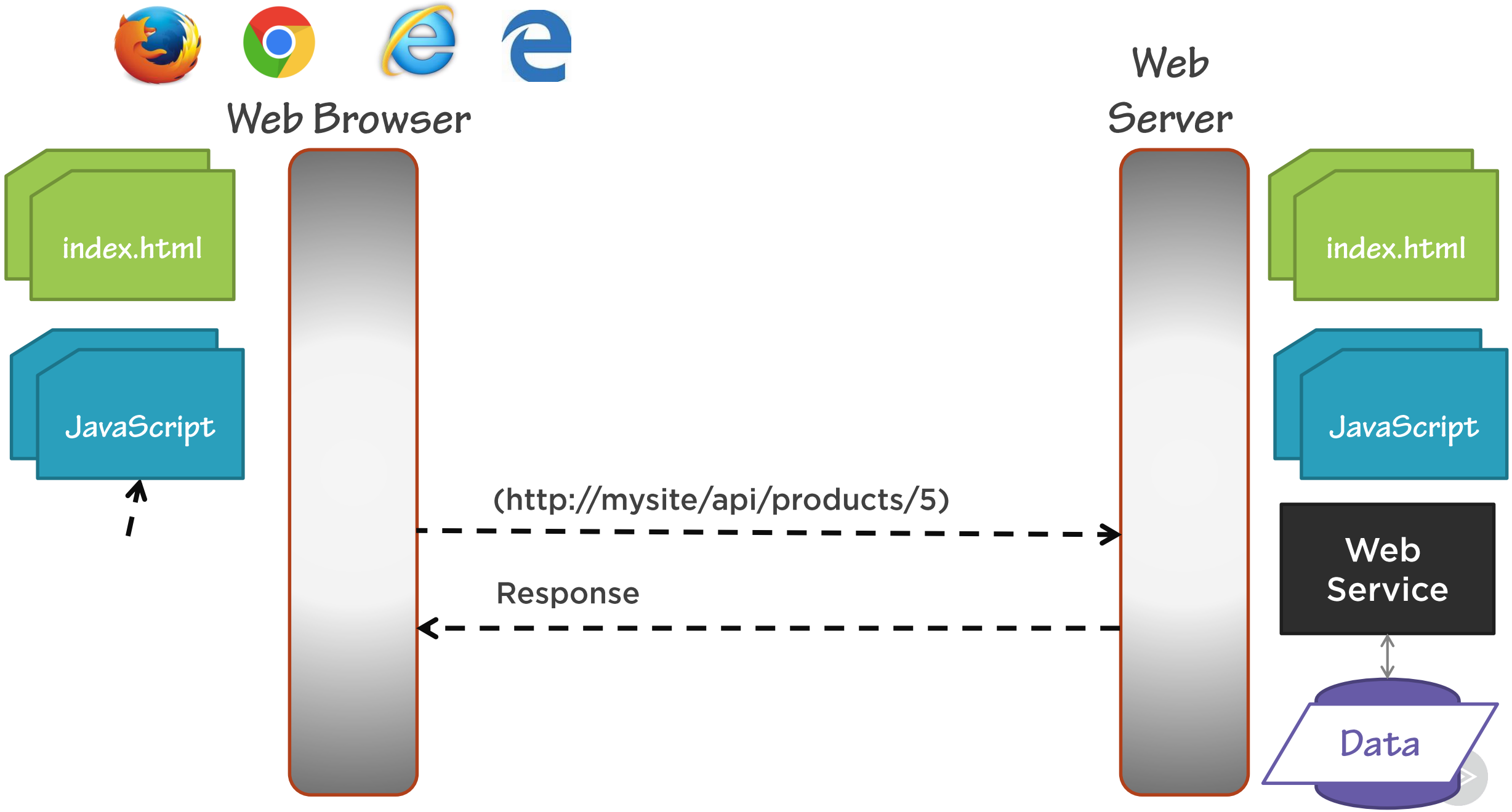
Reading a Route Parameter

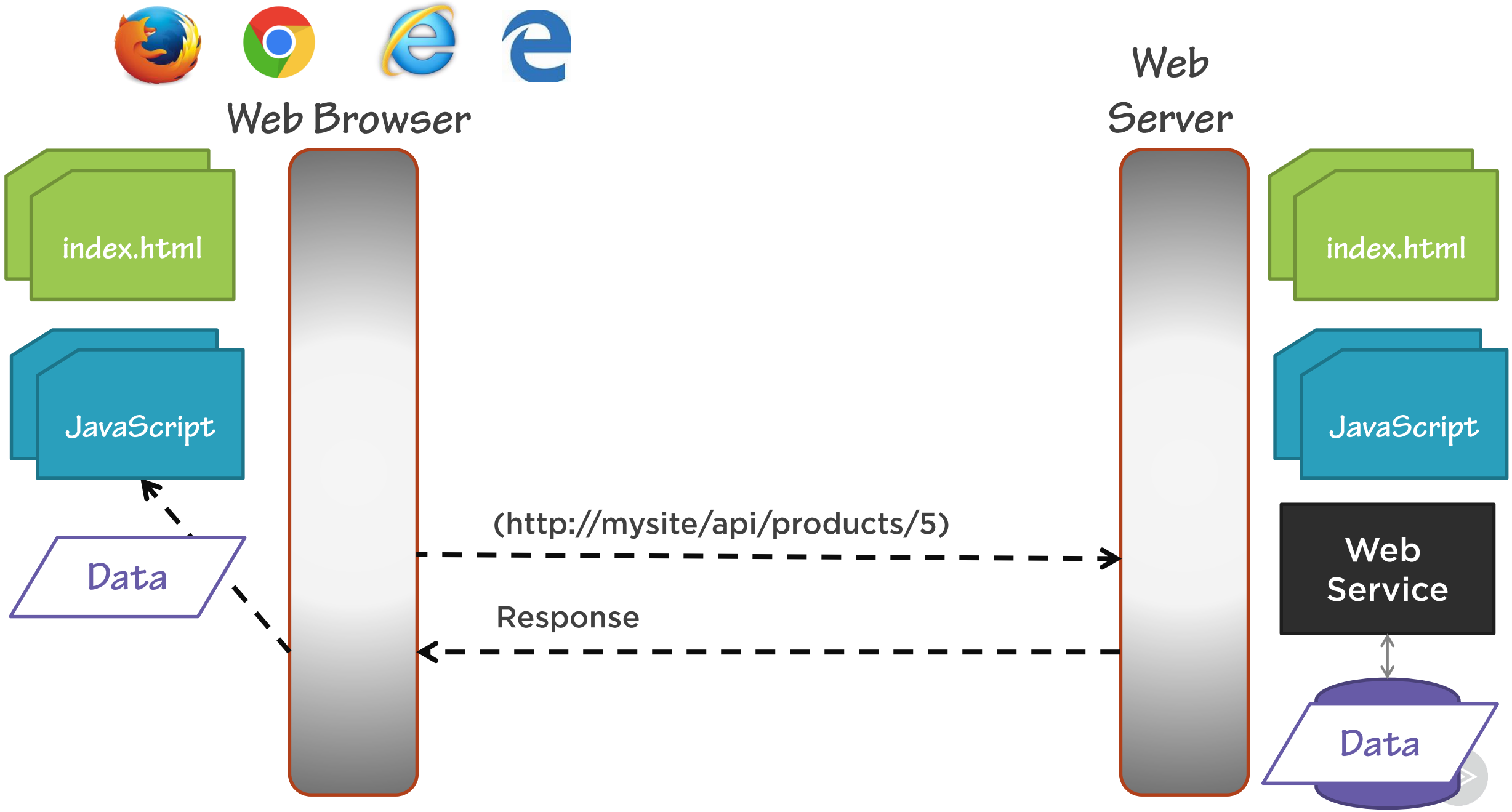
Setting a `canDeactivate` Guard

Refactoring to a Custom Validation Class

Create, Read, Update, Delete (CRUD)

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE
@deborahkurata | blogs.msmvps.com/deborahk/


Web Browser
Web Server
index.html
JavaScript
index.html
JavaScript
(http://mysite/api/products/5)
Response
Web Service
Data


Web Browser
Web Server
index.html
JavaScript
Data
(http://mysite/api/products/5)
Response
index.html
JavaScript
Web Service
Data

Module Overview

Data Access Service

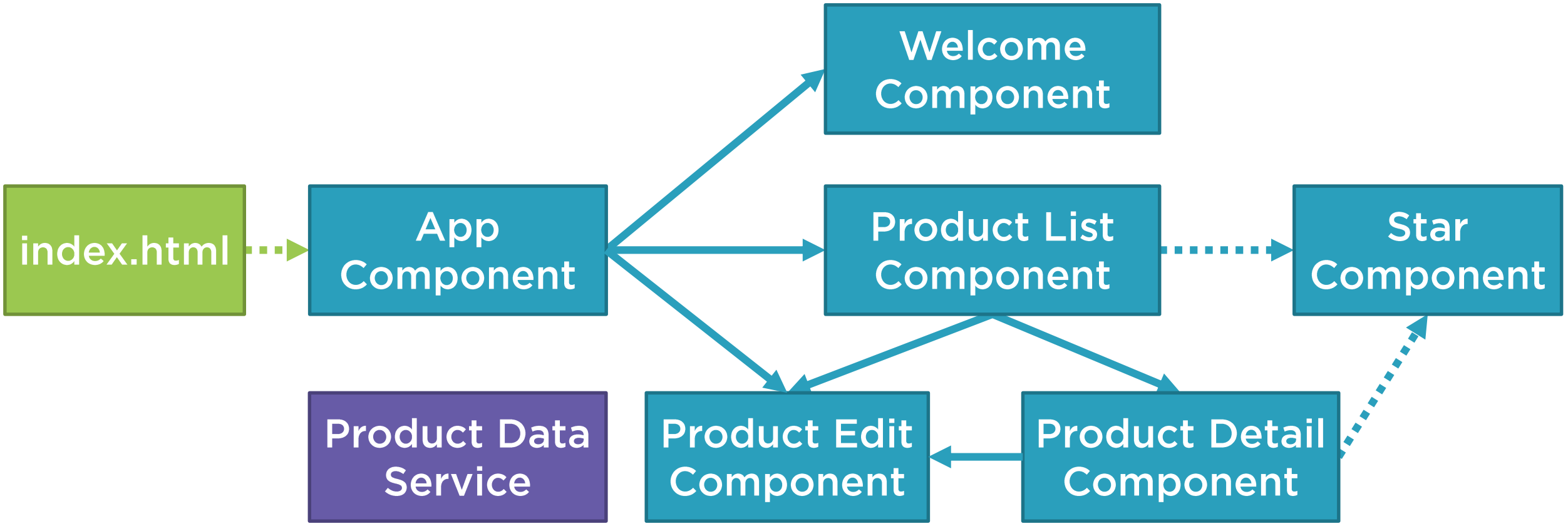
Creating Data

Reading Data

Updating Data

Deleting Data

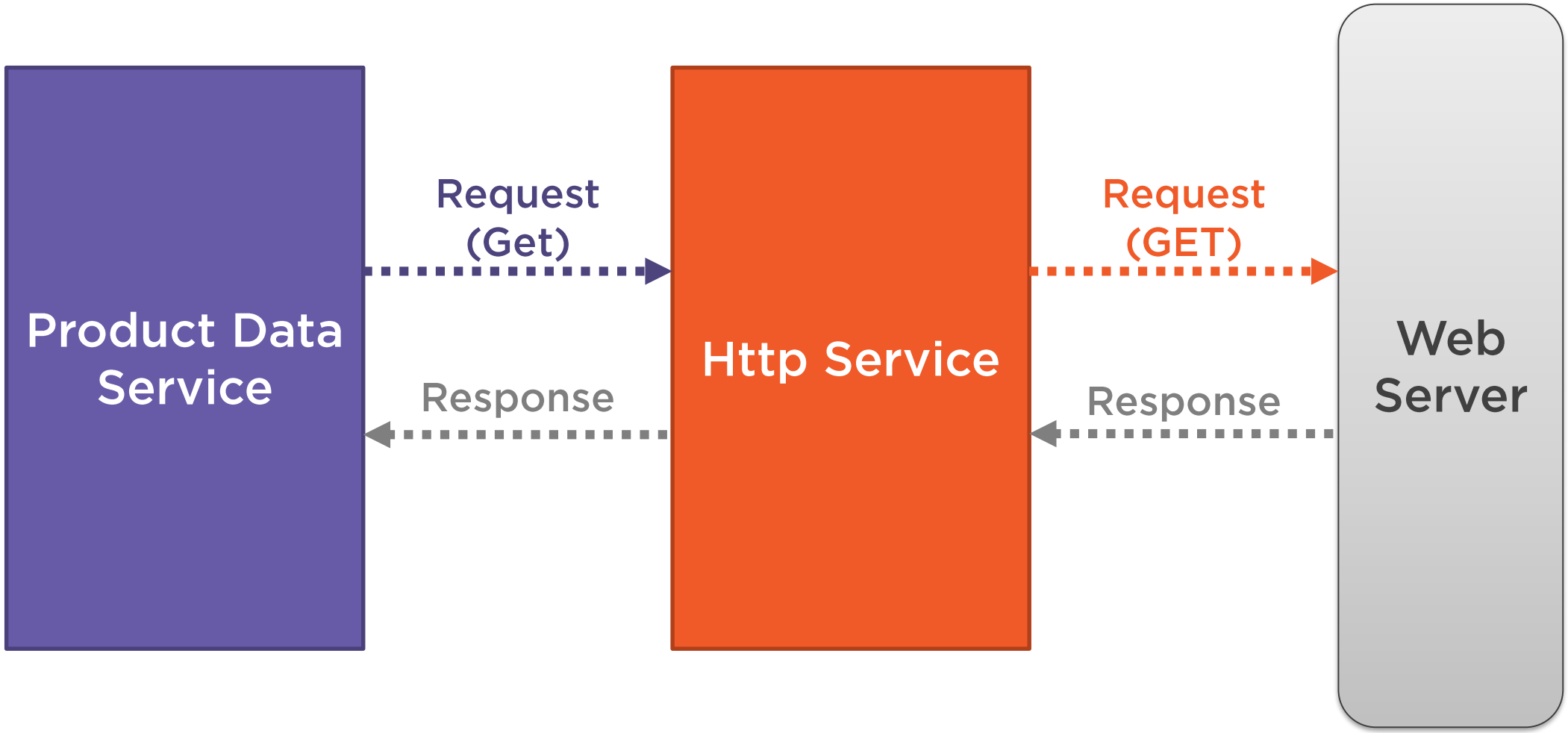
APM Sample Application Architecture



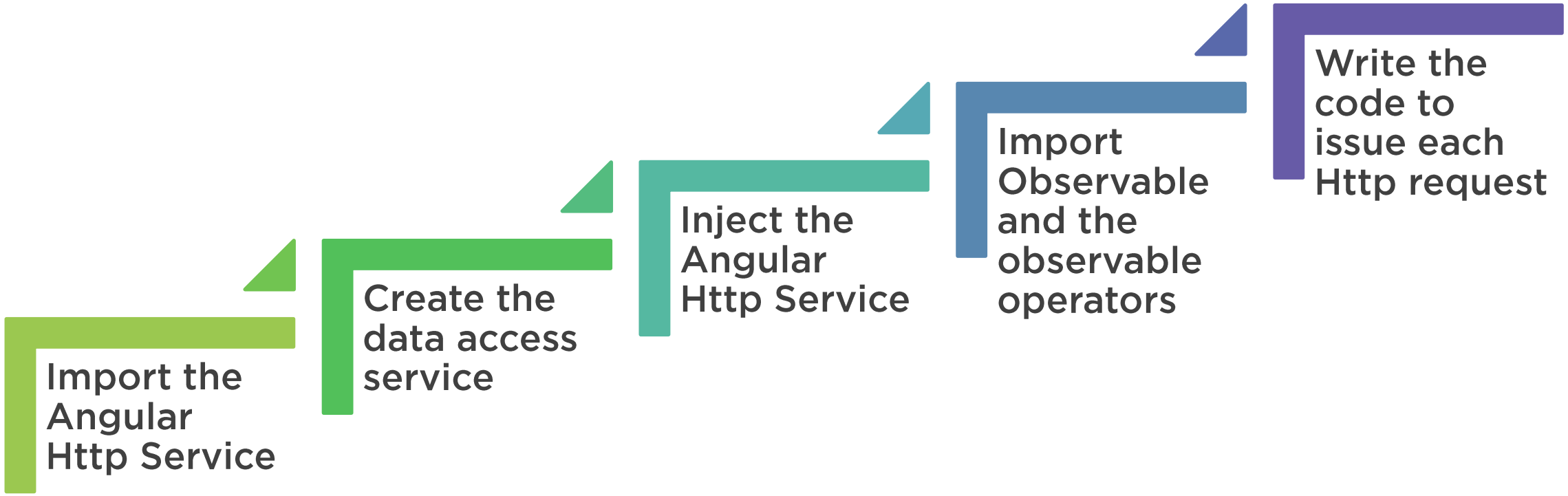
Why Build a Data Access Service?

- Separation of Concerns
- Reusability
- Data Sharing

Sending an HTTP Request



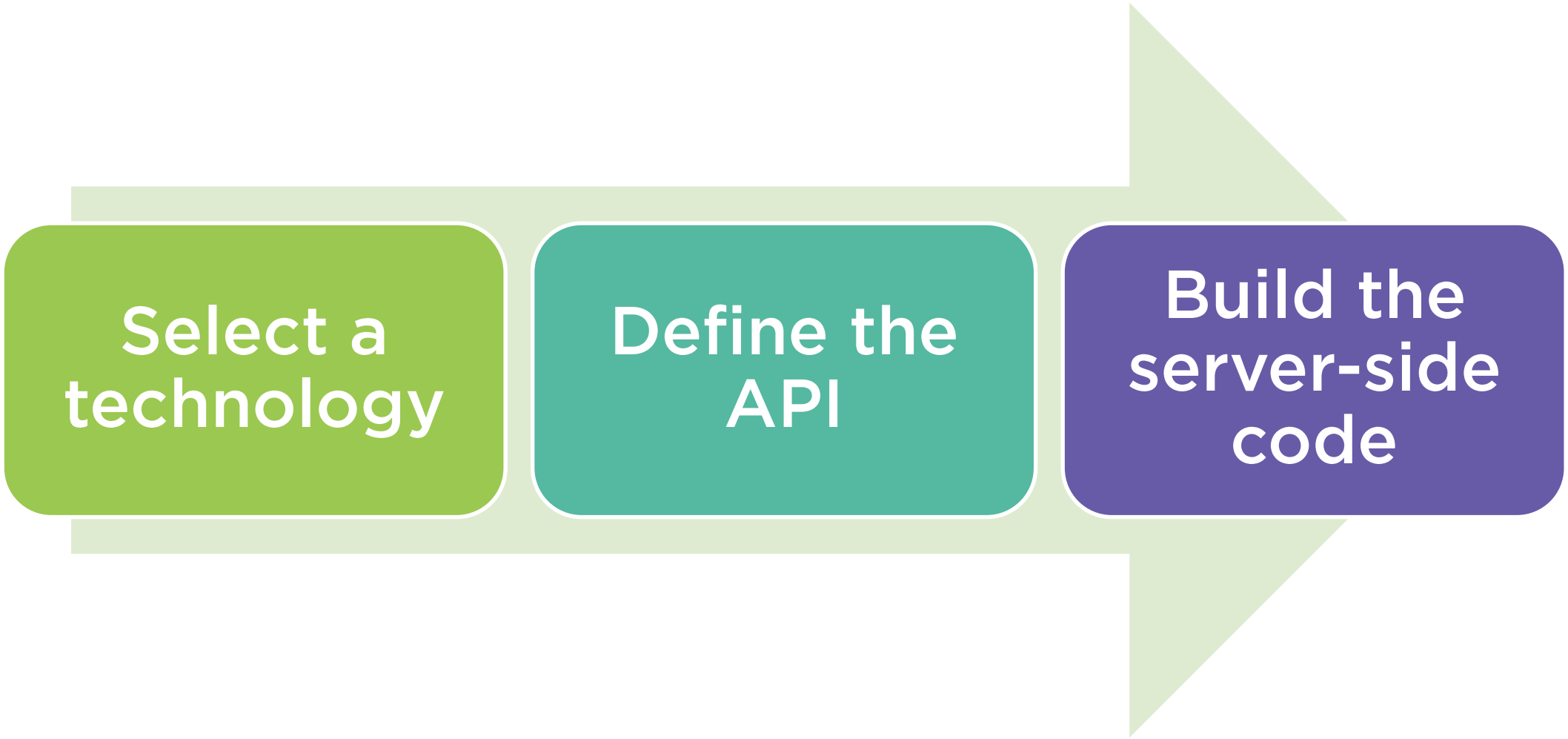
Steps to Building a Data Access Service



Demo

Building a Data Access Service

Setting up the Backend Server



Faking a Backend Server

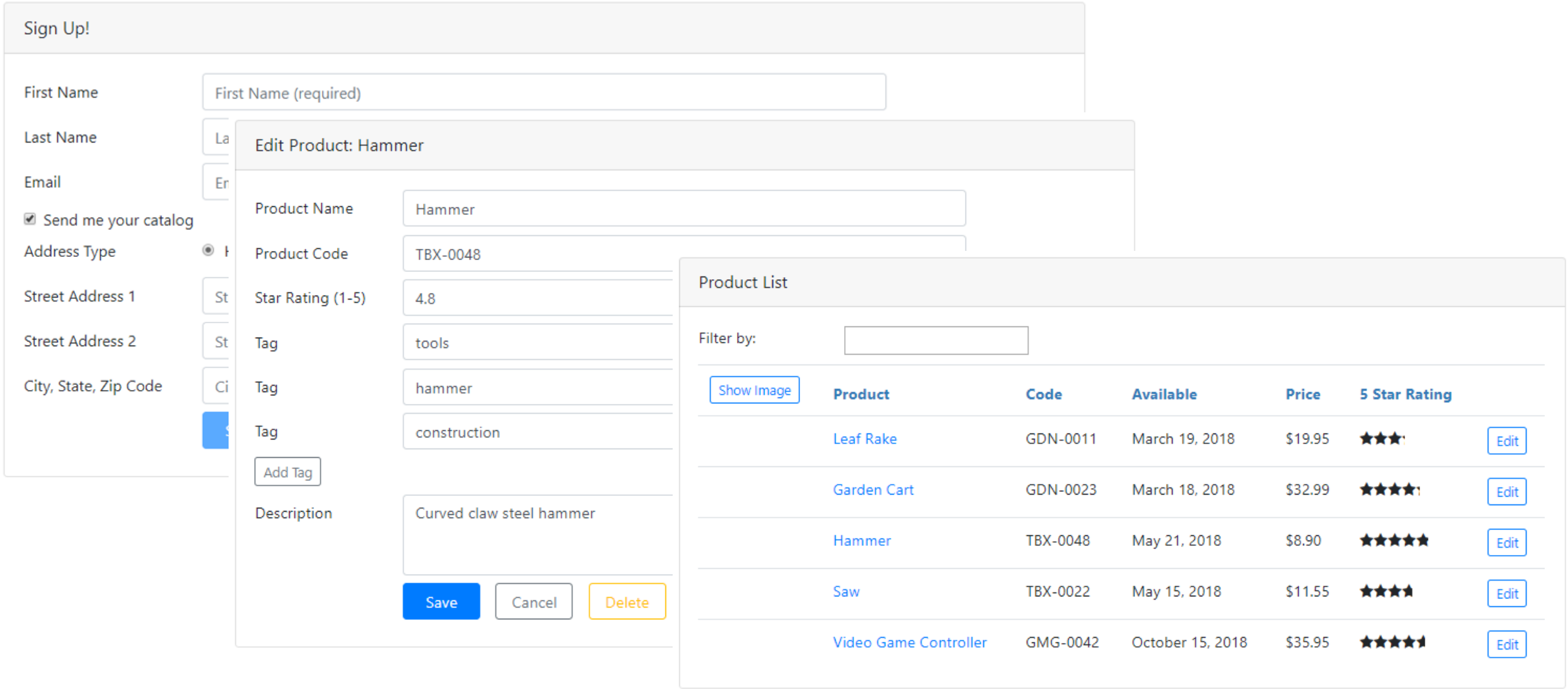
Directly return hard-coded data

Use a JSON file

Write our own code using MockBackend

Use angular-in-memory-web-api

Populating the Form with Data



HTTP Get Request

product.service.ts

```
...
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ProductService {
  private baseUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  getProduct(id: number): Observable<Product> {
     const url = `${this.baseUrl}/${id}`;
     return this.http.get<Product>(url);
  }
}
```

Calling the Data Access Service

product-edit.component.ts

```
...

constructor(private productService: ProductService) { }
...

getProduct(id: number): void {
    this.productService.getProduct(id)
        .subscribe({
            next: (product: Product) => this.displayProduct(product),
            error: err => this.errorMessage = err
        });
}
```

Demo

Populating the Form with Data: HTTP get

Demo

Populating the Form with Data: Subscribe

Saving Edits

Edit Product: Hammer

Product Name: Hammer

Product Code: TBX-0048

Star Rating (1-5): 4.8

Tag: tools — Delete Tag

Tag: hammer — Delete Tag

Tag: construction — Delete Tag

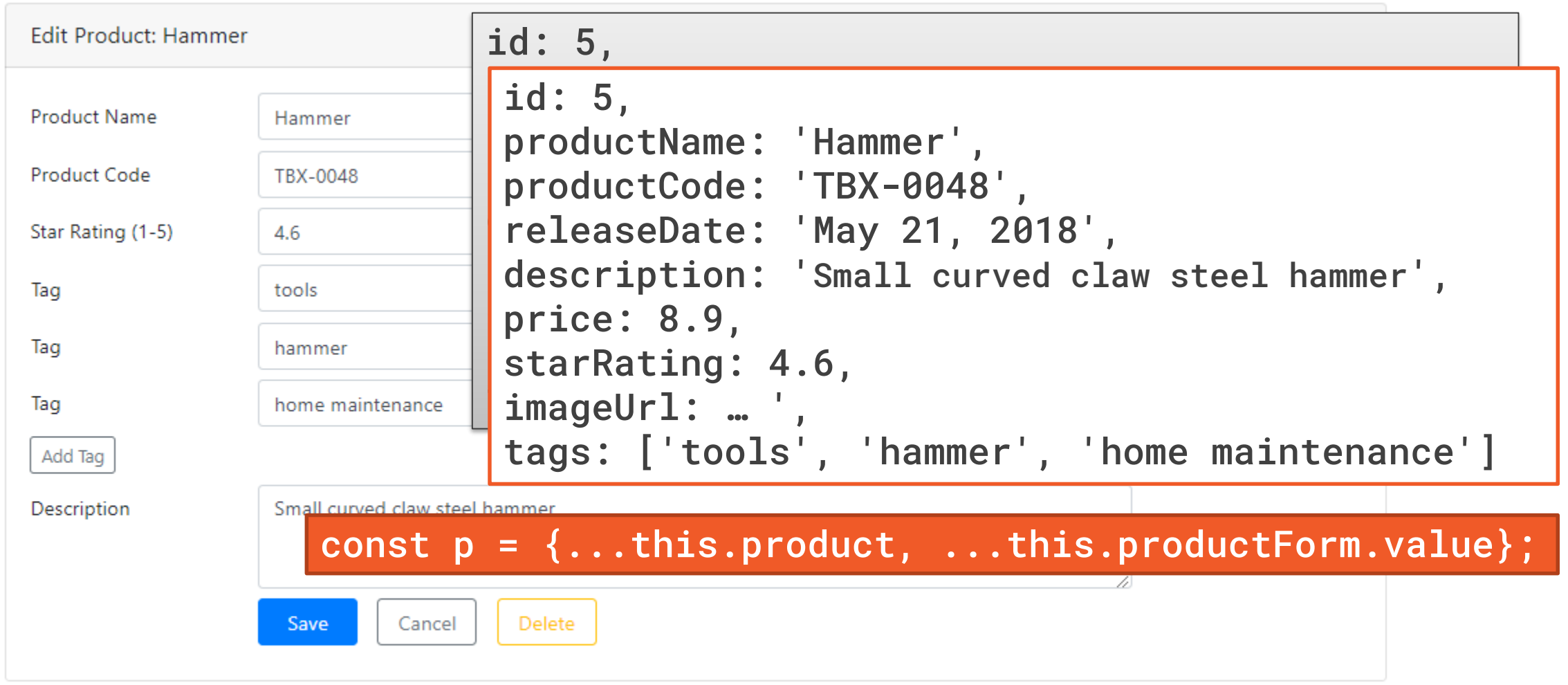
Add Tag

Description: Curved claw steel hammer

Save | Cancel | Delete

{ "productName": "Hammer", "productCode": "TBX-0048", "starRating": 4.8, "description": "Curved claw steel hammer", "tags": ["tools", "hammer", "construction"] }

Saving Edits



{ "productName": "Hammer", "productCode": "TBX-0048", "starRating": "4.6", "description": "Small curved claw steel hammer", "tags": ["tools", "hammer", "home maintenance"] }

Post vs Put

POST (api/products)

Posts data for a resource or set of resources

Used to:

- Create a new resource when the server assigns the Id
- Update a set of resources

Not idempotent

PUT (api/products/5)

Puts data for a specific resource with an Id

Used to:

- Create a new resource when the client assigns the Id
- Update the resource with the Id

Idempotent

HTTP Put Request

product.service.ts

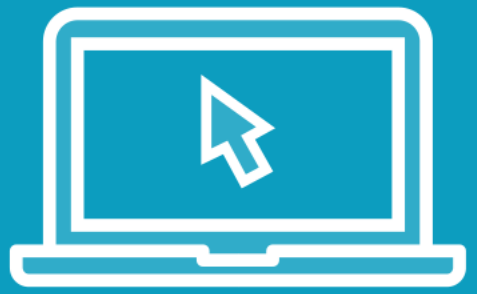
```
...
export class ProductService {
  private baseUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  updateProduct(product: Product): Observable<Product> {
    const headers = new HttpHeaders({ 'Content-Type': 'application/json' });

    const url = `${this.baseUrl}/${product.id}`;
    return this.http.put<Product>(url, product, { headers: headers });
  }
}
```

Calling the Data Access Service

product-edit.component.ts

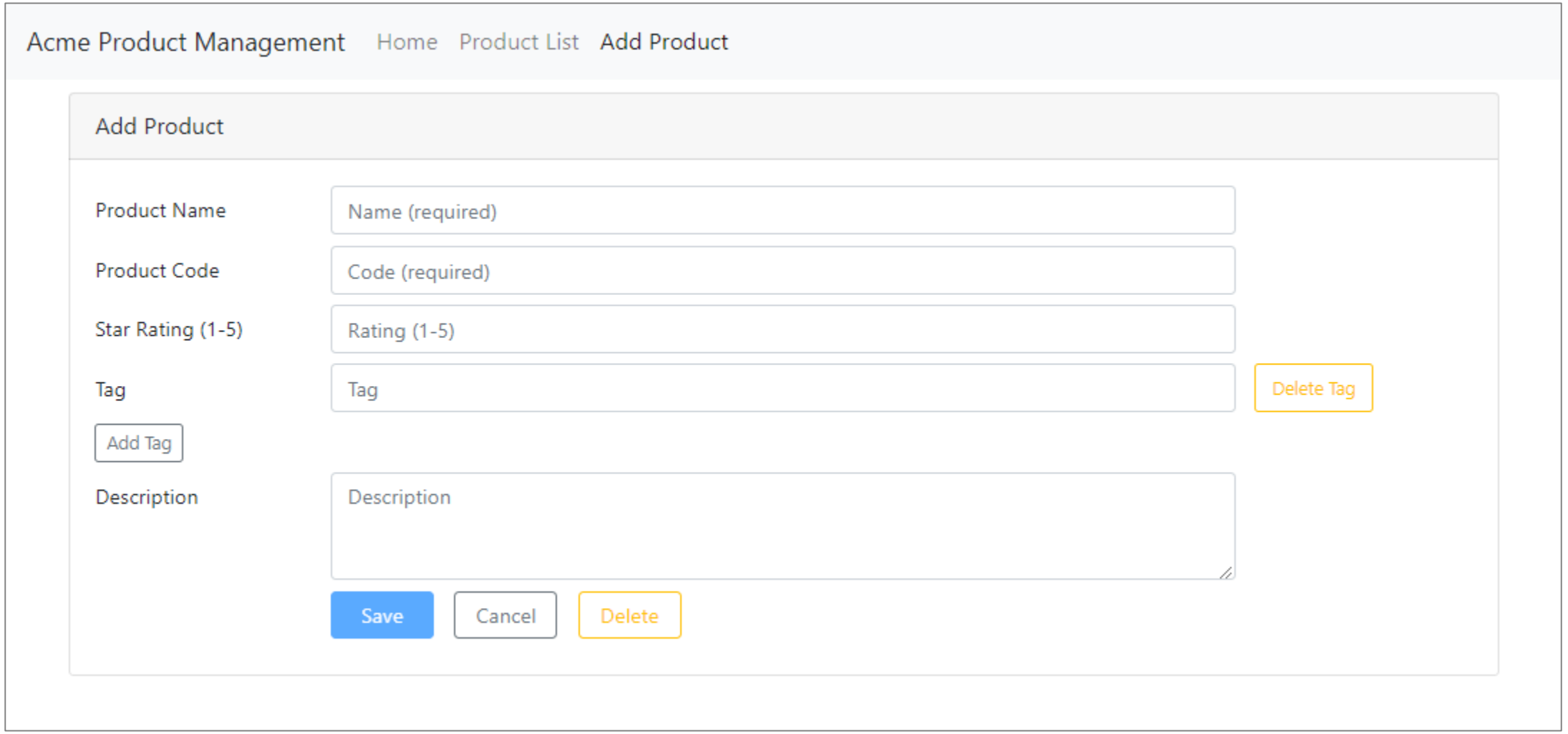
```
editProduct: void {
    this.productService.updateProduct(p)
        .subscribe({
            next: () => this.onSaveComplete(),
            error: err => this.errorMessage = err
        });
}
```

Demo



Saving Edits

Creating New Items



Initializing an Object

product.service.ts

```
initializeProduct(): Product {
  return {
      id: 0,
      productName: null,
      productCode: null,
      tags: [''],
      releaseDate: null,
      price: null,
      description: null,
      starRating: null,
      imageUrl: null
  }
}
```

HTTP Post Request

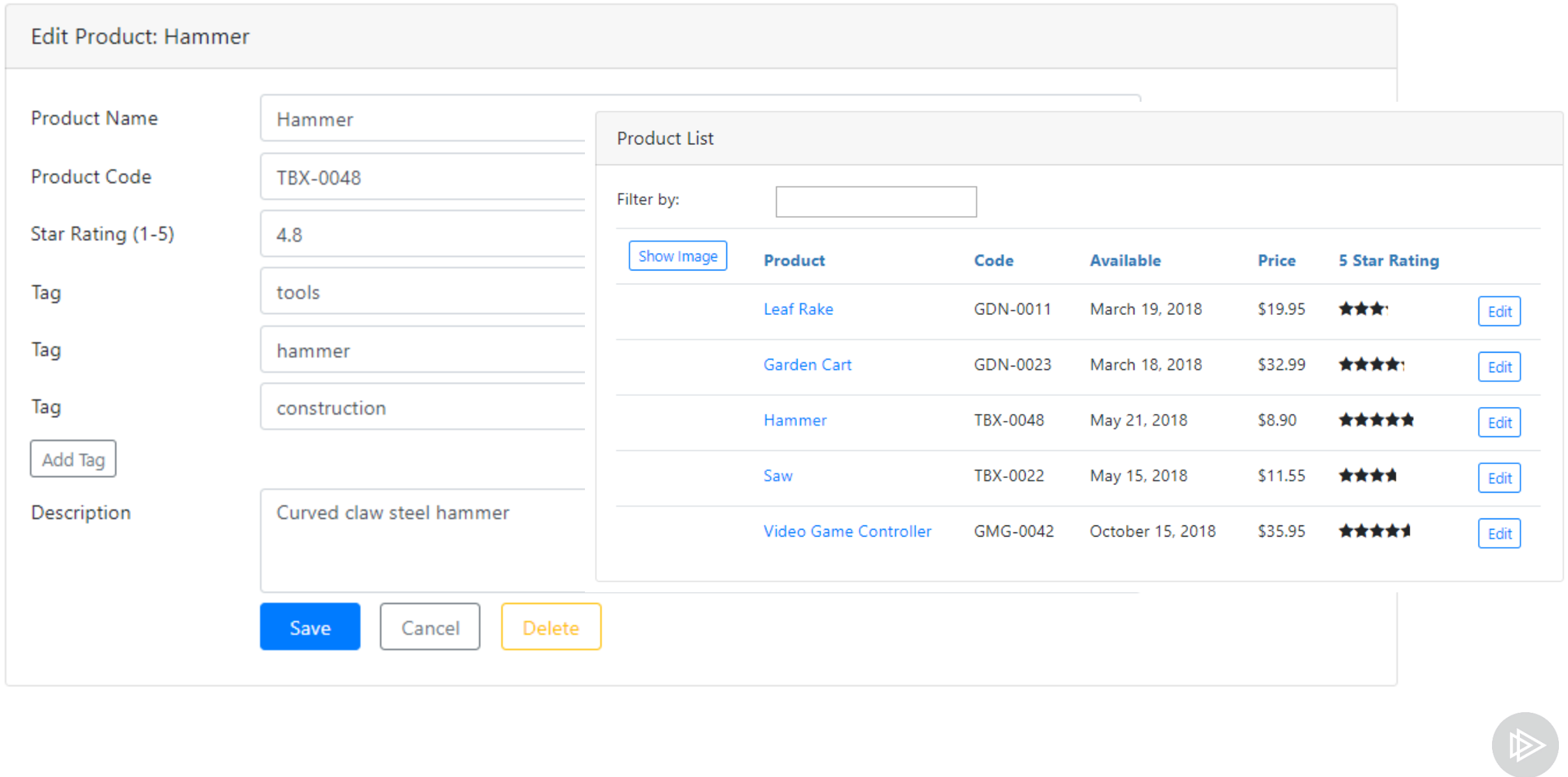
product.service.ts

```
...
export class ProductService {
  private baseUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  createProduct(product: Product): Observable<Product> {
   const headers = new HttpHeaders({ 'Content-Type': 'application/json' });

   return this.http.post<Product>(this.baseUrl, product,
                                  { headers: headers });
  }
}
```

Demo

Creating New Items

Deleting an Existing Item



HTTP Delete Request

product.service.ts

```
...
export class ProductService {
  private baseUrl = 'www.myWebService.com/api/products';

  constructor(private http: HttpClient) { }

  deleteProduct(id: number): Observable<{}> {
   const headers = new HttpHeaders({ 'Content-Type': 'application/json' });

   const url = `${this.baseUrl}/${id}`;
   return this.http.delete<Product>(url, { headers: headers });
  }
}
```

Calling the Data Access Service

product-edit.component.ts

```
deleteProduct: void {
    this.productService.deleteProduct(this.product.id)
        .subscribe({
            next: () => this.onSaveComplete(),
            error: err => this.errorMessage = err
        });
}
```

Demo

Deleting an Existing Item

CRUD Checklist: Import the Http Service

app.module.ts

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

@NgModule({
  imports: [ HttpClientModule ],
  ...
})
export class AppModule { }
```

Add HttpClientModule to the imports array of an Angular Module

CRUD Checklist: Data Access Service

Import what we need

product.service.ts

```
...
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { catchError, tap } from 'rxjs/operators';
```

CRUD Checklist: Data Access Service

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

product.service.ts

```
...
export class ProductService {

  constructor(private http: HttpClient) { } }
```

CRUD Checklist: Data Access Service

product.service.ts

```
...

getProduct ...
createProduct ...
updateProduct ...
deleteProduct ...
```

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

Create a method for each http request

CRUD Checklist: Data Access Service

product.service.ts

```
...

const url =
  `${this.baseUrl}/${id}`;
return this.http.get(url);
```

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

Create a method for each http request

Call the desired http method, such as get

- Pass in the Url

CRUD Checklist: Data Access Service

product.service.ts

```
...
const url =
 `${this.baseUrl}/${id}`;

return this.http.get<Product>(url)
 .pipe(
   .catchError(this.handleError)
 );
```

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

Create a method for each http request

Call the desired http method, such as get

- Pass in the Url

Add error handling

CRUD Checklist: Using the Service

product-edit.component.ts

```
...
constructor(private ps: ProductService) { }
```

Inject the Data Access Service

CRUD Checklist: Using the Service

product-edit.component.ts

```
...

this.ps.getProduct(id)
  .subscribe();
```

Inject the Data Access Service

Call the subscribe method of the returned observable

CRUD Checklist: Using the Service

product-edit.component.ts

```
...

this.ps.getProduct(id)
 .subscribe({
  next: (product: Product) =>
    this.onRetrieved(product)
 });
```

Inject the Data Access Service

Call the subscribe method of the returned observable

Provide a function to handle an emitted item

CRUD Checklist: Using the Service

product-edit.component.ts

```
...

this.ps.getProduct(id)
 .subscribe({
  next: (product: Product) =>
    this.onRetrieved(product),
  error: err =>
    this.errorMessage = err
 });
```

Inject the Data Access Service

Call the subscribe method of the returned observable

Provide a function to handle an emitted item

Provide an error function to handle any returned errors

Summary

Data Access Service

Creating Data

Reading Data

Updating Data

Deleting Data

Final Words

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/


Template-driven
Reactive

Angular Forms

Template-driven

Easy to use

Similar to AngularJS

Two-way data binding ->
Minimal component code

Automatically tracks form and
input element state

Reactive

More flexible ->
more complex scenarios

Immutable data model

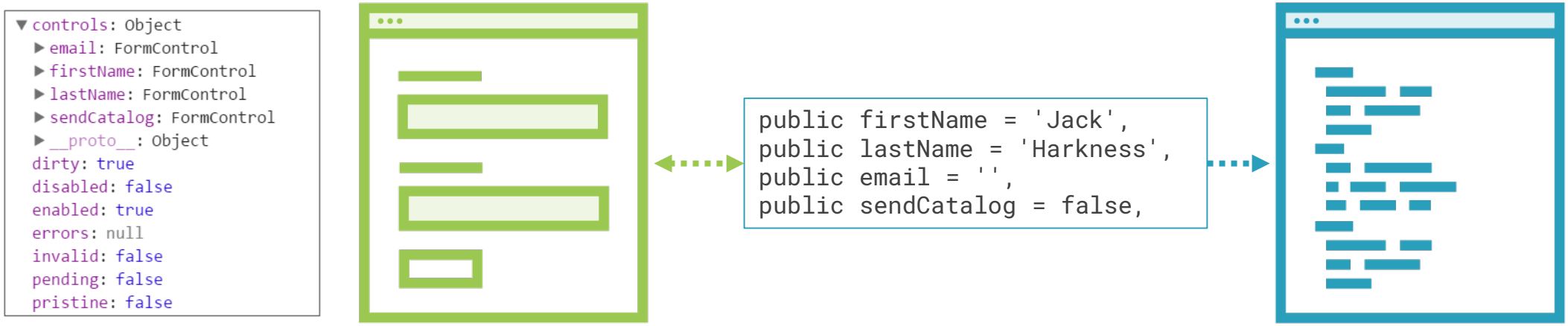
Easier to perform an action
on a value change

Reactive transformations ->
DebounceTime or DistinctUntilChanged

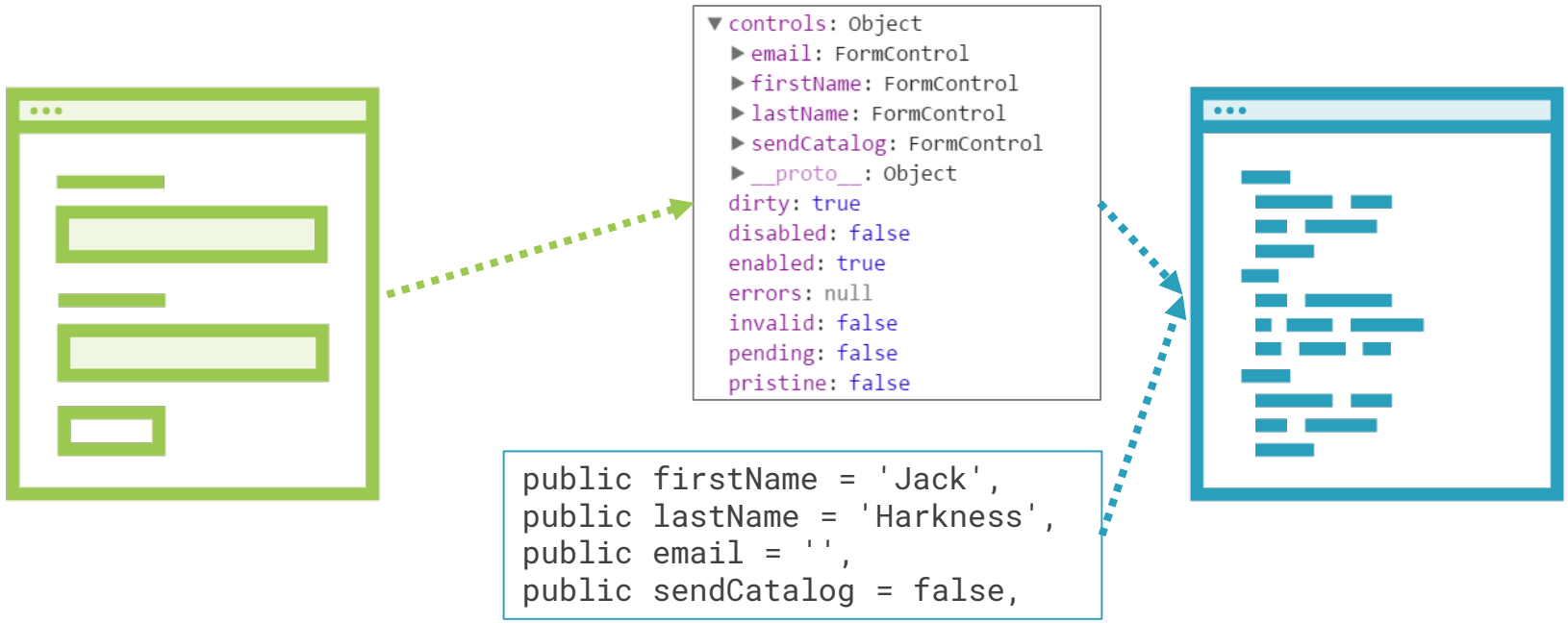
Easily add input elements dynamically

Easier unit testing

Template-driven



Reactive




GIVE
ME
MORE


RIGHT
WRONG
IT DEPENDS

Edit Product: Hammer

Product Name: Hammer

Product Code: TBX-0048

Star Rating (1-5): 4.6

Tag: tools — Delete Tag

Tag: hammer — Delete Tag

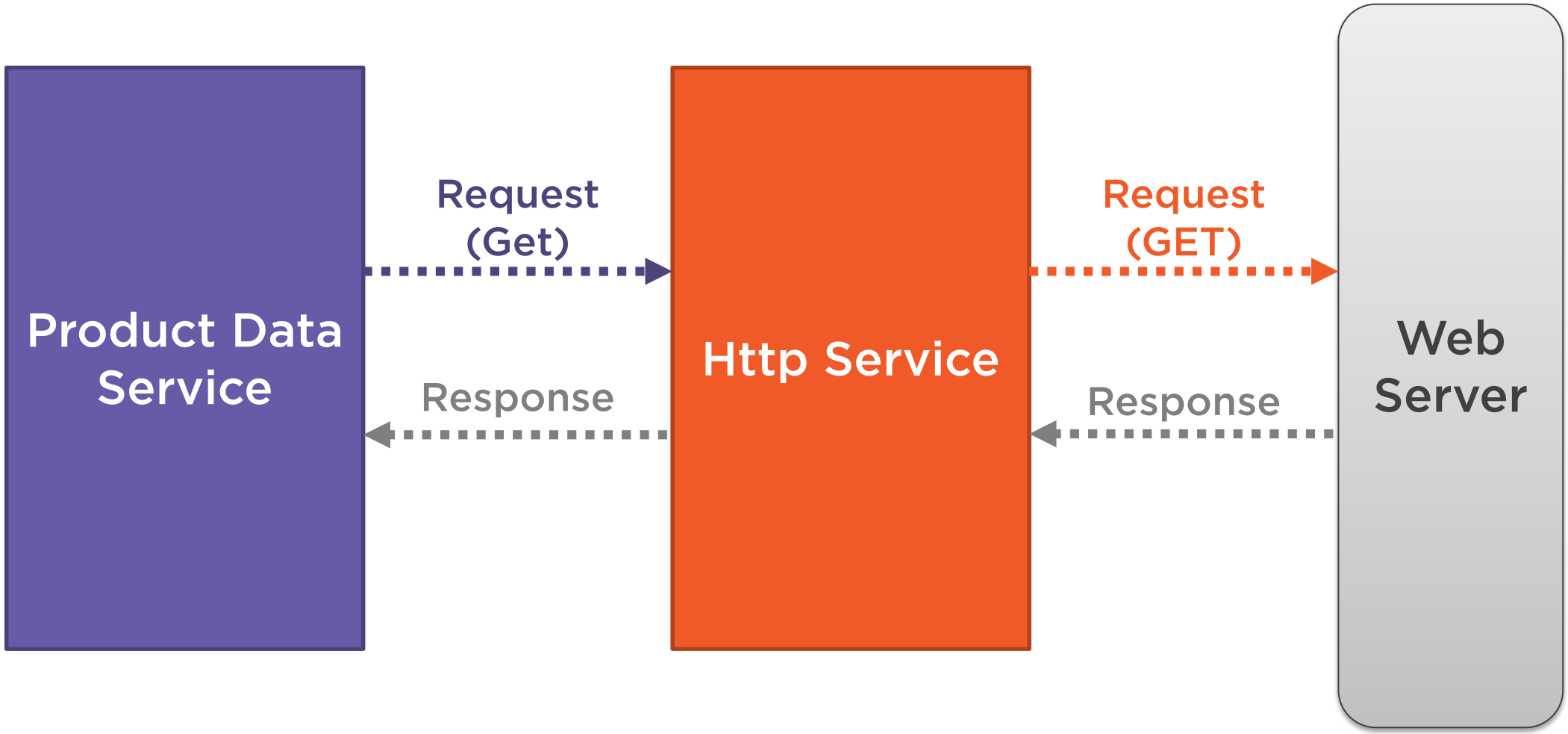
Tag: home maintenance — Delete Tag

Add Tag

Description: Small curved claw steel hammer

Save | Cancel | Delete

Sending an HTTP Request



Checklist

Steps and tips

Revisit as you build

Learning More

Beginner Angular Courses

- Angular: Getting Started
- Angular: First Look

Template-driven Forms Course

- Angular Forms

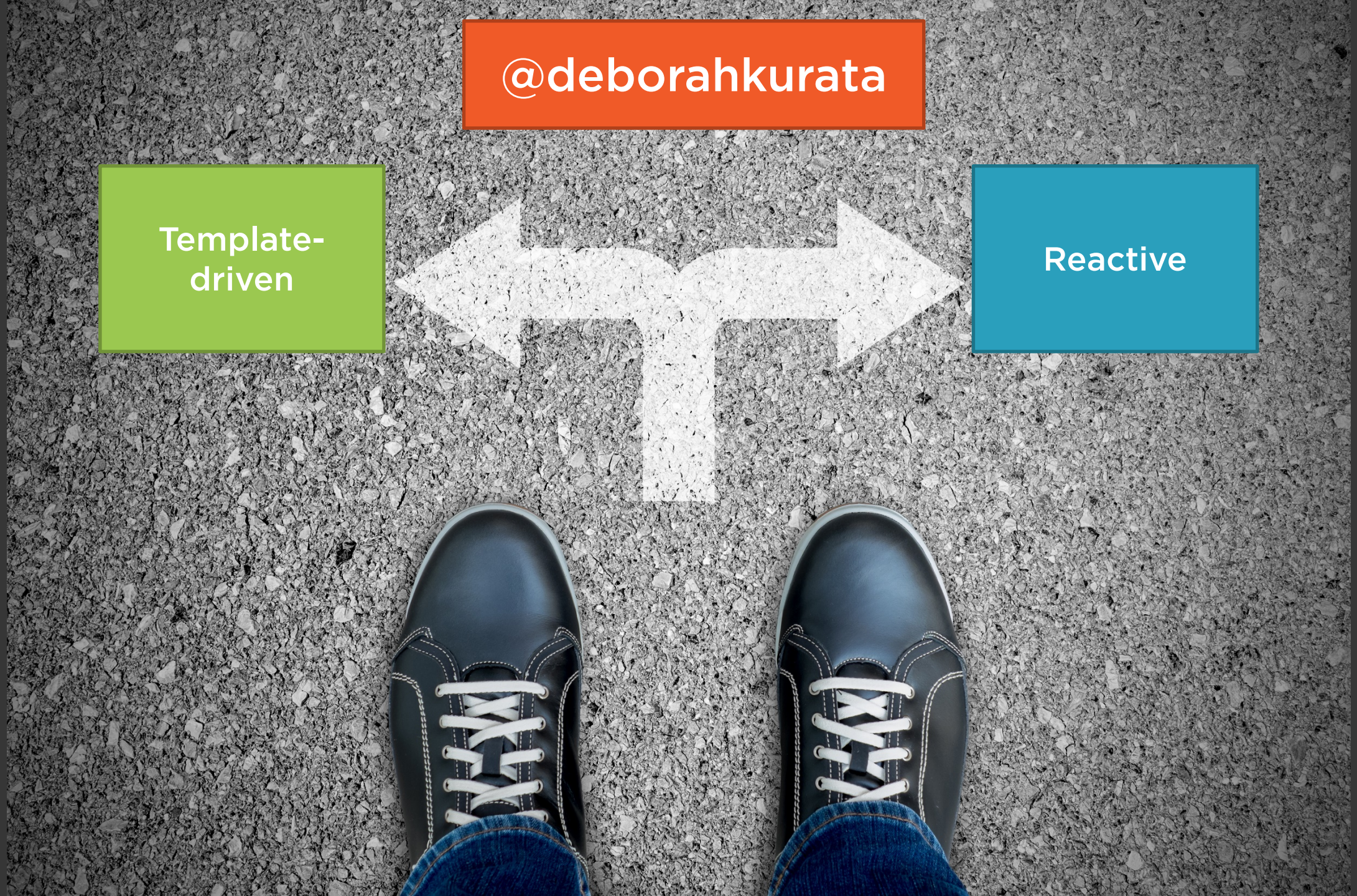
Angular Documentation

- Angular.io

Learning More

Intermediate Angular Courses

- Angular Routing
- Angular HTTP Communication
- Angular Component Communication


@deborahkurata
Template-driven
Reactive