



Day 32



Sample Form

[Name of Practice]
REGISTRATION FORM
(Please Print)

Select one of the following values

Mandatory

Fill in the blanks

Specific value type

Select one or more from the following

Attachments

Today's Date: 2/8/2012			PCP:					
PATIENT INFORMATION								
Patient's last name:		First:	Middle:	<input type="checkbox"/> Mr. <input type="checkbox"/> Mrs.	<input type="checkbox"/> Miss <input type="checkbox"/> Ms.	Marital status: Single <input type="checkbox"/> Mar <input type="checkbox"/> Div <input type="checkbox"/> Sep <input type="checkbox"/> Wid <input type="checkbox"/>		
Is this your legal name? <input type="checkbox"/> Yes <input type="checkbox"/> No		If not, what is your legal name?		(Former name):		Birth date:	Age:	Sex: <input type="checkbox"/> M <input type="checkbox"/> F
Street address:			Social Security no.:		Home phone no.: ()			
P.O. box:		City:		State:		ZIP Code:		
Occupation:		Employer:				Employer phone no.: ()		
Chose clinic because/referred to clinic by (Please check one box):				<input type="checkbox"/> Dr.		<input type="checkbox"/> Insurance plan		<input type="checkbox"/> Hospital
<input type="checkbox"/> Family		<input type="checkbox"/> Friend		<input type="checkbox"/> Close to home/work		<input type="checkbox"/> Yellow Pages		<input type="checkbox"/> Other
Other family members seen here:								

INSURANCE INFORMATION							
(Please give your insurance card to the receptionist.)							
Person responsible for bill:		Birth date:	Address (if different):		Home phone no.: ()		
Is this person a patient here? <input type="checkbox"/> Yes <input type="checkbox"/> No							
Occupation:		Employer:	Employer address:		Employer phone no.: ()		
Is this patient covered by insurance? <input type="checkbox"/> Yes <input type="checkbox"/> No							
Please indicate primary insurance		<input type="checkbox"/> [Insurance]	<input type="checkbox"/> [Insurance]	<input type="checkbox"/> [Insurance]	<input type="checkbox"/> [Insurance]	<input type="checkbox"/> [Insurance]	
<input type="checkbox"/> [Insurance]		<input type="checkbox"/> [Insurance]	<input type="checkbox"/> [Insurance]	<input type="checkbox"/> Welfare (Please provide coupon)		<input type="checkbox"/> Other	
Subscriber's name:		Subscriber's S.S. no.:		Birth date:	Group no.:	Policy no.:	Co-payment:



Terminology - Form

RSVP

Form

Controls

Button to submit

Name:

Email:

Phone:

Attending: ☐ YES ☐ NO

A diagram of an RSVP form. The form is enclosed in a dashed blue border. It contains labels for Name, Email, and Phone, each followed by a text input field with placeholder text. Below these is an "Attending" section with two radio buttons labeled "YES" and "NO". At the bottom is a "Send" button. Annotations include an arrow pointing to the top of the dashed box labeled "Form", an arrow pointing to the dashed box labeled "Controls", and an arrow pointing to the "Send" button labeled "Button to submit".



Forms

- Angular has 2 types of forms
 - Template driven
 - Reactive
- Template driven
 - Form (HTML) defines the form
 - Good for static forms
- Reactive
 - Form's logic is in the component (TypeScript)
 - More complex but more flexible

RSVP `rsvp.component.html`

Name:

Email:

Phone:

Attending: ☐ YES ☐ NO

Define the structure
of the form

Captures the value
of the form

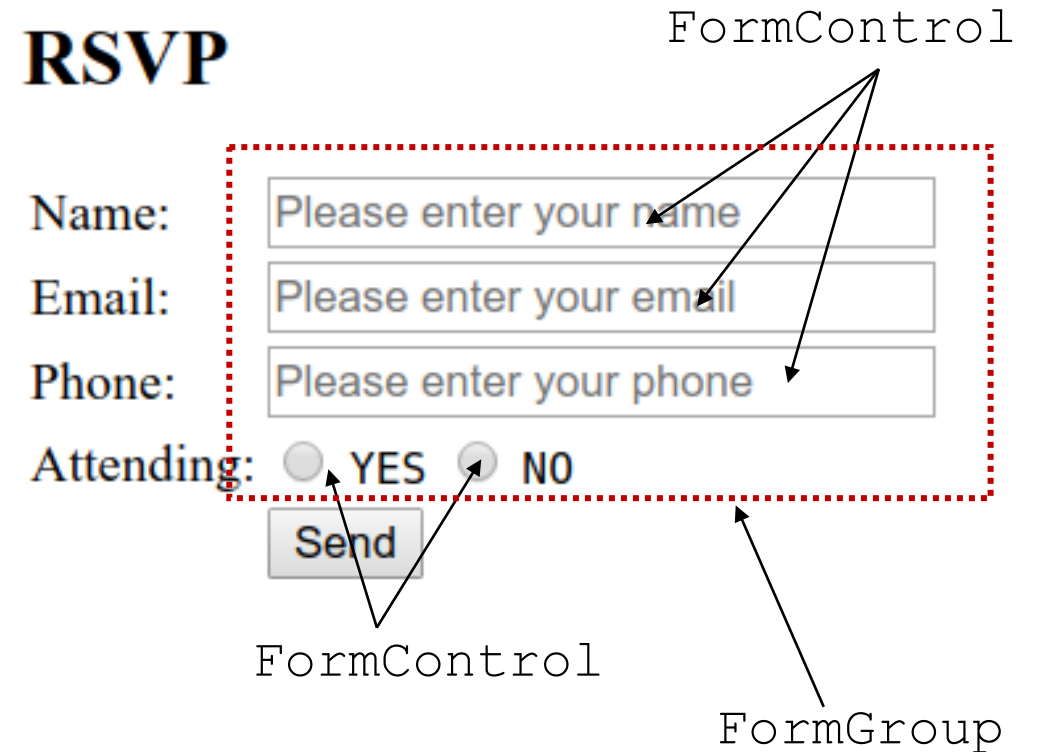
Form model

`rsvp.component.ts`



Terminology - Angular

- Control - **FormControl**
 - Represents a form control
 - Eg. input, textarea, checkboxes, etc
- Group - **FormGroup**
 - Holds one or more controls, array or group
 - Used for logical grouping eg customer info
- Array - **FormArray**
 - Grouping controls, groups and/or array





Configuring Reactive Forms Module

- Need to be added the module

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

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



Create the Form

`<form>` ← Form

Name: `<input type="text" name="name">`

...

Attending:

`<input type="radio" name="attending" value="yes">` YES

`<input type="radio" name="attending" value="no">` NO

`<button type="submit">`

Send

`</button>`

← Button to submit

`</form>`

Controls



Define the Form Model

```
import { FormBuilder, FormGroup } from '@angular/forms'
```

```
@Component({ ... })
```

```
export class RSVPComponent implements OnInit {
```

```
  rsvpForm: FormGroup
```

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

Helper service for
building controls,
groups and arrays

```
  ngOnInit() {
```

```
    this.rsvpForm = this.fb.group({
```

Create a group

```
      name: this.fb.control<string>(''),
```

```
      email: this.fb.control<string>(''),
```

```
      phone: this.fb.control<string>(''),
```

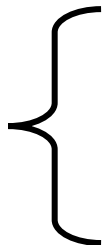
```
      attending: this.fb.control<string>('')
```

Control type

```
    })
```

```
  }
```

Control name





Map FormGroup to the Form

Bind the `rsvpForm` (FormGroup) to the form by binding it to the `formGroup` attribute

```
<form [formGroup]="rsvpForm">
```

```
  Name: <input type="text" FormControlName="name">
```

```
  Email: <input type="email" FormControlName="email">
```

```
  Phone: <input type="tel" FormControlName="phone">
```

```
  Attending:
```

```
    <input type="radio" value="yes" FormControlName="attending"> YES
```

```
    <input type="radio" value="no" FormControlName="attending"> NO
```

```
  <button type="submit">
```

```
    Send
```

```
  </button>
```

```
</form>
```

Map each form field to the controls defined in the FormGroup



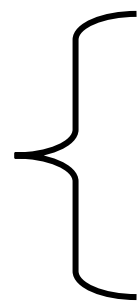
Repeat Groups

Add a new TODO to the table

Each of the group is a mini form

FormGroup

FormArray



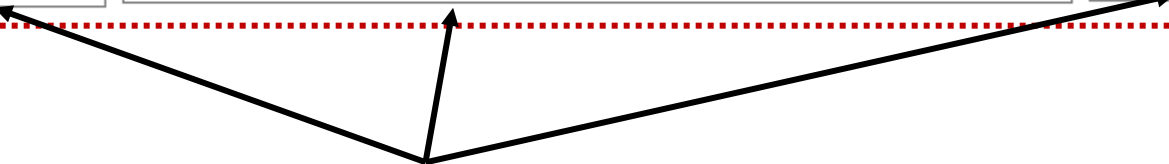
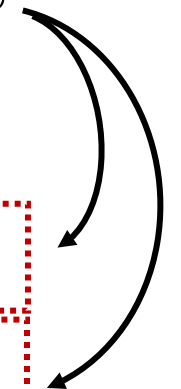
Add TODO

Date	Description	Priority
07 / 13 / 2020 <input type="button" value="x"/>	Go jogging this evening	Medium <input type="button" value="v"/>
mm / dd / yyyy		Low <input type="button" value="v"/>

Save

FormControl

Submit the form





TODO Form

Add a new 'row' when
this button is pressed

```
<form (ngSubmit)="processForm()">
  <button type="button">Add TODO</button>
  <tbody>
    <tr>
      <td> <input type="date"> </td>
      <td> <input type="text"> </td>
      <td>
        <select>
          <option value="low">Low</option>
          ...
        </select>
      </td>
    </td>
  </tbody>
  <button type="submit">Save</button>
</form>
```

Fragment of
HTML table

Repeat these



Defining the Model

```
@Component({ })
export class TodoComponent implements OnInit {
  todoForm: FormGroup
  todoArray: FormArray
  constructor(private fb: FormBuilder) { }
  ngOnInit() {
    this.todoArray = this.fb.array([])
    this.todoForm = this.fb.group({ todos: this.todoArray })
  }
  addTodo() {
    const todoGroup = this.fb.group({
      date: this.fb.control<Date>(new Date()),
      description: this.fb.control<string>(''),
      priority: this.fb.control<string>('')
    })
    this.todoArray.push(todoGroup)
  }
}
```

Add array to the form (group)

Construct the group (mini form) and add it to the array



Mapping the Model to the Form

```
<form [formGroup]="todoForm" (ngSubmit)="processForm()">
  <button type="button" (click)="addTodo()">Add TODO</button>
  <tbody formArrayName="todos">
    <tr *ngFor="let todo of todoArray.controls" [formGroup]="todo">
      <td> <input type="date" formControlName="date"> </td>
      <td> <input type="text" formControlName="description"> </td>
      <td>
        <select formControlName="priority">
          <option value="low">Low</option>
          ...
        </select>
      </td>
    </tr>
  </tbody>
  <button type="submit">Save</button>
</form>
```

Each array element is a mini form

Loop thru the todoArray. Each element is a FormGroup. Bind the FormGroup to the formGroup directive; use formControlName to bind the controls to the form fields



Reading the Form Values

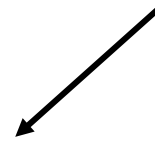
```
@Component({ ... })
export class RSVPComponent implements OnInit {
  rsvpForm: FormGroup

  constructor(private fb: FormBuilder) { }

  ngOnInit() { ... }

  processForm() {
    const rsvp = this.rsvpForm.value as RSVP
    // do something with rsvp
    ...
  }
}
```

Get the values from the group





Validation

- The process of validating that the values are correct before processing
- Types of validation
 - Syntactic - valid format, length of an entry, appropriate number range, mandatory field, etc.
 - Semantic - user name is not available, withdrawing more than the balance, etc.
- Syntactic validation are performed on the client/browser
 - Ensure that data is clean before submitting to server for processing
- Semantic validation are performed on the server
 - Typically require checking against database



Angular Form Validation

- Performs syntactic validation
- Comes with a set of build in validators
 - `required` - mandatory field
 - `requiredTrue` - required a checkbox to be checked
 - `email` - the entry is in a valid email format
 - `min, max` - validate a number range
 - `minLength, maxLength` - validates the minimum and maximum length
 - `pattern` - use regular expression to validate a field



Form Validation

```
import { FormBuilder, FormGroup, Validators } from '@angular/forms'

@Component({ ... })
export class RSVPComponent implements OnInit {
  rsvpForm: FormGroup

  constructor(private fb: FormBuilder) { }

  ngOnInit() {
    this.rsvpForm = this.fb.group({
      name: this.fb.control<string>('', [ Validators.required ] ),
      email: this.fb.control<string>('',
        , [ Validators.required, Validators.email ]),
      phone: this.fb.control<string>(''),
      attending: this.fb.control<string>('', [ Validators.required ])
    })
  }
}
```



Form Validity

- Is the form or a control is valid or invalid
 - `FormGroup.valid`
 - `FormGroup.invalid`
 - `FormControl.valid`
 - `FormControl.invalid`
- Is a field valid or invalid
 - `get()` returns the control from a group

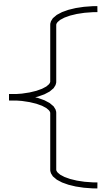

```
rsvpForm.get('email').valid
```
- Does a field has a specific type of error


```
rsvpForm.get('email').hasError('email')
```



Affordance - Hints and Error Messages

Display a message
if a control has a
specific error



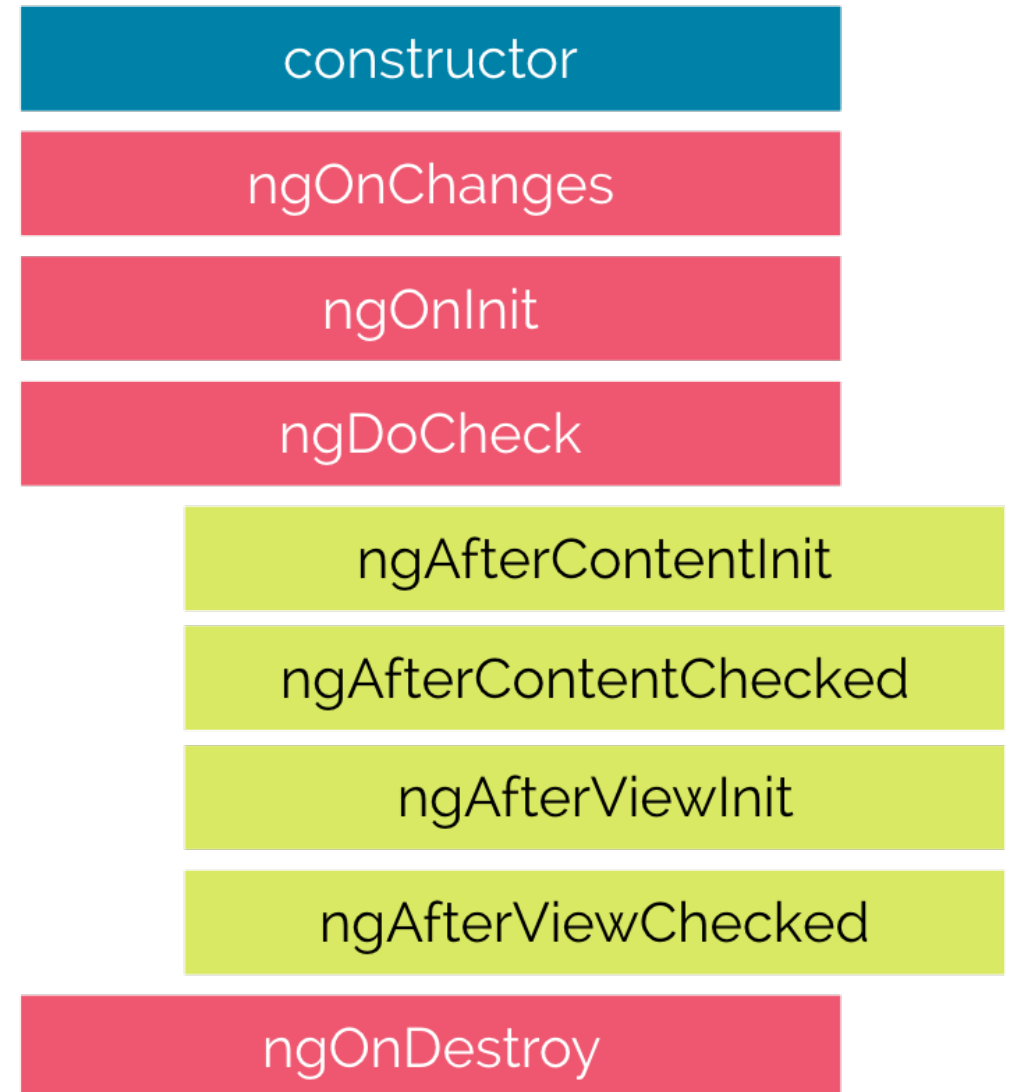
```
<form [formGroup]="rsvpForm">
  Name: <input type="text" formControlName="name">
  <div *ngIf="rsvpForm.get('name').hasError('required')">
    Please enter your name
  </div>
  Email: <input type="email" formControlName="email">
  <div *ngIf="rsvpForm.get('email').hasError('required')">
    Please enter your email
  </div>
  <div *ngIf="rsvpForm.get('email').hasError('email')">
    Please enter a valid email
  </div>
  ...
  <button type="submit" [disabled]="rsvpForm.invalid">
    Send
  </button>
</form>
```

Only enable the submit button if
the entire form is valid



Component Lifecycle

- Angular creates, renders and destroy the component
- Lifecycle hooks allow you to perform certain operations at these key moments
 - Eg. load data before the component is destroyed
- Implement one or more of these lifecycle interfaces
 - `OnInit` - called just after the component is created but before displaying
 - `OnDestroy` - called just before the component is destroyed
 - `OnChanges` - called when the `@Inputs` are updated because of attribute binding
 - `AfterViewInit` - called after the component's view has been created





Using Lifecycle Hooks

```
import { OnInit, OnDestroy } from '@angular/core';

export class AppComponent implements OnInit, OnDestroy {
  form!: FormGroup
  sub$: Subscription

  ngOnInit() {
    //From OnInit interface
    this.form = this.fb.group({ ... })
    this.sub$ = this.anObservable.subscribe(...)
  }

  ngOnDestroy() {
    //From OnDestroy interface
    this.sub$.unsubscribe()
  }
}
```

ngOnInit will
be called just
after component
is created

Initialize components eg.
create form, subscribe to
observables

ngOnDestroy
will be called just
before component
is destroyed

Clean up resources that can
cause memory leaks before
component is destroyed



Publishing to GitHub Pages

- GitHub Pages is a web hosting website for static pages
 - There are limitations
 - See <https://help.github.com/articles/what-is-github-pages/#usage-limits>

- Accessed with the following

`https://<username>.github.io/<repo_name>`

- GitHub will serve pages from **gh-pages** branch of your repository



Angular Deploy

- Angular has build in deployment to a few remote servers
 - Firebase hosting - `@angular/fire`
 - Azure - `@azure/ng-deploy`
 - Now - `@zeit/ng-deploy`
 - Github pages - `angular-cli-ghpages`
- Add deployment to project
 - See <https://www.npmjs.com/package/angular-cli-ghpages>
`ng add angular-cli-ghpages`



Deploying to Github Pages

- Compiled Angular is pushed to `gh-pages` branch
- Angular application is hosted under

`https://<username>.github.io/<repo_name>`

- Need to 'shift' Angular's base from `/` to `/<repo_name>/`
 - Otherwise loading resources will fail



Deploying to Github Pages

```
ng deploy --base-href=/myapp/
```

← Must have the last /

```
ng deploy
```



```
angular.json
```

```
...
"deploy": {
  "builder": "angular-cli-ghpages:deploy",
  "options": {
    "baseHref": "/myapp/"
  }
}
```



Deploying to Github Pages with Custom Domain

```
ng deploy --cname=myapp.acme.com
```

- Do not need to use `--base-href`
- `--cname` option will create a file called CNAME in the compiled Angular application
 - Contains your domain name
- Set custom domain manually if you forget to set custom domain during deployment
 - Settings (top right corner), Pages, Custom Domain
- Need to set CNAME record in DNS
 - You must own the domain!

```
myapp CNAME <username>.github.io
```



Unused



Custom Validators

```
import { ValidatorFn, AbstractControl, ValidationErrors }  
      from '@angular/forms'
```

```
const nonWhiteSpace = (ctrl: AbstractControl) => {  
  if (ctrl.value.trim().length > 0)  
    return (null)  
  return { nonWhiteSpace: true } as ValidationErrors  
}
```

Returns null if no error

Custom validators are functions with a single AbstractControl parameter

Returns a object indicating what error(s) have occurred
formCtrl.hasError() method checks this

```
emailCtrl = this.fb.control('', [  
  Validators.required, Validators.email,  
  Validators.nonWhiteSpace  
])
```