Workshop

Angular Forms

Forms

- Main components of most business applications
- Needed in most apps, e.g. login and registration

HTML5 Forms

Forms

<code>

A simple HTML5 form

```html
<form>
  <label for="title">Title:</label>
  <input type="text" id="title" name="title">

  <button type="submit">Speichern</button>
</form>
```

Forms

<code>

A simple HTML form with validation

```
<form>
  <label for="title">Title:</label>
  <input type="text" id="title" name="title" required>

  <button type="submit">Speichern</button>
</form>
```

HTML forms are not enough

Forms in Angular

Forms in Angular

- Possibilities of HTML5 forms are restricted
- We need to connect our form with our data model
- We have to update the data model on every change on the input elements

Reactive Forms

Reactive Forms

- Form and validation defined in component class
- Form will be accessed in template
- Create **FormGroups** composed of
    - FormControls
    - FormArrays
    - FormGroups

Reactive Forms

<code>

Import ReactiveFormsModule to get them working

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

@Component({
  selector: 'app-book-new',
  standalone: true,
  imports: [ReactiveFormsModule],
  …
})
export class BookNewComponent {
}
```

Reactive Forms - Create FormGroup

- Add FormControls to FormGroup

```
this.form = new FormGroup({
  title: new FormControl(''),
  isbn: new FormControl('')
});
```

Reactive Forms - Form Binding

- Connect FormGroup to template form with `formGroup`
- Connect FormControl with inputs with `formControlName`

```
<form [formGroup]="form" (ngSubmit)="submit()">
  ...
  <input type="text" formControlName="title">
  ...
</form>
```

Reactive Forms - Form Binding

- Connect FormGroup to template form with `formGroup`
- Connect FormControl with inputs with `formControlName`

```
<form [formGroup]="form" (ngSubmit)="submit()">
  ...
  <input type="text" formControlName="title">
  ...
</form>
```

Reactive Forms - FormBuilder

FormBuilder

- Helper to build up forms
- Creates a FormGroup with list of FormControls with optional Validators

Reactive Forms - Form Builder

- Everything exported by `@angular/forms`
- Inject FormBuilder in your class

```
import { FormBuilder, FormGroup } from '@angular/forms';
  ...
private readonly formBuilder = inject(FormBuilder)
```

Reactive Forms - Form Builder

- Declare a FormGroup
- Setup the group with the FormBuilder service

```
form: FormGroup;
...

this.form = this.formBuilder.group({
  'title': ['']
});
```

Task

Add a BookNew Form and Route

Outbreak

`TypedForms and nullability`

Typed Forms

Using types in your Angular forms can help catch errors early in the development process, improve code readability and maintainability, and enhance the overall developer experience.

Without Typings

```
this.form = new UntypedFormGroup({
  title: new UntypedFormControl(''),
  isbn: new UntypedFormControl('')
});
```

```
const title = form.value.title.text
```

Typed Forms

- Angular will automatically infer the type if we set default Values

```
const bookTitle = new FormControl('Angular.DE');
// → FormControl<string|null>

bookTitle.setValue(11); // Error
```

Specifying an Explicit Type

```
this.form = new FormGroup<BookForm>({
  title: new FormControl<string | null>(''),
  isbn: new FormControl<string | null>('')
});
```

```
interface BookForm {
  title: FormControl<string | null>,
  isbn: FormControl<string | null>,
}
```

Nullability

- Type of FormControl can become *null* at any time

```
const bookTitle = new FormControl<string | null>('Angular.DE');
bookTitle.reset();

console.log(bookTitle); // null
```

Reactive Forms - Form Builder

- Declare a **nonNullable** FormGroup

```
private readonly formBuilder = inject(FormBuilder)


this.form = this.formBuilder.nonNullable.group({ ... });
```

Reactive Forms - Form Builder

→ Declare a **nonNullable** FormGroup

```
private readonly formBuilder = inject(NonNullableFormBuilder)


this.form = this.formBuilder.group({ ... });
```

Nullability

We can set a FormControl as nonNullable

```
const bookTitle = new FormControl<string>('Angular.DE', { nonNullable:
true});
bookTitle.reset();

console.log(bookTitle); // Angular.DE
```

Form Validation

Reactive Forms - Validators

Validator throws an error as soon as validation fails

```
this.form = this.formBuilder.group({
   'author':   ['', [Validators.required]],
   'title':    ['', [Validators.required]],
   'subtitle': [''],
   'abstract': [''],
});
```

Built-in Validators

- min / max
- required / requiredTrue
- email
- minLength / maxLength
- pattern
- nullValidator
- compose / composeAsync

Reactive Forms - Form Validation

A form can have several different statuses. Each possible status is returned as a string literal

```
type FormControlStatus = 'VALID' | 'INVALID' | 'PENDING' | 'DISABLED';
```

FormControl Status

pristine	Indicates whether the control is in its initial state
touched	Indicates whether the control has been blurred or focused
dirty	Indicates whether the control has changed

Reactive Forms - Form Validation

Helper functions to get controls and errors

```
form.get('title')?.hasError('required')
form.get('author')?.hasError('required')

form.hasError('required', 'title')
form.hasError('required', 'author')
```

Reactive Forms - Form Validation

Helper functions to get controls and errors

```
<input formControlName="title" />
 @if(form.get('title')?.dirty &&
     form.get('title')?.hasError('required')) {
   <small>Please insert a title.</small>
}
<button type="submit" [disabled]="form.invalid">Save</button>
```

Task

Add Form Validation

Reading Form Values

Reactive Forms - Form Binding

- Whenever the user clicks the submit button ***ngSubmit*** will be emitted

```
<form [formGroup]="form" (ngSubmit)="submit()">
  ...
  <button type="submit">Save</button>
</form>
```

Reactive Forms

<code>

Read Values from FormGroup

```
submit() {
  // returns the actual value (without disabled controls)
  const book = this.form.value;
  // returns the actual value (with disabled controls)
  const book = this.form.getRawValue();
}
```

Reactive Forms

<code>

Extend the BookApiService

```
@Component({…})
export class BookNewComponent {

  private readonly bookApiService = inject(BookApiService)

  submit() {
    this.bookApiService.create(this.form.getRawValue()).subscribe()
  }
}
```

Task

Extend BookApiService

Custom Validators

Defining custom validators

The built-in validators don't always match the exact use case of your application, so you sometimes need to create a custom validator.

Validator functions can be either synchronous or asynchronous

Validator functions

Sync validators	Synchronous functions that take a control instance and immediately return either a set of validation errors or null.
Async validators	Asynchronous functions that take a control instance and return a Promise or Observable that later emits a set of validation errors or null.

Validator function

<code>

Defining custom Validator

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

export function validAuthorName(): ValidatorFn {
    return (control: AbstractControl) : ValidationErrors | null => {
        const value = control.value || null;
        const hasNumeric = /[0-9]+/.test(value); // Check if value has numerics
        return hasNumeric ? { invalidAuthor : true } : null;
    }
}
```

Validator function

<code>

Adding to FormGroup

```
this.form.nonNullable.group({
      author:  ['', [Validators.required, validAuthorName()]],
      title: ['', [Validators.required]],
      ....
    }, )
```

Task

Write custom validator

Creating FormControls dynamically

FormArray

- FormArray just like a FormGroup is also a form container
- Does not require us to know all the controls upfront
- Can have undetermined number of form controls
- Each control will have a numeric position in the form controls array
- FormControls can be added or removed dynamically

FormArray API

controls	This is an array containing all the controls that are part of the array
length	This is the total length of the array
at(index)	Returns the form control at a given array position
push(control)	Adds a new control to the end of the array
removeAt(index)	Removes a control at a given position of the array

Using FormArray

```
form = this.formBuilder.group({
       ... other form controls ...
       authors: this.formBuilder.array([''])
});
get authors(): FormArray {
     return this.form.controls["authors"] as FormArray;
}
```

Using FormArray (with Validator)

```
form = this.formBuilder.group({
      ... other form controls ...
      authors: this.formBuilder.array([''', [Validators.required]])
});
```

Removing FormControl to FormArray

```
deleteAuthor(authorIndex: number) {
  this.authors.removeAt(authorIndex);
}
```

Adding FormControl to FormArray

```
addAuthor() {
  this.authors.push(new FormControl('', [Validators.required]));
}
```

Binding to template

<code>

```
<ng-container formArrayName="authors">
  @for (author of authors.controls; track $index) {
    <input [formControlName]="$index" placeholder="Author"/>
    <button (click)="deleteAuthor($index)">
      Remove Author
    </button>
  }
</ng-container>
```

Binding to template

```
<button type="button" (click)="addAuthor()">
   Author hinzufügen
</button>
```

Task

Provide multiple author FormControls

symetics

Forms in Angular

Ausgeblendet / wenig relevant

Two types:

1. Template-driven forms
    - → Created and configured only in the HTML code
    - → No default access via TypeScript
2. Reactive Forms
    - → Created and configured in the TypeScript code
    - → Connected to a form in HTML code

Ausgeblendet /
wenig relevant

Template-driven forms

Template-driven forms

Ausgeblendet / wenig relevant

- For building simple forms
- No need to get programmatic access to it
- Easiest way to create a form in Angular

Template-driven forms

Import FormsModule to get them working

Ausgeblendet / wenig relevant

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

@NgModule({
  imports: [FormsModule]
})
```

Template-driven forms

Ausgeblendet / wenig relevant

- FormsModule provides necessary directives
- Directives:
  - ngForm
  - ngModel

Template-driven forms - ngF

Ausgeblendet / wenig relevant

- Automatically added to each form
- Angular representation of form is a FormGroup, which contains multiple FormControls (input, textarea, select)
- Possibility to get access to the form and input states in the template

Template-driven forms - ngF

Ausgeblendet /
wenig relevant

Get access to your form with ngForm

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

Stores the formGroup on a local template variable

Template-driven forms - Dat

Ausgeblendet / wenig relevant

- What to do?
  - Load data into a form and update inputs
  - Update changes on form input elements
- We need something like this:

```
<input [value]="book.title" (input)="book.title=$event.target.value">
```

Template-driven forms - Dat

**Ausgeblendet /
wenig relevant**

- What to do?
  - Load data into a form and update inputs
  - Update changes on form input elements
- We need something like this:

That looks like boilerplate code

```
<input [value]="book.title" (input)="book.title=$event.target.value">
```

Template-driven forms - ngM

Ausgeblendet / wenig relevant

- ngModel Directive
  - Two-Way Data Binding
  - Reading and writing values from inputs
  - Creates a FormControl for the input element and registers it at the form

Template-driven forms - ngM

Ausgeblendet / wenig relevant

First optimization

```
<input
  [value]="book.title"
  (input)="book.title=$event.target.value">
```

```
<input
  [ngModel]="book.title"
  (ngModelChange)="book.title=$event">
```

[value] = “book.title”

EventEmitter property that returns the input box value when it fires

Template-driven forms - ngM

Ausgeblendet / wenig relevant

First optimization

```
<input
  [value]="book.title"
  (input)="book.title=$event.target.value">
```

That looks like boilerplate code!

```
<input
  [ngModel]="book.title"
  (ngModelChange)="book.title=$event">
```

[value] = “book.title”

EventEmitter property that returns the input box value when it fires

Template-driven forms - ngM

Ausgeblendet / wenig relevant

Second optimization

```
<input
  [ngModel]="book.title"
  (ngModelChange)="book.title=$event"
  name=”title”>
```

[value] = “book.title”

EventEmitter property that returns the input box value when it fires

```
<input [(ngModel)]="book.title" name=”title”>
```

Template-driven forms - ngM

Ausgeblendet / wenig relevant

Second optimization

```
<input
  [ngModel]="book.title"
  (ngModelChange)="book.title=$event">
```

[value] = “book.title”

EventEmitter property that returns the input box value when it fires

Why does that exist?

```
<input [(ngModel)]="book.title">
```

When it’s so easy!

Template-driven forms - ngM

Ausgeblendet / wenig relevant

usage of e.g. transformation

```
<input
  [ngModel]="book.isbn"
  (ngModelChange)="transformIsbn($event)">
```

Template-driven forms - vali

Ausgeblendet /
wenig relevant

- → Let the user know what he is doing wrong
- → Guide a user through your form
- → Increases user experience

Template-driven forms - vali

Ausgeblendet /
wenig relevant

- access to state of a form input
- easy visualisation of the control (in-)valid state

```
<form>
  <input [(ngModel)]="book.title" name="title">
</form>
```

Form Validation - ngModel / classes

Ausgeblendet / wenig relevant

State	Class if true	Class if false
Control **has been visited**	.ng-touched	.ng-untouched
Control value **has changed**	.ng-dirty	.ng-pristine
Control value **is valid**	.ng-valid	.ng-invalid

Template-driven forms - ngM

Ausgeblendet / wenig relevant

Get access to your input with ngModel

```
<form #form="ngForm">
  <input [(ngModel)]="book.isbn" name="isbn" #isbn="ngModel">
</form>
```

Stores the control of the model on a local template variable

Form Validation - ngForm an

Ausgeblendet /
wenig relevant

bind local variable
to Control Instance

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

```
{
  "value": {
    "title": ''
  },
  "controls": {
    …
    "title": {…}
    …
  },
  "dirty": true,
  "valid": false,
  "pristine": false,
  "touched": true
}
```

Form Validation - ngForm

Use state via local template variables to defi

Ausgeblendet / wenig relevant

```
<form #form="ngForm">
  <input type="text" required [(ngModel)]="book.title"
         name="title" #title="ngModel">

  <div [hidden]="!title.errors?.[‘required’] || title.pristine">
    Title is required
  </div>
</form>
```

Template-driven forms - vali

Ausgeblendet / wenig relevant

- Example of possible input validation via ty
  - text
  - email
  - date
  - number
  - month
  - url
  - tel

```
<form>
  <input
    type="email"
    [(ngModel)]="book.author.email"
    name="email">
</form>
```

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

Template-driven forms - vali

Ausgeblendet / wenig relevant

- example of possible input validation via attribute
  - required
  - maxlength
  - minlength
  - Pattern
  - requiredTrue

```
<form>
  <input
    type="text" required
    [(ngModel)]="book.title"
    name="title">
</form>
```

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input

Form Validation - ngForm

Use ngSubmit to trigger a form submit

Ausgeblendet / wenig relevant

```
<form #form="ngForm" (ngSubmit)="save(form.value)">
  ...
  <button type="submit" [disabled]="form.invalid">
    Submit
  </button>
  ...
</form>
```