DYNAMIC REACTIVE FORMS FROM JSON

#FVGDEV

DAVIDE SERAFINI

Freelance Fullstack Developer | Digital Consultant
(...and I work with VISUP® too!)

LinkedIn | Stack Overflow | Facebook | Instagram | Twitter

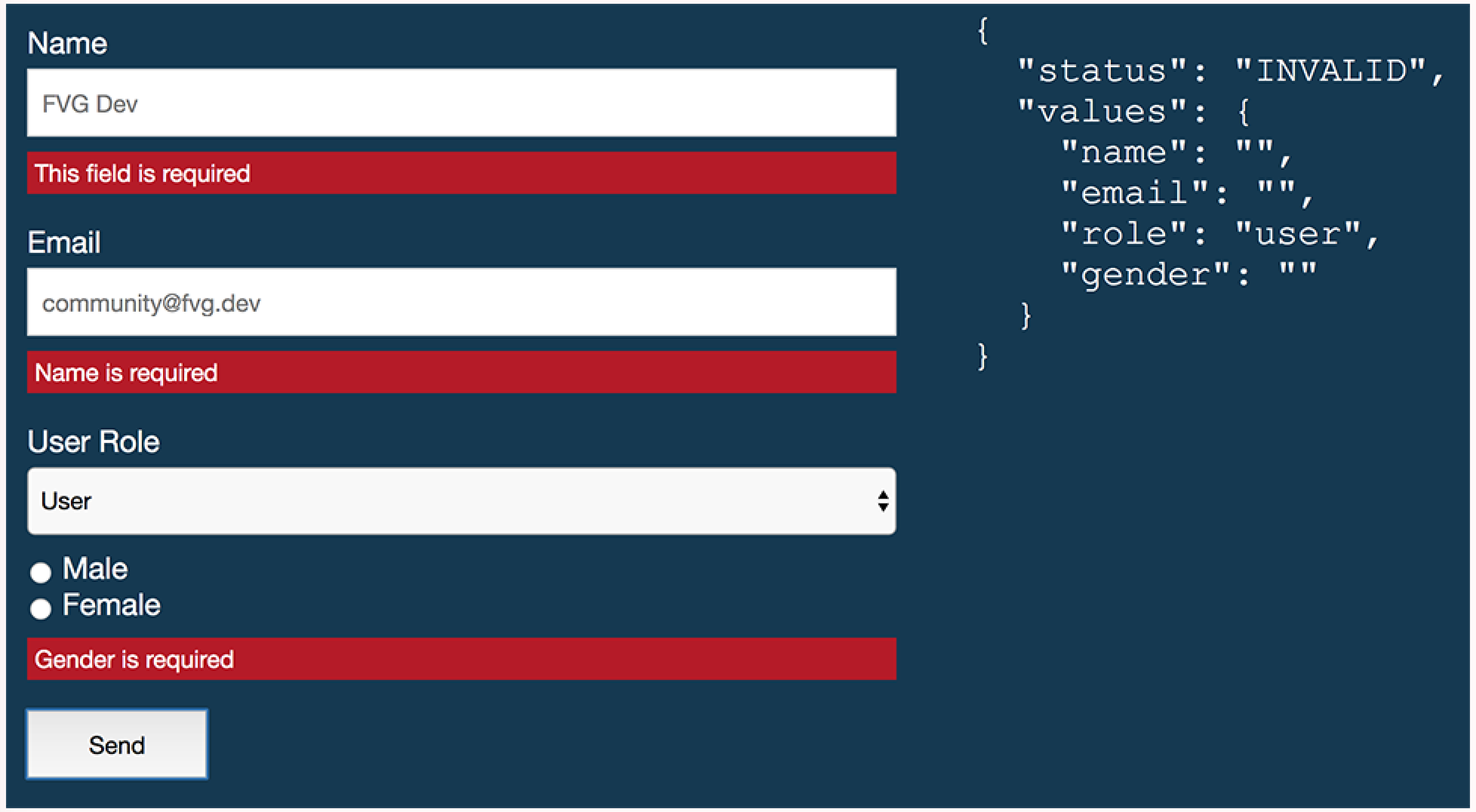
WHY SHOULD I DO THIS?

Using a JSON to instantiate a form can be useful when:

1. You have a lot of forms to create
2. It might be easier to maintain and update a JSON file
3. Forms change a lot based on API responses
4. Forms can be "backend driven"

FINAL RESULT: DEFINITION

- Name, Email, Gender are **required**
- Email must be correctly **formatted**



Name

FVG Dev

This field is required

Email

community@fvg.dev

Name is required

User Role

User

● Male
● Female

Gender is required

Send

```
{
  "status": "INVALID",
  "values": {
    "name": "",
    "email": "",
    "role": "user",
    "gender": ""
  }
}
```

STEP 1: THE SIMPLE WAY - TYPESCRIPT

Configuration of form

```
ngOnInit() {
  this.form = this.formBuilder.group({
    name: ['', Validators.required],
    email: ['', Validators.compose([Validators.required,
                                    Validators.email])],
    role: ['user'],
    gender: ['', Validators.required]
  });
}
```

STEP 1: THE SIMPLE WAY - TEMPLATE

Definition of form and email input in template

```
<form [formGroup]="form" (submit)="onSubmit()">
  <div class="form-row">
    <label for="email">
      Email
    </label>
    <input type="email"
           formControlName="email"
           placeholder="community@fvg.dev"
           id="email"
           required>

    <div *ngIf="form.get('email').hasError('required') &&
                form.touched" class="errorMessage">
      Name is required
    </div>
    <div *ngIf="form.get('email').hasError('email') &&
                form.touched" class="errorMessage">
      Check email format
    </div>
  </div>
```

STEP 2: THE NGFOR WAY - TYPESCRIPT

JSON used for configuration: name input

```json
{
  "label": "Name",
  "type": "input",
  "name": "name",
  "value": "",
  "inputType": "text",
  "placeholder": "FVG Dev",
  "validations": [{
    "name": "required",
    "message": "Name is required"
  }, {
    "name": "minlength",
    "message": "Name is too short",
    "value": 5
  }],
}
```

STEP 2: THE NGFOR WAY - JSON

JSON used for configuration: role select

```json
{
  "label": "Role",
  "type": "select",
  "name": "role",
  "value": "user",
  "options": [{
    "value": "user",
    "text": "User"
  }, {
    "value": "admin",
    "text": "Admin"
  }]
}
```

STEP 2: THE NGFOR WAY - JSON

JSON used for configuration: gender radios

```
{
  "label": "",
  "type": "input",
  "inputType": "radio",
  "name": "gender",
  "value": "",
  "options": [{
    "value": "m",
    "text": "Male"
  }, {
    "value": "f",
    "text": "Female"
  }],
  "validations": [{
    "name": "required",
    "message": "Gender is required"
  }]
}
```

STEP 2: THE NGFOR WAY - INTERFACES

Interface used for JSON

```
export interface DynamicFormFieldConfig {
  // Fields for any form field
  label: string;
  type: string;
  name: string;
  value: string;
  placeholder?: string;
  validations?: Validation[];

  // Fields for inputs
  inputType?: string;

  // Fields for selects, input radio
  options?: Option[];
}
```

STEP 2: THE NGFOR WAY - INTERFACES

Interface used for JSON

```
export interface Option {
  value: string;
  text: string;
}

export interface Validation {
  name: string;
  message: string;
  value?: number | string;
}
```

STEP 2: THE NGFOR WAY - TYPESCRIPT

Configuration of form

```
ngOnInit() {
  this.form = this.createDynamicFormGroup(
                this.config,
                this.formBuilder.group({})
              );
}
createDynamicFormGroup(
  formConfig: DynamicFormFieldConfig[],
  formGroup: FormGroup): FormGroup {

  formConfig.forEach((fieldConfig: DynamicFormFieldConfig) => {
    if (fieldConfig.type !== 'button') {
      const formControl = this.formBuilder.control(
        fieldConfig.value,
        this.addValidation(fieldConfig.validations)
      );
      formGroup.addControl(fieldConfig.name, formControl);
    }
  });
  return formGroup;
}
```

STEP 2: THE NGFOR WAY - TYPESCRIPT

Configuration of form

```
addValidation(validations: Validation[] = []) {
  const validatorsList: ValidatorFn[] = [];
  validations.forEach(validation => {
    const name = validation.name;
    const value = validation.value;

    switch (name) {
      case 'required':
      case 'email':
        validatorsList.push(Validators[name]);
        break;
      case 'minlength':
        validatorsList.push(
          Validators.minLength(value as number)
        );
        break;
    }
  });
  return validatorsList;
}
```

STEP 2: THE NGFOR WAY - TEMPLATE

First, loop on the JSON configuration object

```
<div class="form-row" *ngFor="let fieldConfig of config">
```

Then, check control type

```
<ng-container *ngIf="fieldConfig.type === 'input'">
```

Create input

```
<input *ngIf="fieldConfig.inputType !== 'radio'"
        [id]="fieldConfig.name"
        [formControlName]="fieldConfig.name"
        [type]="fieldConfig.inputType"
        [placeholder]="fieldConfig.placeholder || ''">

<ng-container *ngIf="fieldConfig.inputType === 'radio'">
  <input *ngFor="let option of fieldConfig.options"
          [formControlName]="fieldConfig.name"
          [type]="fieldConfig.inputType"
          [value]="option.value">{{ option.text }}
</ng-container>
```

STEP 2: THE NGFOR WAY - TEMPLATE

Create select

```
<ng-container *ngIf="fieldConfig.type === 'select'">
  <select [id]="fieldConfig.name"
          [formControlName]="fieldConfig.name">
    <option *ngFor="let option of fieldConfig.options"
            [value]="option.value">
            {{ option.text }}
    </option>
  </select>
</ng-container>
```

STEP 2: THE NGFOR WAY - TEMPLATE

Labels

```
<label [for]="fieldConfig.name" *ngIf="fieldConfig.label">
  {{ fieldConfig.label }}
</label>
```

Error messages

```
<div *ngFor="let validation of fieldConfig.validations">
  <div
    *ngIf="form.get(fieldConfig.name).hasError(validation.name)
           && form.touched"
    class="errorMessage">
    {{ validation.message }}
  </div>
</div>
```

STEP 2: THE NGFOR WAY - TEMPLATE

Summary

```
<ng-container *ngIf="fieldConfig.type === 'select'">
  <label [for]="fieldConfig.name" *ngIf="fieldConfig.label">
    {{ fieldConfig.label }}
  </label>

  <select *ngIf="fieldConfig.type === 'select'"
          [id]="fieldConfig.name"
          [formControlName]="fieldConfig.name">
    <option *ngFor="let option of fieldConfig.options"
            [value]="option.value">{{ option.text }}</option>
  </select>

  <div *ngFor="let validation of fieldConfig.validations">
    <div
      *ngIf="form.get(fieldConfig.name).hasError(validation.name)
             && form.touched"
      class="errorMessage">{{ validation.message }}</div>
  </div>
</ng-container>
```

STEP 3: THE DIRECTIVE WAY - FORM

```
<form [formGroup]="form"
                (submit)="onSubmit()"
                class="form-wrapper">
  <ng-container
      *ngFor="let dynamicFieldConfig of dynamicFormConfig;"
      appDynamicField
      [fieldConfig]="dynamicFieldConfig"
      [formGroup]="form">
  </ng-container>
</form>
```

STEP 3: THE DIRECTIVE WAY - FIELD COMPONENTS

Create component for Inputs e Selects, with the code used before, accepting also a JSON in @Input()
First, create an interface defining the DynamicFormField

```
export interface DynamicFormField {
  fieldConfig: DynamicFormFieldConfig;
  formGroup: FormGroup;
}
```

STEP 3: THE DIRECTIVE WAY - FIELD COMPONENTS

Create components for Select, Inputs and Button: DynamicSelectComponent, DynamicInputComponent and DynamicButtonComponent

```
export class DynamicSelectComponent
                implements DynamicFormField {
  fieldConfig: DynamicFormFieldConfig;
  formGroup: FormGroup;

  constructor() { }
}
```

STEP 3: THE DIRECTIVE WAY - FIELD COMPONENTS

Create components for Select, Inputs and Button: DynamicSelectComponent, DynamicInputComponent and DynamicButtonComponent

```
<div [formGroup]="formGroup" class="form-row">
  <label [for]="fieldConfig.name">
    {{ fieldConfig.label }}
  </label>
  <select [id]="fieldConfig.name"
          [formControlName]="fieldConfig.name">
    <option *ngFor="let option of fieldConfig.options"
            [value]="option.value">{{ option.text }}</option>
  </select>

  <div *ngFor="let validation of fieldConfig.validations">
    <div
      *ngIf="formGroup.get(fieldConfig.name).hasError(validation.
             && formGroup.touched"
      class="errorMessage">{{ validation.message }}</div>
  </div>
</div>
```

STEP 3: THE DIRECTIVE WAY - DIRECTIVE

Create components for Select, Inputs and Button: DynamicSelectComponent, DynamicInputComponent and DynamicButtonComponent

```
const componentMapper = {
  input: DynamicInputComponent,
  select: DynamicSelectComponent,
  button: DynamicButtonComponent
};
```

STEP 3: THE DIRECTIVE WAY - DIRECTIVE

Create components for Select, Inputs and Button: DynamicSelectComponent, DynamicInputComponent and DynamicButtonComponent

```
export class DynamicFieldDirective implements OnInit {
  @Input() fieldConfig: DynamicFormFieldConfig;
  @Input() formGroup: FormGroup;
  componentRef: ComponentRef<DynamicFormField>;

  constructor(
    private componentFactoryResolver: ComponentFactoryResolver,
    private viewContainerRef: ViewContainerRef
  ) { }

  ngOnInit() {
    const componentToCreate = componentMapper[this.fieldConfig.ty
    const componentFactory = this.componentFactoryResolver
      .resolveComponentFactory<DynamicFormField>(componentToCreat
    this.componentRef = this.viewContainerRef
      .createComponent(componentFactory);
    this.componentRef.instance.fieldConfig = this.fieldConfig;
    this.componentRef.instance.formGroup = this.formGroup;
  }
}
```

SUMMARY

ComponentFactoryResolver: used to resolve a ComponentFactory for each dynamic component

ComponentFactory: "an object that knows how to create a component"

ViewContainerRef: used to get a reference to the view that will host the dynamic components

ViewContainerRef.createComponent: used to get a reference to the newly created component

ComponentRef: reference to the newly created component

componentRef.instance.property: to set an Input property of the newly created component

CODE

https://github.com/davideserafini/ng-dynamic-forms-json/

THANK YOU!

LinkedIn | Stack Overflow | Facebook | Instagram | Twitter