Forms in Angular

What Are Forms in Angular?

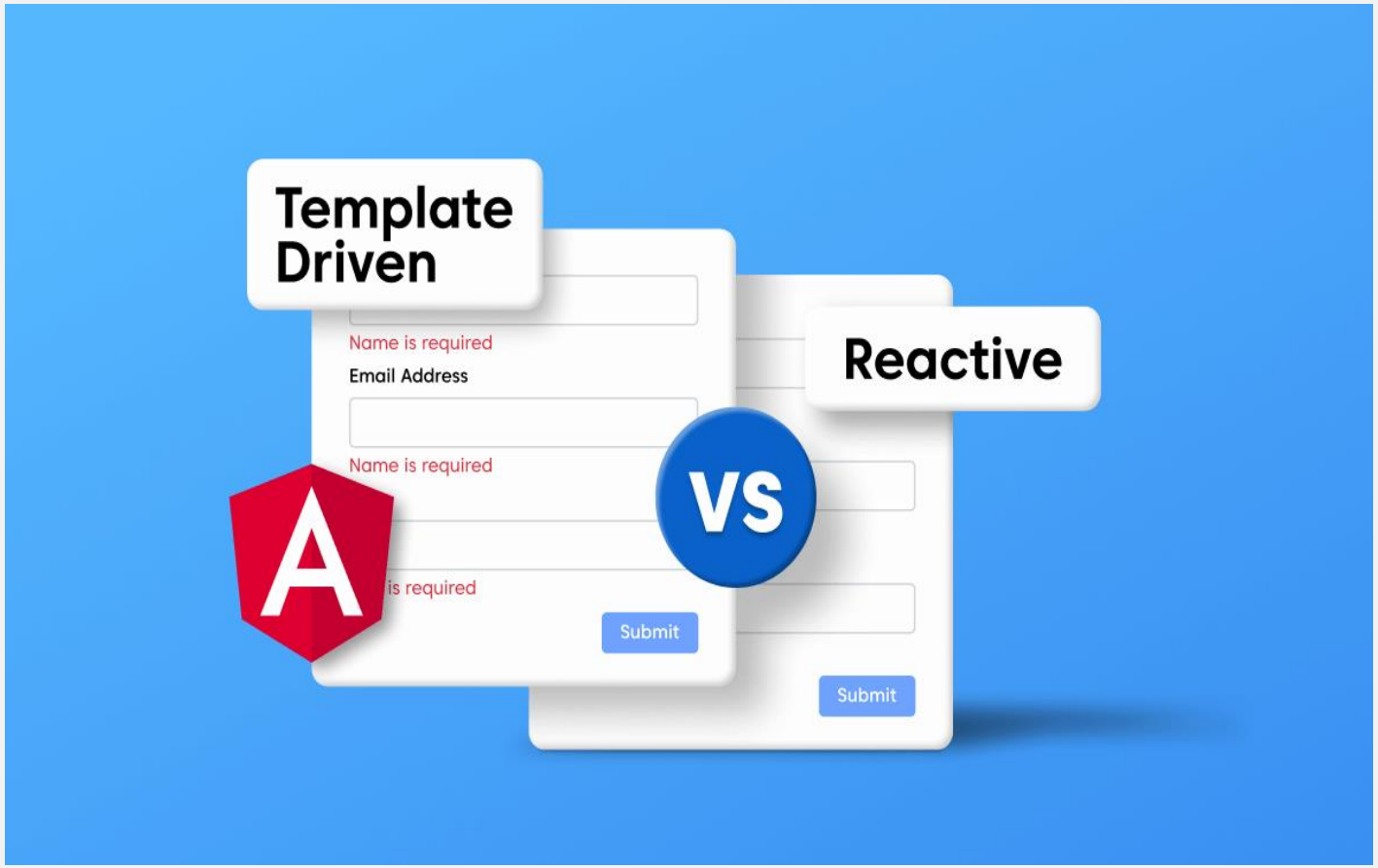
Forms in Angular are used to **capture and validate user input** from HTML elements.

Angular provides **two ways** to build forms:

1- Import **FormsModule**

2- #form=ngForm

3- Register Controls



1. Template-Driven Forms

🧠 What Are Template-Driven Forms?

> Template-driven forms are created and managed **mainly in the HTML template** using Angular directives.

✅ Angular tracks the form state **automatically** behind the scenes.

✅ Setup

ts Copy Edit

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

✅ Add to your standalone `imports` or module `imports`.

🔧 Key Directives

Directive	Purpose
`ngForm`	Tracks the form itself
`ngModel`	Two-way data binding for inputs
`name` attribute	Required to register input in form
`#form="ngForm"`	Template reference to access status

✅ 1. How to Register an Input

Use the `ngModel` directive:

html Copy Edit

```html
<input name="email"ngModel>
```

> `name` is **required** to register the input with Angular's form system.

You can either use it ngModel or two-way biding **[(ngModel)]**

1. Template-Driven Forms

✅ 2. How to Register a Form

Use a **template reference variable**:

html Copy Edit

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

> `form` now gives you access to properties like `form.valid`, `form.value`, etc.

✅ 3. How to Submit a Form

Use the `(ngSubmit)` event:

html Copy Edit

```html
<form #form="ngForm" (ngSubmit)="submit(form)">
  <button type="submit">Send</button>
</form>
```

> Call a method and pass the form to handle validation and submission.

1. Template-Driven Forms

✅ 4. Form Validation with Directives

Use **Angular-enhanced HTML validation attributes:**

html Copy Edit

```html
<input name="email"ngModel required email maxlength="50">
```

> Angular adds built-in validation for:

- `required`
- `minlength` / `maxlength`
- `pattern`
- `email`

1. Template-Driven Forms

✅ 4. Form Validation with Directives

Use **Angular-enhanced HTML validation attributes:**

html Copy Edit

```html
<input name="email"ngModel required email maxlength="50">
```

> Angular adds built-in validation for:

- `required`
- `minlength` / `maxlength`
- `pattern`
- `email`

1. Template-Driven Forms

✅ 5. Styling Form Inputs Based on State

Angular applies automatic CSS classes to inputs:

Class	Meaning
`ng-valid`	Control is valid
`ng-invalid`	Control is invalid
`ng-touched`	User has focused and blurred
`ng-untouched`	User hasn't interacted
`ng-pristine`	Control has not changed
`ng-dirty`	Control value has changed

```css
input.ng-invalid.ng-touched {
  border: 1px solid red;
}
```

> This makes it easy to highlight errors in UI without JS.

These Classes are added for both Types of Forms and inputs

1. Template-Driven Forms

```
<form #form="ngForm" (ngSubmit)="submit(form)">
    <label>Full Name</label>
    <input type="text" name="fullName" ngModel required minlength="5" />

    <label>Email</label>
    <input type="email" name="email" ngModel required email />

    <label>Age</label>
    <input type="number" name="age" ngModel required min="18" />

    <button type="submit" [disabled]="form.invalid">Submit</button>
</form>
```

```
@if (form.dirty&&form.invalid) {
<div class="error-message">
    <span style="font-weight: bold">Please fill out all fields correctly.</span>
    @if (form.control.get('fullName')?.invalid) { <span>Full Name Must be 5 letters</span> }
    @if (form.control.get('email')?.invalid) { <span>Email Must be valid</span> }
    @if (form.control.get('age')?.invalid) { <span>You age must be 18 or more</span> }
</div>
}
```

1. Template-Driven Forms

```
<form #signUpForm="ngForm" (ngSubmit)="onFormSubmit(signUpForm)">
    <label>Full Name</label>
    <input type="text" name="fullName" ngModel required minlength="5"/>
    <label>Email</label>
    <input type="text" name="email" ngModel required email/>
    <br>
    <button type="submit">Submit Form</button>
</form>
```

✅ Submitting the Form

ts Copy Edit

```ts
submit(form: NgForm) {
  if (form.valid) {
    console.log(form.value); // { name: '...', email: '...' }
  }
}
```

> Form data is passed as an object via `form.value`

🚨 Common Mistakes

Mistake	Why It Fails
Missing `name` attribute	Input won't register in `ngForm`
Using `[(ngModel)]` without import	Requires `FormsModule`
Expecting custom logic in TS only	Most logic is in the template

```
<form #signUpForm="ngForm" (ngSubmit)="onFormSubmit(signUpForm)">
  <label>Full Name</label>
  <input
    type="text"
    name="fullName"
    ngModel
    required
    minlength="5"
    #fullNameInput="ngModel"
  />
  <label>Email</label>
  <input
    type="text"
    name="email"
    ngModel
    required
    email
    #emailInput="ngModel"
  />
  <br />
  <button type="submit">Submit Form</button>
</form>

<!-- @if(signUpForm.dirty&&signUpForm.invalid){
<p class="input-error">
    Form Invalid
</p>
} -->

@if(fullNameInput.dirty&&fullNameInput.invalid){
<p class="input-error">Full Name must be 5 letters at least</p>
} @if(emailInput.dirty&&emailInput.invalid){
<p class="input-error">Must be Valid email</p>
}
```

2. Reactive Forms

🔹 What Are Reactive Forms?

> Reactive Forms are **programmatically built and managed in TypeScript** using Angular's reactive form APIs.

✅ Provide **complete control**, great for:

- Complex validation
- Dynamic forms
- Unit testing

✅ 1. Setup

Import the module:

ts Copy Edit

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

Then add to:

- `imports` array in `@Component()` for standalone components
- Or `NgModule` for traditional apps

2. Reactive Forms

✅ 2. Create a Form in TypeScript

ts

Copy Edit

```ts
form = new FormGroup({
  name: new FormControl(''),
  email: new FormControl(''),
});
```

2. Reactive Forms

How to link TS Form to the template?

✅ 3. Template Usage

html Copy Edit

```html
<form [formGroup]="form" (ngSubmit)="submit()">
  <input formControlName="name">
  <input formControlName="email">
  <button [disabled]="form.invalid">Submit</button>
</form>
```

- Use `[formGroup]` to bind the form
- Use `formControlName` to bind controls

2. Reactive Forms

✅ 4. Accessing Form Values & Status

ts Copy Edit

```ts
submit() {
  if (this.form.valid) {
    console.log(this.form.value); // { name: '...', email: '...' }
  }
}
```

✅ 5. Add Validators

ts Copy Edit

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

this.form = new FormGroup({
  name: new FormControl('', Validators.required),
  email: new FormControl('', [Validators.required, Validators.email])
});
```

2. Reactive Forms

✅ 6. Show Error Messages in Template

html

Copy Edit

```html
<input formControlName="email">
<div *ngIf="form.get('email')?.invalid && form.get('email')?.touched">
  <small *ngIf="form.get('email')?.errors?.['required']">Email is required</small>
  <small *ngIf="form.get('email')?.errors?.['email']">Invalid email format</small>
</div>
```

```ts
get fullName() {
  return this.userForm.get('fullName');
}
```

✅ 7. Dynamic Forms (Add Controls at Runtime)

ts

Copy Edit

```ts
this.form.addControl('phone', new FormControl(''));
```

Remove:

ts

Copy Edit

```ts
this.form.removeControl('phone');
```

2. Reactive Forms

📊 Template-Driven vs Reactive Forms

Feature	Template-Driven	Reactive
Where it's defined	HTML	TypeScript
Control	Limited	Full programmatic
Validation location	Template	TypeScript
Dynamic Forms	Difficult	Easy
Testability	Low	High
Best for	Simple forms	Complex forms

2. Reactive Forms

🧠 What is `valueChanges`?

> `valueChanges` is an **Observable** provided by Angular's `FormControl`, `FormGroup`, or `FormArray`.

It **emits the latest value** every time the form value changes.

✅ Use Cases

- 🔄 Live preview (e.g. update a preview while typing)
- 🔴 Enable/disable buttons or sections
- 💾 Auto-save as user types
- 🔎 Trigger validation or logic dynamically

💡 Real-World Example

ts Copy Edit

```ts
this.form.get('email')?.valueChanges
  .pipe(debounceTime(300))
  .subscribe(value => {
    this.checkEmailAvailability(value);
  });
```

> Debounced API call when email input stops typing

🧩 Basic Example

ts Copy Edit

```ts
this.form = this.fb.group({
  name: [''],
  email: ['']
});

this.form.valueChanges.subscribe(val => {
  console.log('Form value changed:', val);
});
```

✅ Logs:

css Copy Edit

```css
{ name: 'M', email: '' }
{ name: 'Ma', email: '' }
{ name: 'Mah', email: '' }
```

✅ Watch a Specific Control

ts Copy Edit

```ts
this.form.get('email')?.valueChanges.subscribe(value => {
  console.log('Email changed:', value);
});
```

2. Reactive Forms

🔹 `setValue()` – Exact Match Required

> Sets **all controls** in the form. Throws an error if you miss any.

✅ Example:

ts Copy Edit

```ts
this.form = this.fb.group({
  name: [''],
  email: ['']
});

this.form.setValue({
  name: 'Mahmoud',
  email: 'mahmoud@example.com'
});
```

⛔ This will throw an error if you omit even one control.

2. Reactive Forms

🔹 `patchValue()` – Partial Update

> Updates only the **specified controls**. Skips the rest silently.

✅ Example:

ts Copy Edit

```ts
this.form.patchValue({
  email: 'updated@example.com'
});
```

- ✅ `name` will remain unchanged
- ✅ No error even if you don't pass all fields

2. Reactive Forms

✅ 2. Create a Form in TypeScript

ts Copy Edit

```ts
form = new FormGroup({
  name: new FormControl(''),
  email: new FormControl(''),
});
```

OR using `FormBuilder` for less boilerplate:

ts Copy Edit

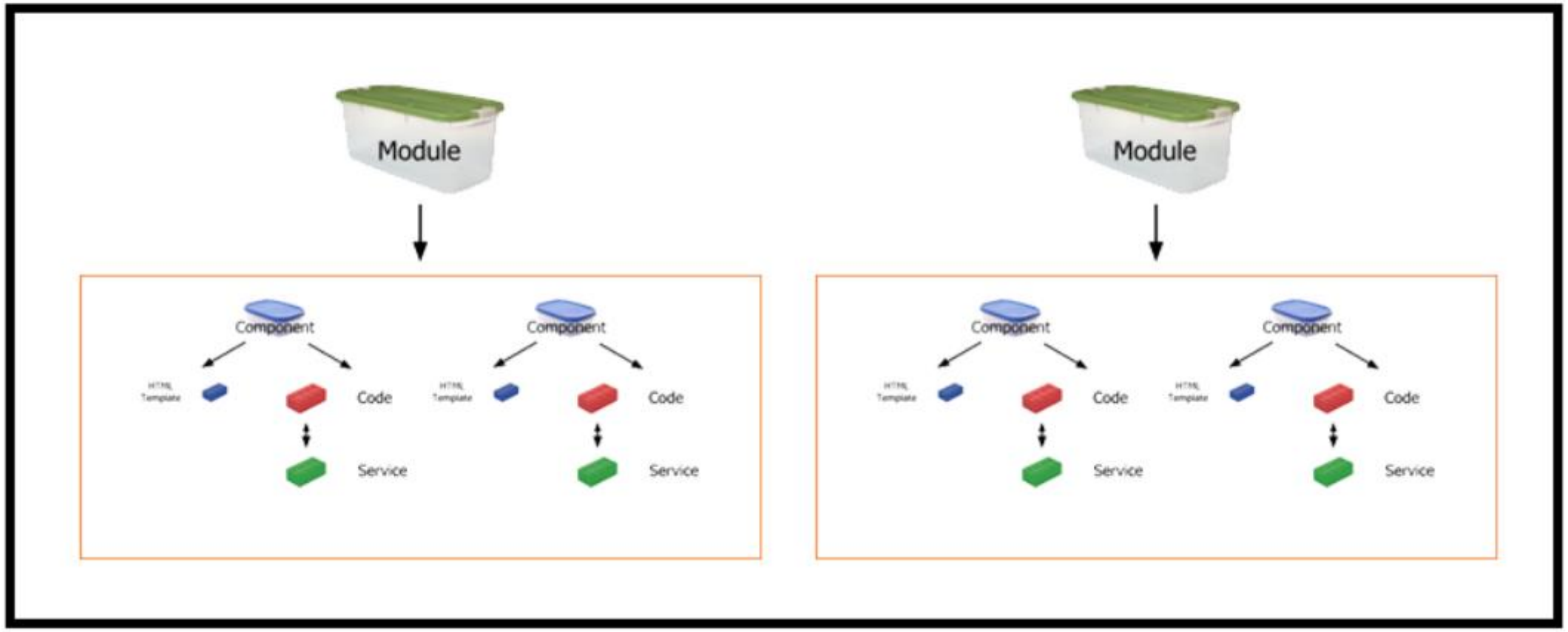
```ts
constructor(private fb: FormBuilder) {
  this.form = this.fb.group({
    name: [''],
    email: [''],
  });
}
```

ts Copy Edit

```ts
this.form = this.fb.group({
  name: ['', [Validators.required]],
  email: ['', [Validators.required, Validators.email]],
});
```

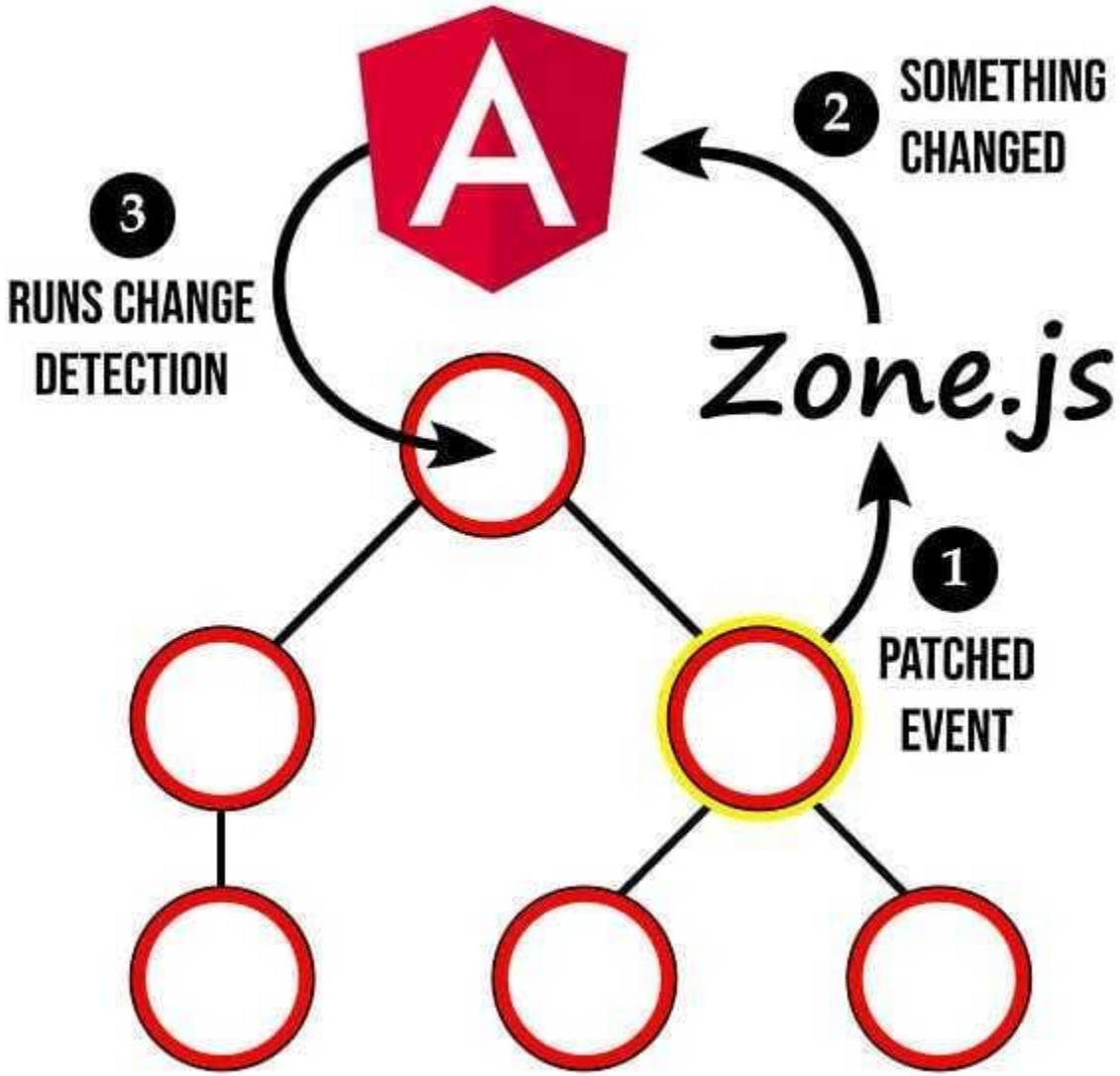
NgModules

App

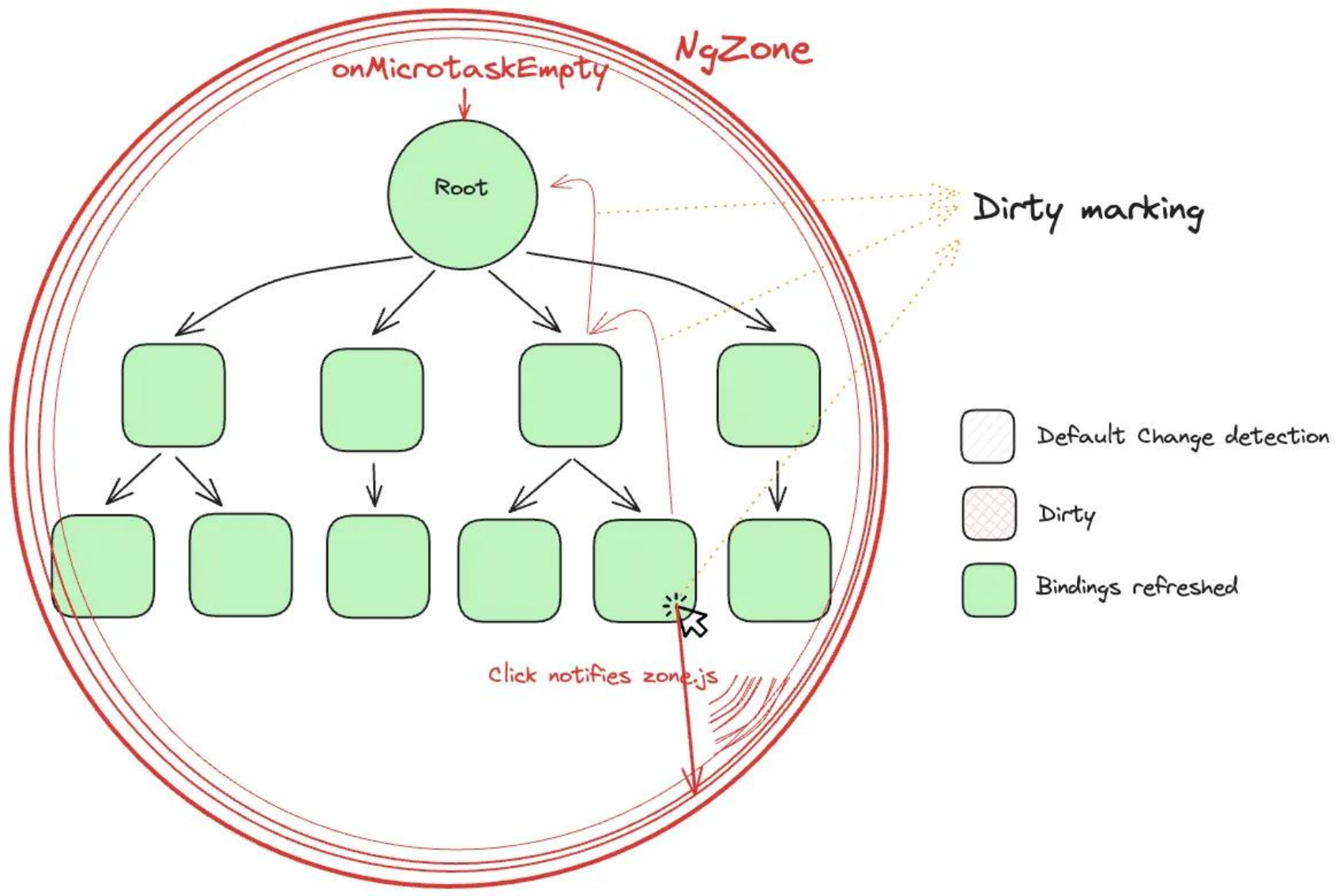

Module
Module
Component
Component
Component
Component
Code
Code
Code
Code
Service
Service
Service
Service

Change Detection

How Angular Updates the UI??



How Angular Updates the UI??



How Angular Updates the UI??

🔁 Change Detection in Angular

> Angular automatically checks your component and updates the view **when inputs, events, or service data change**.

🔧 This process is called **Change Detection**.

🧠 Key Idea:

> Every time **you click, type, or fetch data**, Angular **checks the component tree** to find what needs to update.

🤯 Problem?

In large apps, checking the **entire component tree** every time can be slow.

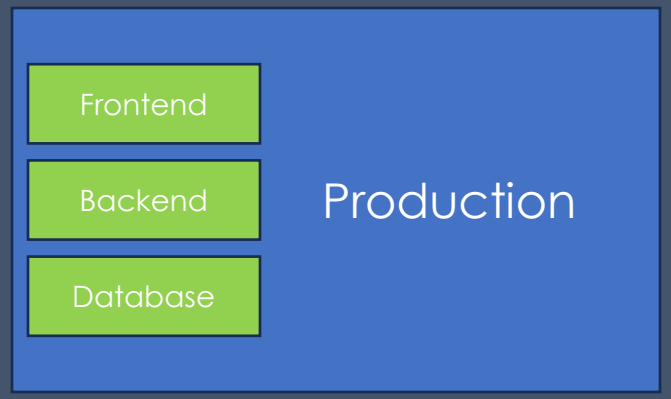
How Angular Updates the UI??

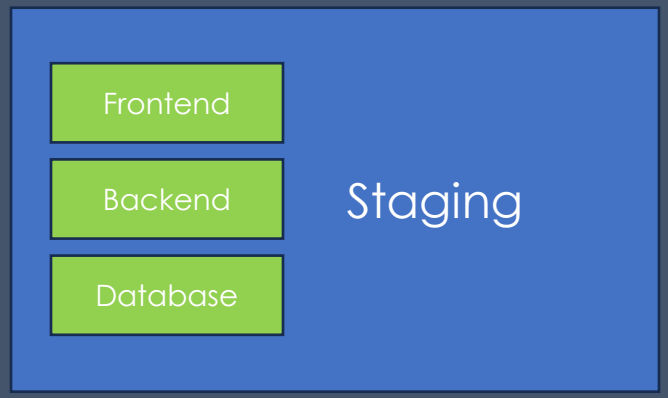
✅ Solutions:

Tool	What it does
`OnPush` Strategy	Only updates if input changes
Signals	Only updates when the signal's value changes
`ChangeDetectorRef`	(Advanced) Manual control

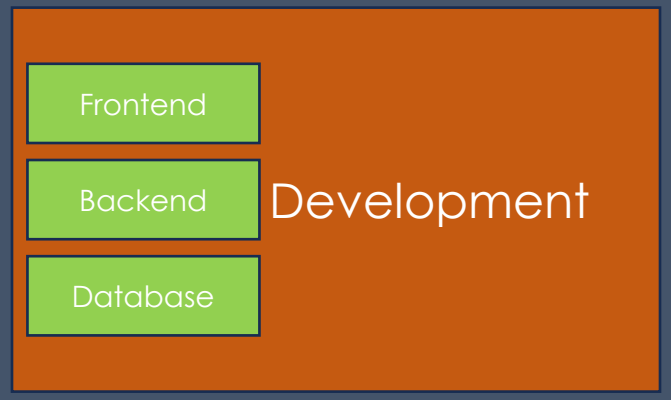
Angular relies on Zone.js to detect changes automatically.

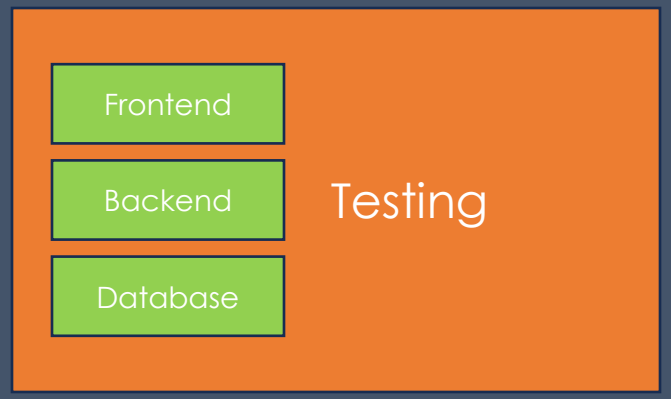
But Signals and OnPush are the way forward — especially in zone-less apps.





Environment





📘 Angular Environment Configuration

🧠 What Is It?

> Angular lets you define **different settings** for different environments — like development, staging, or production.

✅ Great for switching between local and live APIs
✅ No need to change code manually each time you build

📁 Where?

Located in the `src/environments/` folder:

Copy Edit

```
📁 environments
├ 📄 environment.ts        ← Development
└ 📄 environment.prod.ts   ← Production
```

🧩 Example: environment.ts

ts Copy Edit

```ts
export const environment = {
  production: false,
  apiUrl: 'http://localhost:3000'
};
```

🧩 Example: environment.prod.ts

ts Copy Edit

```ts
export const environment = {
  production: true,
  apiUrl: 'https://api.yoursite.com'
};
```

✅ How to Use in Code

ts Copy Edit

```ts
import { environment } from '../environments/environment';

this.http.get(`${environment.apiUrl}/users`);
```

✅ Automatically switches based on build mode.

🛠️ Build for Production

bash Copy Edit

```bash
ng build --configuration production
```

➡️ Uses `environment.prod.ts`

Project Build

Ng Build -- configuration production