GUIDE

Angular v22: Signals, Defaults, and AI Readiness

> ⚠️ **HIGH PRIORITY** — Important fixes. Upgrade soon.

> **Version:** 22.0.0 | **Released:** 2026-06-03 | **Upgrade from:** 21.x

---

Release at a Glance

Angular v22 lands with a powerful suite of improvements, cementing the framework's reactive future and modernizing core defaults. This major release is not just about new features; it's about refining the developer experience and setting the stage for future innovations, particularly in the realm of AI.

Here's a quick TL;DR for what you need to know:

- **Signal Forms API is now stable:** Embrace fine-grained reactivity for forms, offering improved performance and developer ergonomics.
- **OnPush is the default change detection:** New components now implicitly use `OnPush`, a significant shift for performance-conscious applications.
- **HTTP Client defaults to Fetch API:** Modernizes network requests under the hood, aligning with web standards.
- **AI Readiness:** Enhanced support for LLM prompts and AI IDE setup paves the way for intelligent developer tooling and application features.

Headline New Features

Angular v22 brings several pivotal changes that redefine how we build and perceive Angular applications. These aren't just incremental updates; they represent a strategic evolution towards a more performant, reactive, and future-proof framework.

Signal Forms API: Stable and Ready for Production

The highly anticipated Signal Forms API has officially graduated to stable! This marks a significant milestone in Angular's journey towards fine-grained reactivity. Signal Forms offer a more intuitive and performant way to manage form state, leveraging the power of signals to track changes efficiently.

Why this matters: Traditional reactive forms often involve subscriptions and manual change detection triggers. Signal Forms simplify this by making form controls, groups, and arrays inherently reactive, reducing boilerplate and improving performance by only updating what's changed.

Before (Traditional Reactive Forms)

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-old-profile',
  template: `
    <form [formGroup]="profileForm" (ngSubmit)="onSubmit()">
      <input formControlName="firstName" placeholder="First Name">
      <input formControlName="lastName" placeholder="Last Name">
      <button type="submit">Submit</button>
    </form>
    <p>Form Status: {{ profileForm.status }}</p>
  `,
})
export class OldProfileComponent implements OnInit {
  profileForm!: FormGroup;

  constructor(private fb: FormBuilder) {}

  ngOnInit() {
    this.profileForm = this.fb.group({
      firstName: ['', Validators.required],
      lastName: [''],
    });

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

  onSubmit() {
    console.log('Form submitted:', this.profileForm.value);
  }
}
```

After (Stable Signal Forms)

```
import { Component, signal } from '@angular/core';
import { FormBuilder, Validators, SignalFormGroup } from '@angular/forms'; //
Note: SignalFormGroup
```

```
@Component({
  selector: 'app-new-profile',
  template: `
    <form [formGroup]="profileForm" (ngSubmit)="onSubmit()">
      <input [formControl]="profileForm.controls.firstName" placeholder="First
Name">
      <input [formControl]="profileForm.controls.lastName" placeholder="Last
Name">
      <button type="submit">Submit</button>
    </form>
    <p>Form Status: {{ profileForm.status() }}</p> <!-- status is a signal -->
  `,
})
export class NewProfileComponent {
  profileForm: SignalFormGroup<{
    firstName: string;
    lastName: string;
  }>;

  constructor(private fb: FormBuilder) {
    this.profileForm = this.fb.signalFormGroup({
      firstName: this.fb.signalFormControl('', Validators.required),
      lastName: this.fb.signalFormControl(''),
    });

    // Accessing signal values directly
    this.profileForm.valueChanges().subscribe(value => {
      console.log('Form value changed:', value);
    });
  }

  onSubmit() {
    console.log('Form submitted:', this.profileForm.value()); // Access value
as a signal
  }
}
```

📌 `Key Idea:` Signal Forms reduce the mental overhead of managing form state and subscriptions, leading to cleaner, more efficient code.

OnPush is Now the Default Change Detection Strategy

In a move that underscores Angular's commitment to performance, new components created in v22 will now default to the `OnPush` change detection strategy. This is a significant shift from the previous default, which often led to unnecessary re-renders.

Why this matters: `OnPush` components only re-render when their input properties change (by reference), an observable they subscribe to emits, or an event handler is triggered within the component. This drastically reduces the number of change detection cycles, leading to faster applications.

HTTP Client Now Uses Fetch API by Default

The underlying implementation of Angular's `HttpClient` has been modernized to use the browser's native `Fetch API` by default, moving away from `XMLHttpRequest`.

Why this matters: The Fetch API offers a more modern, promise-based interface for making network requests, aligning `HttpClient` with contemporary web standards. While the public API for `HttpClient` remains largely the same, this change brings potential performance benefits and better integration with service workers and other modern browser features.

AI Readiness: LLM Prompts and IDE Setup

Angular v22 takes a proactive step towards integrating AI into the developer workflow and applications. This release includes enhanced support for working with Large Language Model (LLM) prompts and improved setup for AI-powered IDE features.

Why this matters: This lays the groundwork for developers to more easily build AI-driven features within their Angular applications and leverage AI tools for code generation, refactoring, and debugging directly within their development environment. Expect to see more intelligent tooling and application capabilities emerge from this foundation.

New APIs (Stable Signal Forms, Angular Aria, Asynchronous Signals)

Beyond the headline features, v22 brings several APIs to stable status, providing robust tools for building accessible and reactive applications.

Angular Aria: Built-in Accessibility

`Angular Aria` has graduated to stable, offering a collection of primitives and utilities designed to make Angular applications more accessible out-of-the-box. This includes components and directives that help manage ARIA attributes, focus management, and keyboard interactions.

Why this matters: Accessibility is crucial for inclusive web development. `Angular Aria` simplifies the process of building accessible user interfaces, reducing the burden on developers to manually manage complex ARIA specifications.

Asynchronous Signals: Handling Reactive Data Flows

`Asynchronous Signals` are now stable, providing a powerful pattern for managing reactive data flows that involve asynchronous operations (e.g., fetching data from an API). These signals integrate seamlessly with Angular's reactivity model, allowing for cleaner handling of loading states, errors, and data updates.

Why this matters: Managing asynchronous data in a reactive application can be complex. Asynchronous Signals provide a standardized, signal-based approach to handle these scenarios, making the code more readable and maintainable.

Performance Improvements

Angular v22 continues the framework's relentless focus on performance, delivering improvements that make applications feel faster and more responsive.

- **Improved Data Loading Primitives:** The framework now includes better data loading primitives, which contribute to a faster application feel and improved responsiveness, especially in data-intensive applications. This helps reduce perceived latency and enhances the user experience.
- **Reduced Framework Overhead:** Ongoing optimizations have further reduced the framework's runtime overhead, meaning smaller bundle sizes and less work for the browser, leading to quicker initial loads and smoother interactions.
- **Optimized Runtime Performance:** The continuous effort to optimize Angular's rendering engine and change detection mechanisms (especially with the `OnPush` default) translates directly into faster runtime performance for complex applications.

`⚡ Real-world insight:` These performance gains, combined with the `OnPush` default, mean that well-architected Angular applications in v22 can achieve significantly better performance metrics, rivaling or exceeding other popular frameworks.

Breaking Changes and Migration Impact

As a major release, Angular v22 introduces a few breaking changes. While the Angular team strives for smooth upgrades, these changes require attention during migration.

OnPush is Now the Default Change Detection Strategy

This is the most significant behavioral change in v22. New components will automatically use `OnPush`. Existing components, however, will retain their current `Default` strategy unless explicitly updated.

Why this is a breaking change: If you rely on the previous `Default` change detection behavior (where components always checked for changes regardless of input reference equality), new components you create might not update as expected. Existing components will still work, but if you migrate them without understanding `OnPush`, you might encounter issues.

Before (Implicit Default Change Detection)

```
// app.component.ts (Angular < v22)
// No changeDetection specified, defaults to ChangeDetectionStrategy.Default
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <h1>Hello, {{ name }}!</h1>
    <button (click)="updateName()">Update Name</button>
  `,
})
export class AppComponent {
  name: string = 'World';

  updateName() {
    // This would trigger change detection and update the view
    // even if 'name' was an input property from a parent
    this.name = 'Angular Developer';
  }
}
```

After (Explicit OnPush Recommended for Migrated Components)

When migrating existing components or creating new ones where `OnPush` behavior is desired (which is now the default for newly generated components), you should be explicit.

```
// app.component.ts (Angular v22+)
import { Component, ChangeDetectionStrategy } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <h1>Hello, {{ name }}!</h1>
    <button (click)="updateName()">Update Name</button>
  `,
  changeDetection: ChangeDetectionStrategy.OnPush, // Explicitly set for
clarity
})
```

```
export class AppComponent {
  name: string = 'World';

  updateName() {
    // With OnPush, simply changing 'name' might not update the view
    // unless it's an input property whose reference changes,
    // or an event handler inside this component triggers it.
    // For local state, you might need to use signals or markForCheck().
    this.name = 'Angular Developer';
    // If 'name' was an @Input, changing its reference would trigger update:
    // this.name = { ...this.name, value: 'Angular Developer' };
  }
}
```

⚠️ `What can go wrong:` Components relying on mutations of input objects or arrays might not update their views if `OnPush` is active and the reference of the input itself hasn't changed. You might need to explicitly call `ChangeDetectorRef.markForCheck()` or, preferably, ensure immutability or use signals.

HTTP Client Now Uses Fetch API by Default

The switch from `XMLHttpRequest` to `Fetch API` is largely an internal implementation detail, but it can introduce subtle behavioral differences.

Why this is a breaking change: While your `HttpClient` calls will generally work the same, there might be differences in how certain network errors are reported, how `AbortController` integrates, or how some third-party libraries that directly interact with `XMLHttpRequest` might behave. Interceptors that rely on `XMLHttpRequest` specifics might also need review.

Example (Conceptual Impact, Code Remains Similar)

The actual usage of `HttpClient` remains the same:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

@Injectable({ providedIn: 'root' })
export class DataService {
  constructor(private http: HttpClient) {}

  getData() {
    // This code remains the same
    return this.http.get('/api/data');
  }
}
```

🧠 `Important:` While the API surface is stable, developers should be aware of potential nuances, especially around error handling and custom interceptors. Thorough testing of network-related features is advised after upgrading.

Why This Release Matters for Production Teams

Angular v22 is a critical upgrade for all production Angular teams, offering tangible benefits and setting the stage for future innovation.

- **Modernized Defaults for Better Performance:** The shift to `OnPush` as the default and `Fetch API` for HTTP requests signals a clear direction towards performance and modern web standards. Adopting these defaults means your applications are built on a more efficient foundation.
- **Stable Signal Forms:** This is a game-changer for form management. Production applications with complex forms will benefit from cleaner code, improved reactivity, and potentially better performance. It simplifies state management and reduces common pitfalls associated with traditional reactive forms.
- **AI Readiness:** As AI becomes increasingly integrated into software development, Angular v22 provides the necessary hooks and tooling support. This means your applications and development workflows are prepared to leverage future AI capabilities, keeping your team at the forefront of technology.
- **Enhanced Accessibility:** The stable `Angular Aria` APIs empower teams to build more inclusive applications with less effort, reducing the risk of accessibility compliance issues.
- **Future-Proofing:** By embracing signals and modern web APIs, Angular v22 positions your applications for long-term maintainability and easier adoption of future framework advancements.

How to Upgrade

Upgrading to Angular v22 is straightforward using the Angular CLI.

1. **Ensure you are on a supported version:** Angular v22 supports upgrades from v21.x. If you are on an older version, upgrade incrementally (e.g., v17 -> v18 -> ... -> v21 -> v22).
2. **Update your Angular CLI globally:**

```
npm install -g @angular/cli@latest
```

```
or
```

```
    yarn global add @angular/cli@latest
```

1. **Navigate to your project directory and run the update command:**

```
    ng update @angular/cli @angular/core
```

```
This command will update your core Angular packages and the CLI, and it will
also run any migration schematics to help with breaking changes.
```

1. **Review the migration guide:** While `ng update` handles many changes automatically, always consult the official Angular blog post for v22 and the `ng update` output for any manual steps or specific considerations for your application. The official changelog can be found at: https://blog.angular.dev/announcing-angular-v22-c52bb83a4664

`🔥 Optimization / Pro tip:` Run `ng update` with the `--create-commits` flag (`ng update --create-commits`) to generate a commit after each migration schematic runs. This allows you to easily review and revert specific changes if needed.

Ecosystem Impact and Tooling Updates

The Angular ecosystem is robust and generally quick to adapt to new releases. For Angular v22:

- **Angular CLI:** The Angular CLI is fully updated to support v22, including new commands, schematics for migrations, and improved build processes. Ensure your local and project CLI versions are aligned.
- **IDE Extensions:** Expect popular IDE extensions (e.g., Angular Language Service for VS Code) to release compatible updates shortly after the framework's release, enhancing the developer experience with updated linting, autocompletion, and debugging capabilities for new APIs like Signal Forms.

- **Third-Party Libraries:** Most widely used third-party libraries and component frameworks (e.g., Angular Material, NgRx, PrimeNG) typically release compatible versions within weeks of a major Angular release. It's always advisable to check their official documentation or GitHub repositories for v22 compatibility before upgrading your entire application.

The strong ecosystem support ensures that developers can leverage the new features of v22 without significant disruption to their existing toolchains and dependencies.