WHITE PAPER · NOVEMBER 2026

Angular end of life: an upgrade guide for old applications

Why you climb one major version at a time, the stage plan, and what your team should have when the work is done.

Published by TAK Devs
Published November 11, 2026 · Last checked November 10, 2026

Contents

/01 What does Angular end of life cost a business?
/02 How long does Angular support each major version?
/03 Why upgrade one major Angular version at a time?
/04 What does a staged Angular upgrade plan look like?
/05 What should your team have when the upgrade is done?
/06 Has a team moved seven Angular versions in one engagement?
/07 In what order should you test modules during an upgrade?
/08 What should you check before you touch a dependency?
/09 Frequently asked questions
/10 How this paper was checked
/11 References

/00

Executive summary

THE SHORT ANSWER

Angular 20 leaves long-term support on November 28, 2026, according to Angular's official release schedule , and versions 2 to 19 are already unsupported. If your app is older, update one major version at a time, as Angular advises. Fix the toolchain at each step, test every module, and hand over a setup your team can run.

An old Angular app rarely breaks on the day support ends. It gets more expensive to change, one quiet month at a time, until every feature request starts with a risk meeting. This guide is for the CTO who owns that app and the CEO who will fund the fix. It explains what end of life means in practice, why the safe path climbs one major version at a time, and what a staged plan and a clean handover look like, with proof from a seven-version upgrade TAK Devs delivered.

Three findings

01 Angular now supports a major version for about 24 months, but versions released on the old six-month cycle got roughly 18. Angular 20 support ends November 28, 2026.

02 Angular's documentation tells teams crossing several majors to update one major version at a time. Each hop also moves the Node.js and TypeScript floor.

03 An upgrade is finished when the team can run, test and extend the app, not when the version number changes. TrimData moved seven majors with all 8 modules working.

/01

What does Angular end of life cost a business?

End of life means the Angular team stops shipping fixes for that version. Your app keeps running. What stops is the flow of patches for problems found after that date, and the framework's own guide is clear about why that matters: Angular's security best practices[2] say the libraries get regular updates that might fix security defects found in earlier versions.

Those defects are real and recent. The US National Vulnerability Database records CVE-2025-66412, a stored cross-site scripting flaw in the Angular template compiler[3], received in December 2025. Fixes shipped for versions 19, 20 and 21. Every version older than 19 is listed as affected, and none of them got a fix, because none of them was still supported.

An unsupported framework turns every new vulnerability into a permanent one. The OWASP Top 10 entry on vulnerable and outdated components (2021)[4] names software that is unsupported or out of date, and a failure to upgrade frameworks and dependencies in a timely fashion, as direct signs an application is exposed. Security reviewers read from the same list.

The runtime ages too. Angular 11, for example, was built for Node.js 10 or 12 according to Angular's version compatibility table[5]. The Node.js release schedule[6] lists both as end of life, and states that production applications should only use Active LTS or Maintenance LTS releases. So an old Angular app often means an old server runtime as well.

For the business, the cost shows up in three places. Security questionnaires get harder to answer. New engineers lose their first weeks fighting a toolchain nobody can install cleanly. Feature work slows, because every change risks breaking something nobody has tested in years. None of this appears on an invoice, which is exactly why it gets ignored until a customer or an auditor asks.

The longer the wait, the worse the arithmetic. Every six-month gap under the old cycle added another major version to climb, and each hop has its own toolchain and library changes. A team two versions behind faces a short project. A team seven versions behind faces a programme, and it usually discovers that only when a customer contract or a security review forces the question.

/02

How long does Angular support each major version?

Angular publishes its support policy on angular.dev's versioning and releases page[1]. Under the current policy, a major release is typically supported for 24 months, split into two stages.

- Active support, 12 months: regularly scheduled updates and patches.
- Long-term support (LTS), 12 months: only critical fixes and security patches.
- After LTS: no fixes of any kind from the Angular team.

Read the stages carefully, because they decide what you can expect from the Angular team. In active support you get regular releases with fixes and improvements. In LTS you get only what is critical or security-related, so known bugs that are neither will stay in place. Once LTS ends, nothing more ships for that version at all, however serious the issue.

That 24-month window is new. The same page explains that until Angular 22, released on June 3, 2026, Angular shipped a new major version every six months. Older majors therefore had a shorter run. Angular 20, released on May 28, 2025, left active support on November 19, 2025 and leaves LTS on November 28, 2026: about 18 months in total.

> **THE DATE TO PLAN AROUND**
>
> Angular 20 long-term support ends on November 28, 2026, per the release table on angular.dev. Some third-party sites quote November 27. Use the official page, and recheck it before you commit a plan, because Angular describes its dates as general guidance that can change.

The page also lists what is already gone: Angular versions 2 to 19 are no longer supported. If your app sits anywhere in that range, it is past end of life today. Rather than copy a list of supported versions into a plan that will outlive it, link the live table on angular.dev and check it each quarter.

Two numbers decide your exposure: how many majors you are behind, and how many months your current version has left. The first sets the size of the job. The second sets the deadline.

The policy also tells you when to start. Planning an upgrade in the last month of LTS leaves no room for a library that turns out to be abandoned, or a test pass that finds a broken report. Teams that start while their version still has six months or more of LTS left can run the hops at a normal pace, release after each one and keep feature work moving on the side. Teams that start after end of life are already running without patches while they work.

/03

Why upgrade one major Angular version at a time?

Because the framework's own documentation says so. The Angular versioning and releases page[1] states that if you update across multiple major versions, you should perform each update one major version at a time. The Angular Update Guide[7] is built around the same idea: you pick a from-version and a to-version and get the steps for that hop.

There is a practical reason behind the rule. Each major version moves the floor under your code: the TypeScript version, the Node.js version and the build tooling. The gap between an old app and a current one is wide.

TOOLCHAIN FLOOR, ANGULAR 11.0 AGAINST THE NEWEST RELEASE LISTED ON ANGULAR.DEV (ANGULAR VERSION COMPATIBILITY TABLE AND NODE.JS RELEASE SCHEDULE, CHECKED NOV 10, 2026)

Requirement	Angular 11.0	Newest listed release
TypeScript	About 4.0	6.0
Node.js	10.13 or 12.11 and later	22.22.3, 24.15.0 or 26.0.0 and later
Node.js support status	Both lines end of life	Supported release lines

Jump all of that in one move and a broken build tells you almost nothing. Was it the framework, the compiler, the runtime or a library? A single hop changes fewer things at once, so a failure points at its cause. You fix it, test, and move on.

One version at a time feels slower, and it finishes sooner. The work you avoid is the week spent untangling five overlapping breakages that a staged path would have shown you one by one.

/04

What does a staged Angular upgrade plan look like?

A stage plan treats every major version as its own small release, with its own test pass. The shape below works whether you are two majors behind or seven.

1. **Assess before you change anything**
   List every module, every third-party library and the exact Node.js and TypeScript versions in use. Note which libraries are abandoned and will need a replacement. This list is the budget conversation with the CEO.
2. **Map the path**
   Write down every hop from the current version to the target, one major at a time. Mark the hops where Node.js or TypeScript must also move, because those take longer.
3. **Fix the toolchain first at each hop**
   Move Node.js, TypeScript, the CLI and the build configuration before touching application code. A clean build on the new version is the entry ticket to the next step.
4. **Fix dependency breakage**
   Update or replace each library that has no compatible release for the new major. Record every replacement, since it is a change your team will need to know about.
5. **Test every module on the new build**
   Run the full set of business flows, not only the screens that look broken. Caps, vouchers, reports and calculations fail quietly and are found at month end.
6. **Repeat, then freeze features until the last hop**
   New features wait, or live on a separate branch, until the final hop passes. Mixing feature work into an upgrade hides the cause of every failure.

The Angular CLI update command[8] helps inside each hop. It updates the core framework and CLI and can run the related migrations, which is one more reason to move a single major at a time: each run is scoped to one version change.

Two habits keep the plan honest. Keep the app releasable at the end of every hop, so the business is never stuck on a half-finished upgrade. And agree up front which business flows count as the test set, so done means the same thing to engineering and to the people who use the app.

/05

What should your team have when the upgrade is done?

An upgrade that only changes version numbers leaves you dependent on whoever did it. The handover is the part that keeps the investment working after the vendor leaves the room.

- A local environment any engineer can start on day one, with the Node.js version pinned. nvm, the Node Version Manager[9], manages several Node.js versions on one machine and reads a project's .nvmrc file for exactly this.
- A short record of what changed at each hop, including every library that was replaced and why.
- Evidence that every module was tested on the final build, mapped to the agreed business flows.
- A written specification for the next phase, before anyone starts adding features.
- A date in the calendar to recheck the support table on angular.dev.

The specification matters more than it looks. Most upgrades are not the end goal. They are the step that makes the next feature, integration or redesign safe to build. Writing that next phase down while the codebase is fresh in everyone's head saves weeks of rediscovery later.

/06

Has a team moved seven Angular versions in one engagement?

Yes. TrimData[10] builds Falink Cloud, a multi-tenant platform for managing stores, customers and point-of-sale systems. Its admin app ran on Angular 4, with dependencies several years out of date. New features risked breaking spending caps, vouchers and month-end reports, and the old toolchain made the app hard to run locally.

4.2.5 to 11.0.3
Angular versions, seven majors in one engagement

8 of 8
Modules working on the new build

7 tabs
Tenant setup wizard carried over intact

The platform was not a toy. It handles per-customer spending caps, bans and vouchers, live network and throughput metering, usage reporting, list imports, and a setup wizard that connects each tenant's database, proxy and remote servers. Any of those breaking in production would land straight on TrimData's own customers, which is why the upgrade could not be treated as a version bump.

TAK Devs followed the staged path. The team assessed all 8 modules and planned the upgrade across seven major versions, then fixed the dependency and tooling breakage and tested every module on the new build. Caps, bans and vouchers work on the new version, and the live throughput and network meters run again.

The handover matched the list above. TAK Devs set up the local environment with nvm and the correct Node.js version, so the team can run the app locally again, and wrote a specification for the next phase. TrimData booked more work the same day.

> Startlingly fast, efficient and professional.
>
> TrimData, on the Falink Cloud upgrade

One plain caveat. Angular 11 is itself out of support today. The engagement was never meant as the finish line: it moved Falink Cloud from an app nobody could safely change to a stable, tested base, with a written plan for the next phase. That base is what makes a move to a currently supported version practical. Read the full TrimData case study.

/07

In what order should you test modules during an upgrade?

DELIVERY LEAD INPUT NEEDED

[ADD DELIVERY LEAD INSIGHT] The TAK Devs delivery lead's account of the order in which the 8 Falink Cloud modules were tested, and the reason for that order, goes here (100 to 250 words). Do not publish this page or the PDF until this paragraph is replaced with that real input.

/08

What should you check before you touch a dependency?

Run these seven checks before anyone opens a pull request. Each one is short, and each one removes a reason an upgrade stalls halfway through.

ANGULAR UPGRADE CHECKLIST

- [ ] **Find your Angular version and the supported versions on angular.dev.**
  Run ng version in the project, then compare it with the live support table. The gap tells you how many hops you face. Write the number of hops at the top of the plan, so everyone sees the size of the job.

- **Inventory every module and dependency.**
  List modules, third-party libraries and their last release dates. Abandoned libraries are the slowest part of most upgrades. Note the Node.js and TypeScript versions too, since they set which tools will even install.

- **Plan the path one major version at a time.**
  Write each hop down, and mark the ones where Node.js or TypeScript must also move. Agree the target with the people who fund the work, and check it against the live support table, not a copied list.

- **Fix toolchain and dependency breakage at each step.**
  Get a clean build on the new major before touching features. Record every library you replace. A clean build on each hop is the gate to the next one.

- **Test every module after each step.**
  Use an agreed list of business flows, including the quiet ones such as reports and calculations, not only the screens that look broken. Write the list once and reuse it at every hop, so results stay comparable.

- **Keep a working local setup, with the Node version managed through nvm.**
  Pin the version in a .nvmrc file so a new engineer can run the app on day one. The same file keeps build servers and laptops on the same runtime.

- **Write the spec for the next phase before adding features.**
  The upgrade is the base. The specification turns it into the next release instead of the next rediscovery project. Include the date to recheck the support table, so the next upgrade does not arrive as a surprise.

/09

Frequently asked questions

When does Angular 20 reach end of life?

Angular 20 leaves long-term support on November 28, 2026, according to the release table on angular.dev. Some third-party sites quote November 27, so treat the official page as the reference. After that date Angular 20 gets no further critical fixes or security patches from the Angular team.

Can I skip Angular versions when I upgrade?

Angular's documentation says to perform each update one major version at a time when you cross several majors. Skipping mixes framework, TypeScript and Node.js changes into one move, so a broken build is much harder to trace. The Angular Update Guide gives the steps for each single hop.

How long does Angular support each major version?

Under the current policy a major release is typically supported for 24 months: 12 months of active support, then 12 months of long-term support with only critical and security fixes. Versions released on the earlier six-month cycle, such as Angular 20, got about 18 months in total.

Is an Angular app unsafe the day support ends?

Not overnight. The app keeps working. The risk is that vulnerabilities found after that date stay unpatched for your version, and OWASP lists unsupported software as a clear sign of exposure. Every month you wait adds versions to climb and cost to each new feature.

Is Angular 11 a safe place to stop?

No. Angular 11 is out of support, along with every version up to 19. Reaching it from Angular 4 is useful because it gives a stable, tested base and a working toolchain. The next step is to keep climbing to a version that angular.dev lists as supported.

Run the checklist before you touch a dependency

Every Angular version you skip makes the next feature riskier to ship. Request the TAK Devs Angular upgrade checklist and see how many hops stand between your app and a supported version.

Request the Angular upgrade checklist

Read the full TrimData case study

TAK Devs is a custom software and AI development company headquartered in Islamabad with a US presence, serving clients in the US and Europe. ISO 9001 and ISO 27001 certified. Developing Futures.

/10

How this paper was checked

Every source was opened and read on November 10, 2026, and each figure carries the date of its source. Primary sources were preferred: Angular's own documentation, the US National Vulnerability Database, the Node.js release schedule and OWASP. Angular support dates come from angular.dev, which describes them as general guidance that can change, so recheck them before you plan. The TrimData figures come from the live TAK Devs case study page. No competitor content was cited.

/11

References

[1] **Angular team, Google**. Versioning and releases. 2026. https://angular.dev/reference/releases

[2] **Angular team, Google**. Security best practices. 2026. https://angular.dev/best-practices/security

[3] **NIST**. CVE-2025-66412 detail. Dec 1, 2025. https://nvd.nist.gov/vuln/detail/CVE-2025-66412

[4] **OWASP Foundation**. A06:2021 Vulnerable and Outdated Components. 2021. https://top10.owasp.org/2021/A06_2021-Vulnerable_and_Outdated_Components/index.html

[5] **Angular team, Google**. Version compatibility. 2026. https://angular.dev/reference/versions

[6] **OpenJS Foundation**. Node.js Releases. 2026. https://nodejs.org/en/about/previous-releases

[7] **Angular team, Google**. Angular Update Guide. 2026. https://angular.dev/update-guide

[8] **Angular team, Google**. ng update command reference. 2026. https://angular.dev/cli/update

[9] **nvm-sh project, GitHub**. Node Version Manager README. 2026. https://github.com/nvm-sh/nvm

[10] **TAK Devs**. TrimData modernizes Falink Cloud from Angular 4 to Angular 11. Oct 13, 2026. https://takdevs.com/case-studies/trimdata-modernizes-falink-cloud-from-angular-4-to-angular-11/