TECHNICAL REFERENCE

Angular Migration Workflow — How It Works

A repeatable AI-driven process for upgrading Angular applications. The AI handles discovery, execution, blocker resolution, and verification. Human sign-off is required at two explicit gates.

Version 1.0 · February 2026 · Trigger: `/angular-migration` in AI chat

Lifecycle: Phase 1: Plan → Gate 1: Approve → Phase 2: Pre-flight → Phase 3: Execute → Phase 4: Verify → Gate 2: Recommend

---

PHASE 1 — DISCOVERY & PLANNING

1.1 — Gather Context

AI reads the following before writing any plan:

- `package.json` — current Angular and all dependency versions
- `angular.json` — workspace config, builders, schematics
- `tsconfig.json` / `tsconfig.base.json` — TypeScript version and strict flags
- `.eslintrc` / `eslint.config.*` — lint configuration
- `app.module.ts` or `app.config.ts` — detect NgModule vs standalone architecture

1.2 — Research the Migration Path

For each version jump (e.g. 16 → 17 → 18), AI reads:

- Angular Update Guide: `https://update.angular.io/?l=3&v={FROM}-{TO}`
- Angular CHANGELOG: `https://github.com/angular/angular/blob/main/CHANGELOG.md`

Identifies: Node.js version range, TypeScript requirements, removed/deprecated APIs, available schematics, third-party library compatibility (Material, NgRx, RxJS).

1.3 — Codebase Audit

```
npx ng version

# Scan for deprecated API usages
grep -rn "ComponentFactoryResolver\|Renderer\b\|ModuleWithProviders\|HttpClientModule\|Browse

# Count standalone vs module-based components
grep -rn "standalone: true" src/ --include="*.ts" | wc -l
grep -rn "NgModule" src/ --include="*.ts" | wc -l
```

1.4 — Build the Migration Plan

AI writes `implementation_plan.md` with these required sections:

SECTION	CONTENTS
Current State	Versions table: Angular, Node, TypeScript, RxJS, Material, etc.
Target State	New versions for every affected package
Breaking Changes	Only changes relevant to this codebase
Migration Steps	Ordered list of exact commands and file changes
Risk Assessment	High / Medium / Low risk items with mitigations
Estimated Effort	Time estimate per step
Rollback Plan	How to revert via git stash / commit

> **Gate 1 — Human Approval Required.** AI stops and delivers `implementation_plan.md` for review. Nothing is changed until the user explicitly approves. Plan changes can be requested before approval.

PHASE 2 — PRE-EXECUTION SETUP

2.1 — Create a Safety Checkpoint

```
git status
git stash push -m "pre-migration-stash-$(date +%Y%m%d_%H%M%S)"
```

If git is not initialised, this step is skipped and noted in the task log.

2.2 — Verify Node.js Compatibility

```
node --version
```

If the current Node.js version does not satisfy the target Angular version's engine requirement, AI halts and notifies the user. No migration proceeds on an incompatible Node version.

PHASE 3 — EXECUTION

AI executes steps in the exact order from the approved plan. Each step is tracked in `task.md` as `[/]` (in-progress) → `[x]` (done). On failure, the Blocker Resolution Protocol below activates.

Core Migration Commands (in order)

```
# 3.1 — Angular core
npx ng update @angular/core@{TARGET} @angular/cli@{TARGET} --allow-dirty --force

# 3.2 — Angular Material (if used)
npx ng update @angular/material@{TARGET} --allow-dirty --force

# 3.3 — Angular CDK (if used)
npx ng update @angular/cdk@{TARGET} --allow-dirty --force

# 3.4 — RxJS
npm install rxjs@{TARGET_VERSION}

# 3.5 — TypeScript
npm install typescript@{TARGET_VERSION} --save-dev

# 3.6 — Third-party packages (per dependency)
npm install {PACKAGE}@latest

# 3.7 — Automatic schematics
npx ng generate @angular/core:standalone        # NgModule → standalone
npx ng generate @angular/material:mdc-migration  # Material MDC migration

# 3.8 — Update tsconfig.json (strict flags, target, lib) manually as needed
```

Blocker Resolution Protocol

LEVEL	TRIGGER	ACTION
1 — Auto-resolve	Step fails on first try	Retry with `--legacy-peer-deps` / `--force`; fix TS errors per migration guide; replace deprecated APIs; check for renamed modules
2 — Research & Fix	Level 1 fails	Search Angular GitHub issues for the exact error; re-read version migration guide; apply the documented fix
3 — Escalate	Level 1 + 2 fail	Stop at the blocked step. Write `REMAINING_TASKS.md` (completed steps, exact error, remaining steps, suggested fixes). Run Phase 4 on partial state. Notify user.

PHASE 4 — AUTOMATED VERIFICATION SUITE

All checks run automatically after execution (full or partial). Results are logged in a pass/fail summary.

Error Threshold Policy

For every failing check, AI follows this escalation table before deciding whether to fix or hand off to the user:

ERROR COUNT	ACTION
0 errors	Pass — move to next check
1–9 errors	AI auto-fixes each one, re-runs the check to confirm, then continues
10–50 errors	AI fixes recognisable patterns (wrong imports, renamed APIs, type mismatches). Re-runs once. Reports remaining errors in the summary and continues to the next check.
> 50 errors	**AI stops fixing immediately.** Records the full error log in `REMAINING_TASKS.md`. Continues running the remaining checks to gather a full picture, but makes no further fix attempts. Notifies user — blocked on human.

Why stop at 50? Above that threshold errors are almost always systemic — an incompatible library, a wrong version pinned, or an architectural decision that needs a human call. Blindly patching 200+ errors risks making things worse and can run indefinitely.

Checks

CHECK	COMMAND	PASS CONDITION	ON FAILURE
TypeScript types	`npx tsc --noEmit`	Exit code 0	Apply threshold policy above
Dev build	`npx ng build --configuration=development`	Bundle complete, 0 errors	Apply threshold policy above
Production build	`npx ng build --configuration=production`	Production bundle generated	If dev build had >50 errors, skip and mark "skipped due to upstream failure"
Lint	`npx ng lint`	0 errors (warnings OK)	Auto-fix with `--fix` first, then apply threshold policy to remainder
Dev server runtime	`npx ng serve --port 4299` (15s check)	Server stays running	If build failed, mark "skipped — build must pass first"
Bundle size	`ls -lh dist/**/main*.js`	No significant regression	Flag if >20% increase vs pre-migration; report size regardless

Verification Summary (output format)

CHECK	STATUS	ERRORS FOUND	ACTION TAKEN
TypeScript types	✓ / ✗	N	Fixed / Escalated / Skipped
Dev build	✓ / ✗	N	Fixed / Escalated / Skipped
Prod build	✓ / ✗	N	Fixed / Escalated / Skipped
Lint	✓ / ✗	N	Fixed / Escalated / Skipped
Dev server	✓ / ✗	—	Running / Crashed / Skipped
Bundle size	info	—	X MB (±Y% vs baseline)

PHASE 5 — RECOMMENDATIONS

AI generates `RECOMMENDATIONS.md` with prioritised optional improvements. User decides whether to act on them.

PRIORITY	EXAMPLES
High	Migrate remaining NgModule components to standalone • Enable stricter TypeScript flags • Replace `toPromise()` with `firstValueFrom()`
Medium	Remove redundant `CommonModule` imports • Enable `@if` / `@for` control flow syntax • Switch to esbuild builder (`@angular/build`)
Low	Install Angular DevTools • Migrate from Karma to Jest or Web Test Runner

> **Gate 2 — Optional.** AI delivers `RECOMMENDATIONS.md` but does not block on it. If the user opts in, a new workflow run is started focused only on recommendations.

FILES PRODUCED DURING THE WORKFLOW

FILE	CREATED	PURPOSE
`implementation_plan.md`	End of Phase 1	Full migration plan — presented at Gate 1 for human approval
`task.md`	Phase 3 start	Live checklist updated as each step completes
`REMAINING_TASKS.md`	Only if blocked	Completed steps, blocker details, remaining steps, suggested manual fixes
`RECOMMENDATIONS.md`	Phase 5	Prioritised optional post-migration improvements
`walkthrough.md`	End of workflow	Summary: what was migrated, verification results