# Web App Migration Checklist

## When to use

This checklist is intended as a reusable resource for engineering and technical leads involved in planning and executing migrations. Use it to:

- Scope and prepare a successful migration effort.
- Ensure consistency across teams and systems.
- Avoid common migration pitfalls and regression risks.
- Align migration activities with QA, product, and roadmap planning.
- Facilitate retrospectives and continuous improvement.

## How to use

Use this checklist when:

- Upgrading major frameworks, libraries, or dependencies.
- Replacing legacy code.
- Adopting new architecture patterns.
- Planning any multi-sprint technical migration or refactoring initiative.
- Planning cleanup following major feature launches where tech debt has accumulated.

## BEFORE migration starts

*What to consider and prepare in advance:*

- ☐ Define migration goals e.g. remove legacy systems, upgrade frameworks, improve maintainability.)
- ☐ Determine the migration strategy (incremental vs. Big Bang)
    - ☐ o Can changes be delivered safely alongside regular development (e.g., module by module, behind feature flags)?
    - ☐ o Is this a major framework rewrite or architecture shift that must be released all at once?
    - ☐ o Choose based on risk, coupling, testability, and team capacity.
- ☐ Run risk analysis for the migration approach
    - ☐ o What are the failure modes?
    - ☐ o What's the rollback plan if something goes wrong mid-migration?
    - ☐ o Are there downstream teams/products that will be impacted?
- ☐ Align product and UX expectations

    - [ ] o Ensure PMs and designers are aware of potential regressions or visual shifts.
    - [ ] o Make space for updated design reviews if needed.
- [ ] Audit the current system (identify affected features, technical dependencies, and dead code.)
- [ ] Estimate scope and timeline realistically (include buffers for testing, regression, and stabilization)
- [ ] Evaluate and document all necessary library/API replacements
- [ ] Plan to migrate to the latest supported versions to future-proof the implementation.
- [ ] Create one unified migration epic (avoid multiple epic environments at the same time, including epics for concurrent projects)
- [ ] Assign clear ownership (engage engineers familiar with the original code).
- [ ] Plan for regression testing (involve QA early and align timelines accordingly).
- [ ] Communicate system and team dependencies
- [ ] Define success criteria and a “definition of done”

## DURING migration

*How to execute smoothly and avoid common pitfalls:*

- [ ] Schedule dedicated migration sprints or weeks
- [ ] Avoid concurrent feature development (prevent context switching and priority conflicts.)
- [ ] Avoid multiple epic environments (avoid multiple epic environments at the same time, including epics for concurrent projects)
- [ ] Enforce code freeze if possible (otherwise, duplicate work will be needed)
- [ ] Break work into actionable tickets
- [ ] Simplify and refactor during migration (don’t migrate unnecessary complexity).
- [ ] Remove deprecated or unused code and files (e.g., legacy modules, unused configurations).
- [ ] Have the original authors refactor their code (speeds up work and reduces the learning curve).
- [ ] Collaborate closely with QA (Continuous testing vs full regressions)
- [ ] Document and reuse migration patterns
- [ ] Use AI/automation where safe and beneficial
- [ ] Keep migration visible in all planning meetings
- [ ] Document edge cases and legacy behavior as you migrate
- [ ] Log deprecated features explicitly

## AFTER migration

*Wrap-up, validation, and learning capture:*

- ☐ Remove all compatibility layers and workarounds
- ☐ Complete full regression and performance testing
- ☐ Ensure deprecated systems are fully removed (clean up codebase, CI configs, documentation).
- ☐ Document outcomes and learnings (share internally via Confluence, Slack, etc.)
- ☐ Run a migration retrospective (capture insights and improvements for next time).
- ☐ Celebrate completion (reinforce the business value and team effort).
- ☐ Sunset old knowledge bases and update docs

## OUTSIDE major migrations (ongoing practices)

*Stay ahead of tech debt and migration fatigue:*

- ☐ Continuously refactor as part of feature work
- ☐ Potentially establish an “Annual Migration Month” and use it to proactively clean up and upgrade.
- ☐ Set a minimum migration/refactor ticket goal per sprint
- ☐ Balance technical debt in roadmap planning
- ☐ Monitor library/framework versioning (track EOL and major version deprecations).
- ☐ Engineers to own system health (empower cleanup and migration-minded thinking).