CODE QUALITY & MAINTAINABILITY

The Real Reason Your App Slows Down Over Time

Shipping fast today means nothing if you're stuck debugging tomorrow. Let’s break down what really makes your codebase scalable, sane, and future-proof.

golden earth

2

WHY CODE QUALITY MATTERS

- Maintenance often costs more than initial development
- Codebase complexity = slower iteration
- Technical debt compounds like interest
- Clean code allows agility, scale, and speed
- Future developers (or your future self) will thank you

golden earth

3

NAMING CONVENTIONS

- Consistent names improve clarity and debugging speed
- Entities: Singular nouns (Customer)
- Attributes: Specific purpose (firstName, totalAmount)
- Microflows: Verb + Subject (ACT_SendEmail)
- Pages: Purpose + Type (Order_Overview)
- Document and enforce naming rules across teams

golden earth

4

SELF-DOCUMENTING DESIGN

- Your microflow should explain itself
- Visual clarity > clever tricks
- Variables and parameters should tell a story
- Comments = not first option
- Code reviews should include readability feedback

golden earth

5

DOCUMENTATION BEST PRACTICES

- Explain the why behind complex rules
- Write down decisions that devs can’t “guess” later
- Maintain module-level README files
- Create and share a central documentation repo
- Keep docs close to code (inline + linked)

golden earth

REUSABLE COMPONENTS

- Don’t repeat yourself—build once, use everywhere
- Create central libraries for common logic
- Design UI elements as components
- Wrap integrations into reusable connectors
- Maintain versions with clear changelogs

MODULARIZATION STRATEGY

- Split your app into business-focused domains
- Enforce clear boundaries and ownership
- Define responsibilities per module
- Track and document inter-module dependencies
- Enable independent deployment and scaling

TESTING STRATEGY

- Unit test your core logic
- Automate high-risk UI flows
- Test using production-like data volumes
- Simulate edge cases, not just happy paths
- Integrate tests with CI/CD pipeline
- Prioritize test coverage for mission-critical flows

MANAGING TECHNICAL DEBT

- Track all tech debt (not just in your head)
- Flag it in your backlog (with a label or tag)
- Schedule time for refactoring—don't "fit it in" later
- Review architecture decisions quarterly

CODE REVIEW PROCESS

- Have a clear review checklist (naming, logic, reuse, docs)
- Review for clarity, not just bugs
- Use reviews as mentorship, not just gates
- Log decisions and changes post-review
- Encourage pair reviews for tricky sections

ANTI-PATTERNS TO AVOID

- Copy-pasting instead of extracting reusables
- Vague variables like **x, data1, tempValue**
- Logic duplicated across modules
- Massive, all-in-one microflows
- Unexplained workarounds and magic values
- Poorly scoped commits (e.g., “fix issue” with 80 changes)

LET’S TALK

Every team battles maintainability issues.
What’s the biggest code quality challenge you’ve faced?

Drop your stories, frameworks, or hard-earned tips in the comments

Let’s help each other build better.

hello@goldenearth.io

www.goldenearth.io

golden earth