Module 4.5: Testing

Developer saying "I tested it" means they tried the happy path once.

The Non-Negotiable Rule

QA is not optional for ANY project.

Small Project

$20K-30K

Part-time QA required

Medium Project

$30K-50K

Regular QA required

Large Project

$50K+

Dedicated QA required

30%

The 30% Rule

QA should consume 30% of your development hours

When technical documentation is ready, show it to QA to clarify acceptance criteria, identify edge cases, and prepare test cases in parallel with development.

Early QA Involvement

Involve QA **before coding begins** and keep them **updated during development** for ongoing testing alignment.

After UI/UX is Done

Review User Flows

Ensure they match the user stories

Spot Issues Early

Identify potential usability or consistency problems

Confirm Criteria

Verify acceptance criteria are visually supported in designs

Before Sprint

Sprint Planning – Day 1

01

Review Stories

Review upcoming stories and acceptance criteria

02

Clarify Edge Cases

Clarify edge cases with PM and developers

03

Estimate Effort

Estimate testing effort and define test scope

04

Prepare Test Cases

Prepare or update test cases for new stories

During Sprint

Days 2–10

Core Testing Activities

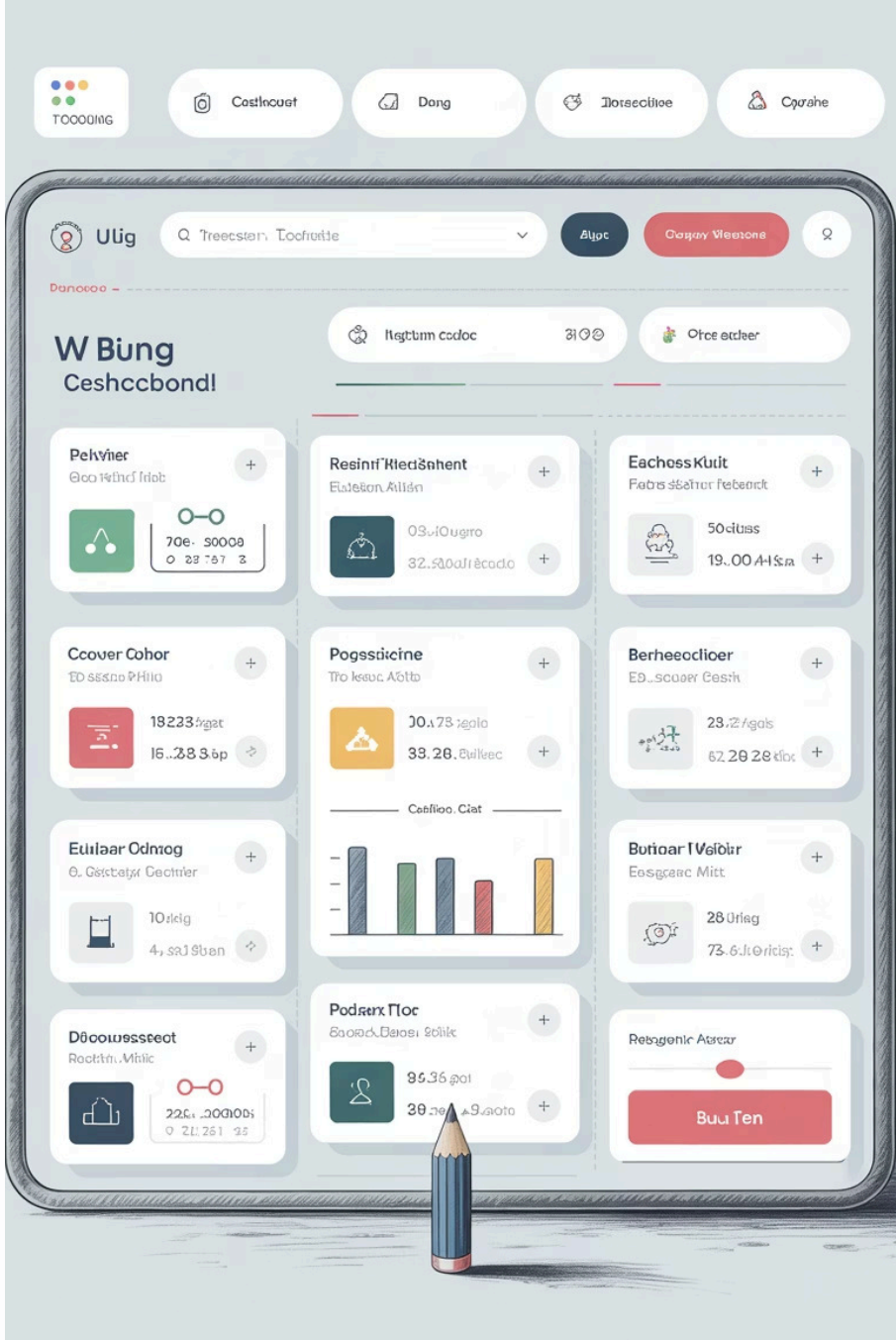
Test each story as soon as it's ready in the dev/staging.

- Functional testing (core features)
- Regression testing (older features not broken)
- UI validation (matches design)
- Basic API checks (if applicable)

Daily Workflow

Log bugs in the tracker and re-test after fixes.

Daily standups: share test progress and blockers.



End of Sprint & After

Days 11–13: End of Sprint

- Full regression on integrated build
- Test across browsers/devices
- Verify all "done" stories meet Definition of Done
- Prepare test summary or release notes

Day 14: Review & Retro

- Demo testing results during review
- Note improvement areas (bug trends, unclear stories)
- Update QA documentation/test cases for next sprint

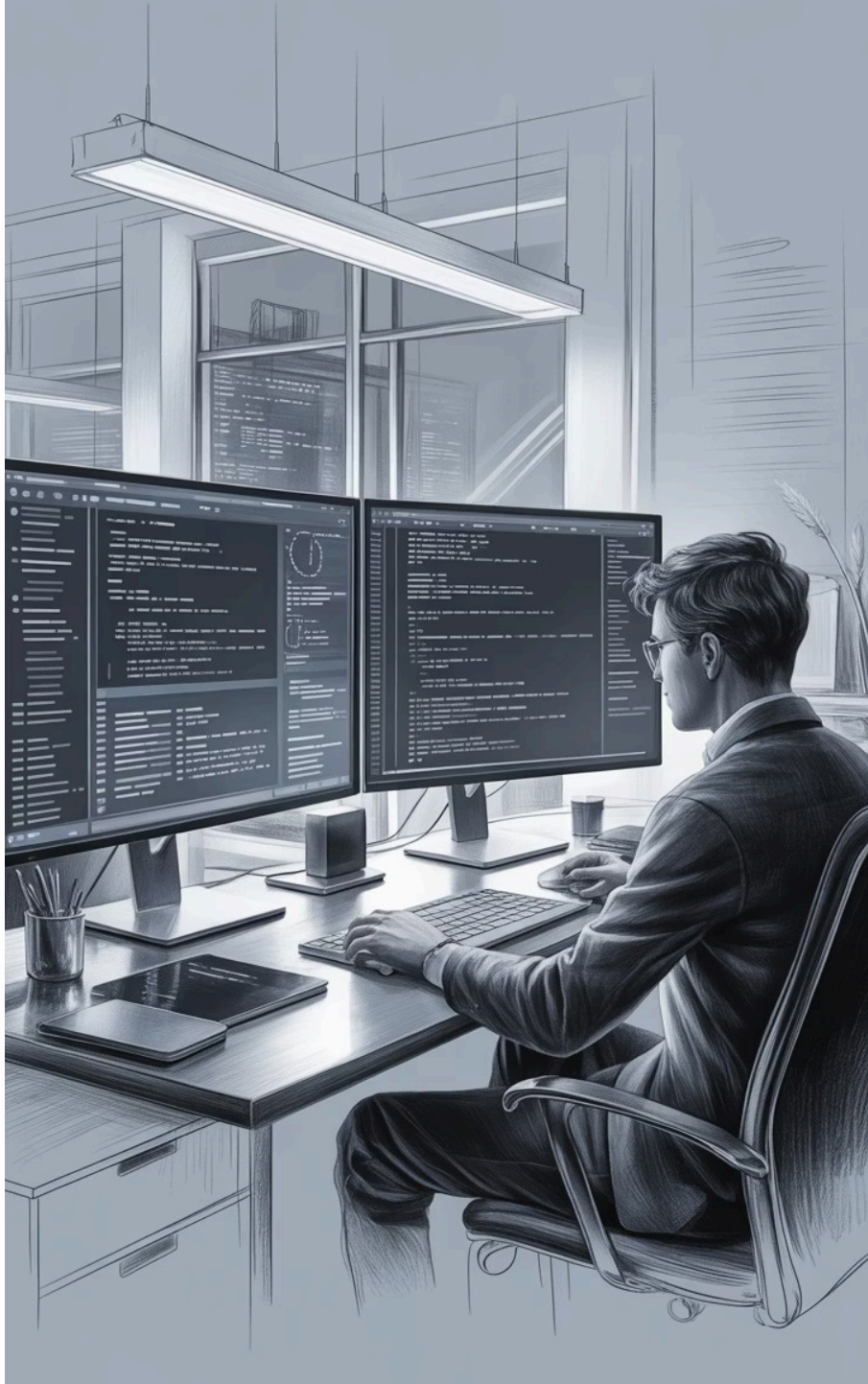
Developer Testing Before "Done"

Code Level

- Unit tests for core logic (happy + key edge cases)
- Lint/format pass; no TypeScript/ESLint errors
- Static analysis & basic security checks
- Meaningful logs + handled errors

Feature Level

- Verify acceptance criteria (Given/When/Then)
- Manual "happy path" + 3–5 edge cases
- Input validation & empty/error states
- Performance sanity check
- Cross-environment sanity testing
- API: correct status codes, errors surfaced
- Data: migrations run, test data available



PR Readiness Checklist

1

Linked Story & AC

Story linked and acceptance criteria ticked off

2

Documentation

Screenshots/video of flow with test notes

3

Known Limitations

Known limitations clearly listed

4

Tests & CI

Unit test coverage updated and CI green

Dev Testing Connects with QA

Shared Acceptance Criteria

Single source of truth for both dev and QA

Testing Flow

Devs test first, then QA verifies independently

Bug Loop

QA logs → dev fixes → dev re-tests → QA re-checks

Handover

Build link, test notes, toggles, test data, affected areas

Simple "Definition of Done"

AC met ✔️ | Unit tests ✔️ | CI green ✔️ | Self-test notes/screens ✔️ | Perf basics ✔️ | No critical known issues ✔️ | QA passed ✔️

Use these as your **checklists in each PR and sprint**. Devs prevent obvious issues; QA protects the whole product.

QA Must Understand Business Logic

Test Real User Flows

Not just buttons, but actual user journeys

Catch Logical Errors

Wrong pricing, incorrect flow order, business rule violations

Better Test Cases

Reflect user intent, not just functionality

QA doesn't need to know marketing strategy, but they must clearly understand **what the app is meant to achieve and how users will use it.**

Why QA Matters

- QA prevents costly bugs that delay launch or damage user trust
- It ensures the product works as described in user stories
- Skipping QA early often doubles the cost later

QA Roles in an MVP

Test Early and Often

QA should join from sprint 1, not just before launch

Performance & Device

Basic performance and device testing

Functional Testing

Does it work as intended?

Regression Testing

Old features not broken

UI/UX Consistency

Visual alignment with designs

QA Deliverables

1

Test Cases or Checklist

Comprehensive testing documentation for each feature

2

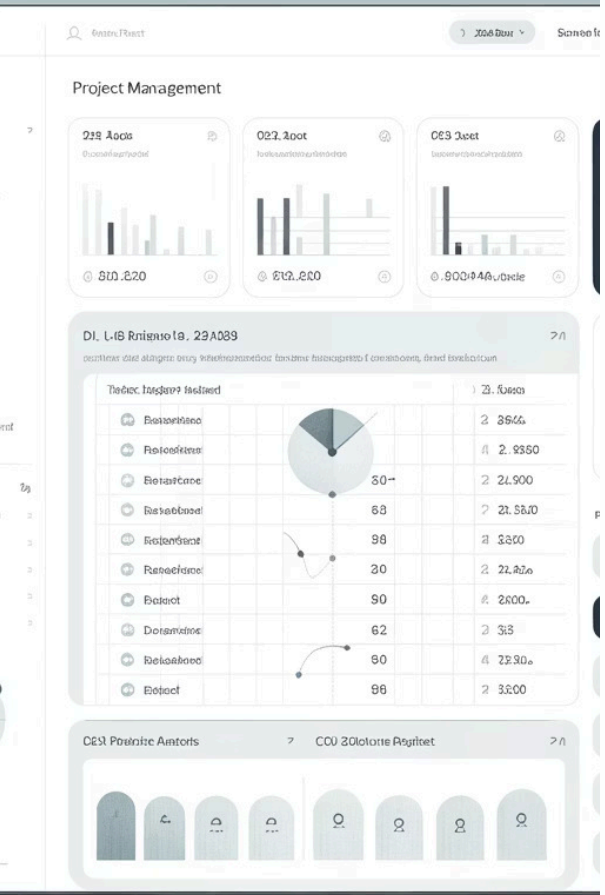
Bug Reports

Clear steps to reproduce with expected vs actual results

3

Definition of Done

Confirmation for each feature meeting quality standards



Typical QA Deliverables in a $30–50k MVP

Test Plan or Checklist

Per sprint documentation outlining testing scope and approach

Bug Tracker

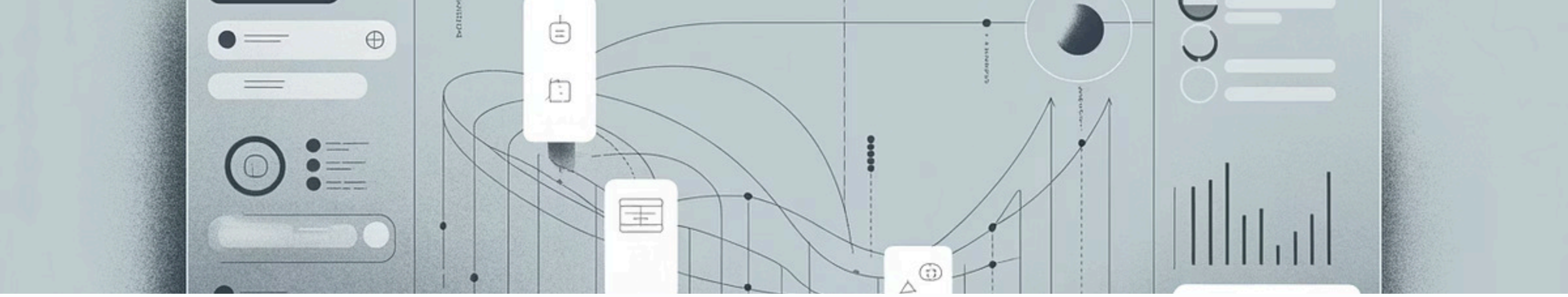
Jira, Linear, or Notion for logging and tracking issues

Test Evidence

Screenshots, videos, and documentation of testing activities

Weekly QA Report

Summary of testing progress, bugs found, and quality metrics



QA Communication & Tools

For Non-Tech Founders

Understanding where and how QA communicates is essential for project transparency.

- Where QA logs bugs (Jira, ClickUp, Notion)
- What a good bug report looks like
- QA reports to PM, not directly to founder

Good Bug Report Elements

1. Steps to reproduce
2. Expected vs actual results
3. Screenshots or video evidence
4. Environment details (device, OS, browser)
5. Severity and priority

Automated Testing and Code Coverage

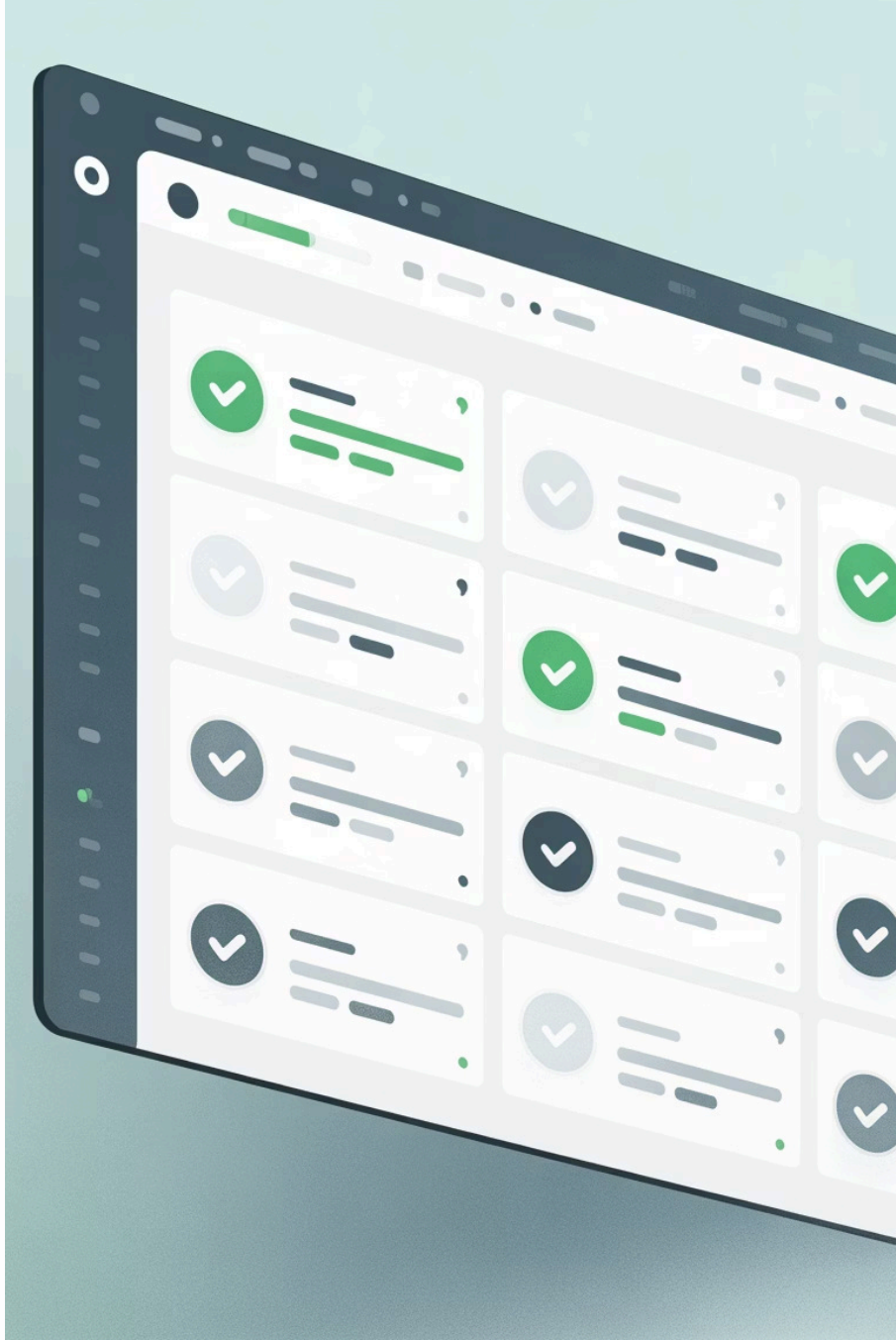
Manual testing catches most visible issues, but it's slow and easy to miss small regressions. That's where **automated testing** helps.

What Automated Tests Are

They're small scripts developers write to **check that key features still work** every time new code is added.

Example 1	Example 2
"User can log in"	"Payment goes through"
Example 3	
"Data saves correctly"	

If one breaks, the system alerts the team before release.



Why It Matters for MVPs

Prevents Regressions

Stops bugs that reappear after fixes

Saves Time

Eliminates hours of manual retesting each sprint

Builds Confidence

Release faster and more often with assurance

For a $30–50k MVP, you don't need full automation — just cover **critical flows** (signup, payment, core feature).

Code Coverage

"Coverage" means **what percentage of the code is checked by automated tests**.

No Coverage

Nothing tested automatically

Moderate Coverage

Half of core logic tested

High Coverage

Most functions validated

You don't need to hit a magic number. Focus on **covering the riskiest and most used parts** — like payments, onboarding, or data handling.

What to Ask Your Team

"Do we have basic automated tests for key flows?"

"What's our code coverage for critical logic — 20%, 50%, or more?"

"Can we run all tests automatically before release?"

These questions keep devs accountable without you needing to understand the code.

Example Bug Report

Bug Title

💥 "Login button doesn't respond after entering correct credentials"

Environment

- Device: iPhone 14
- OS: iOS 18.0
- App version: 1.0.2 (staging)
- Browser: Safari

Steps to Reproduce

1. Open the app
2. Tap "Login"
3. Enter valid email and password
4. Tap the "Login" button

Expected Result

User should be redirected to the dashboard screen.

Actual Result

Nothing happens; the button stays in loading state.

Severity & Priority

Severity: High (blocks login)

Priority: P1 (must fix before release)

Additional Notes

Issue only happens on iOS; Android works fine. API request succeeds (200 OK), but frontend doesn't navigate.

📸 Screenshot or screen recording link attached