HOW TO BECOME AN SDET in 90 Days

Skills • Tools • Automation Framework • CI/CD • Portfolio

A practical 12-week roadmap to build job-ready automation skills and portfolio evidence.

Python | Playwright | Pytest | API | CI/CD

Publish-Safe Roadmap Expectation

A 90-day sprint can build skill and portfolio evidence, but it is not a job guarantee.

What is realistic

Build foundations, automation projects, CI experience, documentation, and interview stories in 12 focused weeks.

What to avoid

Do not claim guaranteed hiring, salary outcomes, or that one tool alone makes someone an SDET.

What this deck uses

Accurate tool roles, modern framework guidance, practical labs, and a portfolio-first learning path.

Key correction: “job-ready” means ready to demonstrate practical projects and apply for realistic QA automation/SDET roles—not a guaranteed job in 90 days.

What an SDET Actually Does

An SDET blends programming, testing, debugging, and delivery workflow.

- Designs automated tests for UI, API, services, and integrations
- Builds maintainable frameworks instead of one-off scripts
- Works with developers, QA, product, and DevOps teams
- Finds defects, explains risk, and improves feedback speed
- Runs tests locally and in CI pipelines

The SDET Skill Stack

Think in layers: foundation, automation, framework, and delivery.

Programming

Python basics, OOP, clean code

UI Automation

Selenium fundamentals, Playwright practice

API Testing

REST, HTTP methods, JSON, Postman, requests

Frameworks

POM, pytest fixtures, assertions, reports

CI/CD

Git, GitHub, GitHub Actions, test reports

Professional Skills

debugging, flaky tests, bug lifecycle, communication

SDET hiring conversations usually focus on whether you can explain your framework, troubleshoot failures, and run tests in a delivery workflow.

The 12-Week Roadmap Overview

Three months, three focused outcomes.

Weeks 1–2: Python Foundations

Start with logic, not tool memorization.

- Variables, conditions, loops, functions
- Lists, dictionaries, strings, files, and exceptions
- Simple problem solving: 20–30 beginner coding exercises
- Write readable functions with clear names
- Practice debugging using print, breakpoints, and error messages

Sample learning loop

Example: create a function that validates login input, then write small tests for valid and invalid values.

OOP and Clean Test Code

Automation frameworks become easier when code is organized.

OOP essentials

Classes, objects, methods, inheritance, composition, and reusable behavior.

Clean code habits

Clear names, small functions, no repeated locators, readable assertions, and useful errors.

Testing mindset

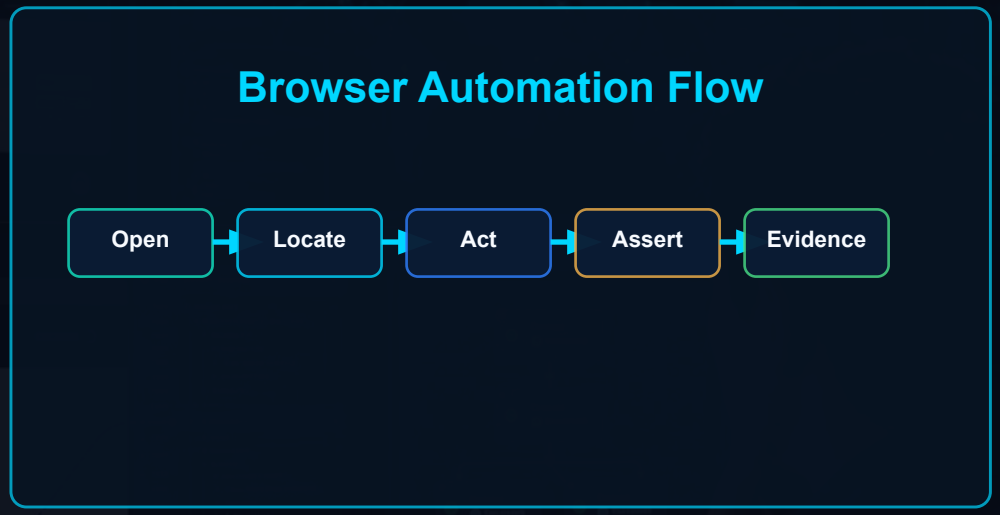
Do not only make the script pass. Make the failure easy to understand.

Bad automation hides problems. Good automation explains the problem clearly.

Weeks 3–4: Web UI Automation

Learn how browser automation really works.

- Use Selenium briefly to understand WebDriver, drivers, waits, and locators
- Practice XPath and CSS selectors, but avoid fragile selectors when possible
- Move into Playwright for modern end-to-end automation practice
- Automate login, forms, navigation, search, and validation flows
- Capture screenshots, traces, and failure evidence



Reliable Locators and Flakiness Control

Modern automation is judged by stability, not just passing once.

Prefer resilient locators

Use accessible roles, labels, text, stable attributes, or agreed test IDs where possible.

Avoid timing hacks

Use framework waits and assertions instead of random sleep statements.

Control test data

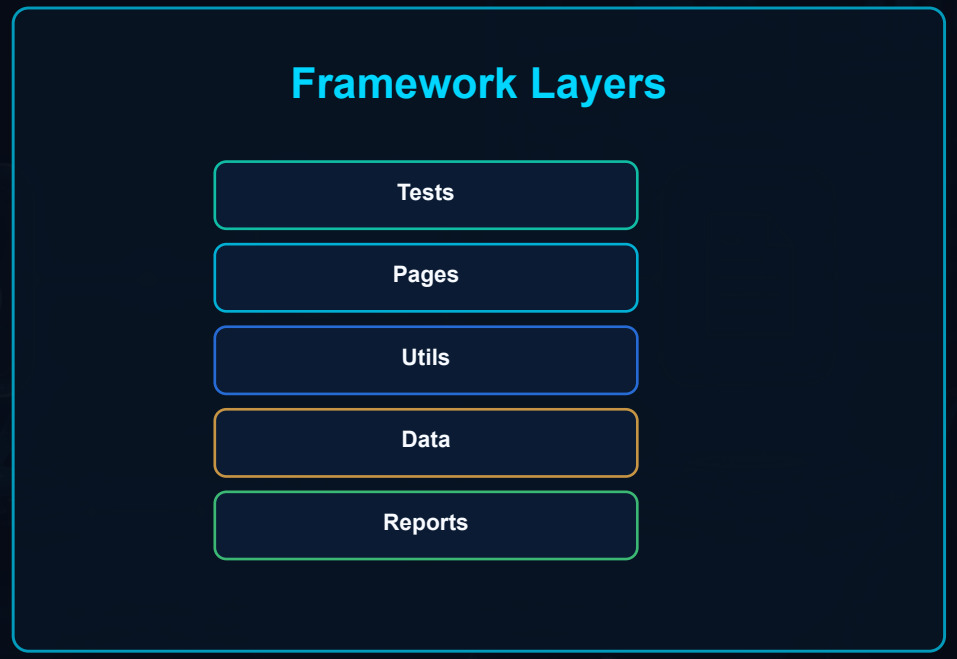
Create predictable data, clean up after tests, and avoid order dependency.

Flaky test question: Did the application fail, or did the automation fail to wait, locate, or isolate data correctly?

Weeks 5–6: Framework Architecture

Move from scripts to a maintainable test framework.

- Use Page Object Model to separate tests from page-specific locators and actions
- Organize code into tests, pages, utilities, data, config, and reports
- Keep assertions in tests and reusable actions in page classes
- Make failures easy to diagnose through logs, screenshots, and reports



Pytest Execution and Reporting

A framework needs a professional test runner.

Fixtures

Setup and teardown for browser, data, API clients, and shared test resources.

Assertions

Validate expected behavior clearly and fail with useful feedback.

Parametrization

Run the same logic against multiple users, inputs, roles, or browsers.

Reports

Generate HTML or JUnit-style outputs for visibility in local runs and CI.

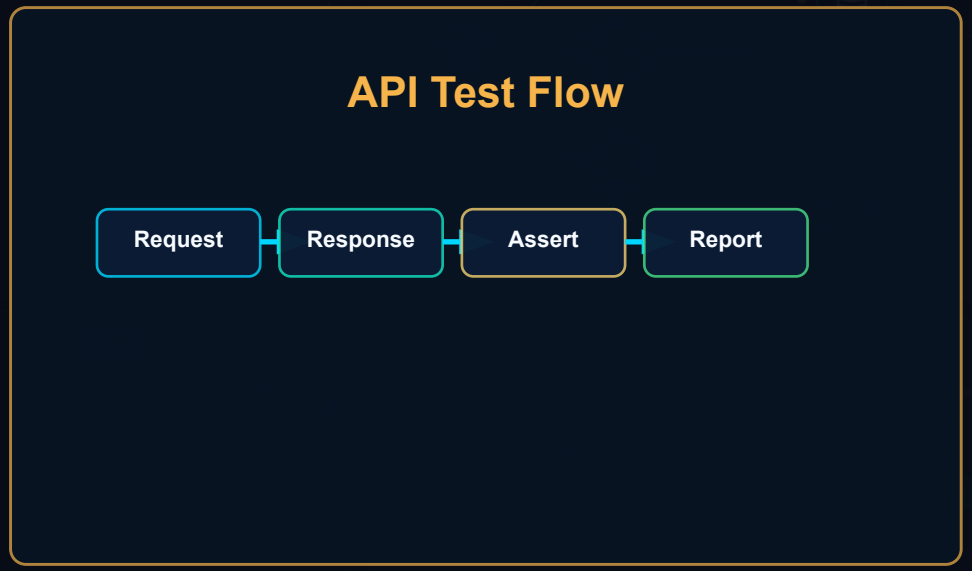
Markers

Separate smoke, regression, API, UI, and slow tests for targeted execution.

Weeks 7–8: API and Service-Layer Testing

Robust automation tests below the UI, not only through the browser.

- Understand REST concepts and resources
- Practice HTTP methods: GET, POST, PUT, PATCH, DELETE
- Validate status codes, headers, response time, and JSON body data
- Use Postman for exploration and collections
- Automate checks with Python requests or an API client layer



Automated API Test Patterns

API automation should test behavior, data, and error handling.

Positive tests

Expected request returns expected response

Negative tests

Invalid input returns correct error

Auth tests

Unauthorized request is rejected

Data checks

JSON fields match expected rules

Chaining

Use data from one response in the next request

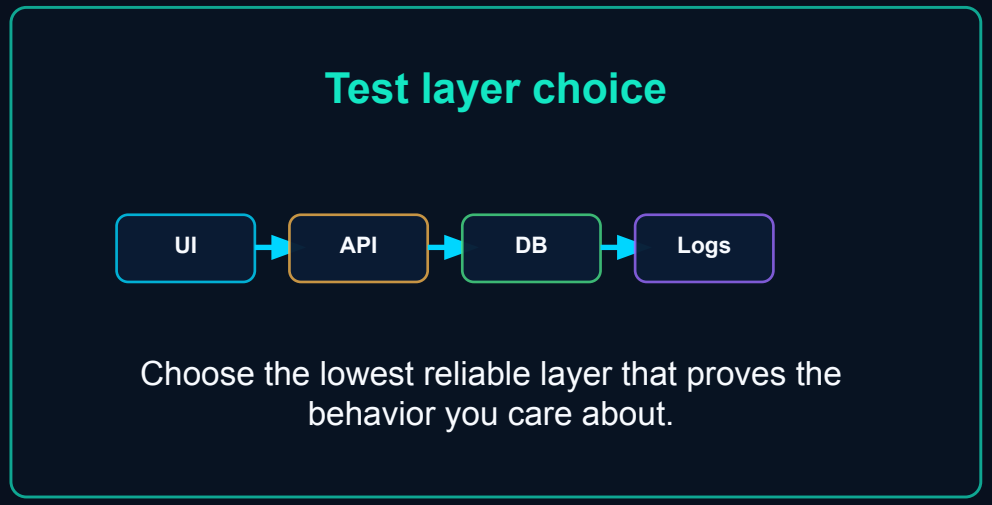
Cleanup

Reset or delete test data when needed

Test Data, Database, and Service Thinking

Good automation controls the conditions around the test.

- Understand setup, teardown, and repeatable data
- Learn basic SQL SELECT, WHERE, JOIN, and simple validation queries
- Use test accounts, test environments, and safe sample data
- Know when to test through UI, API, database, or service layer
- Document assumptions so failures are easier to investigate



Weeks 9–10: Git and GitHub Workflow

Enterprise testing lives inside the development workflow.

- Use branches for isolated changes
- Write meaningful commit messages
- Open pull requests for review
- Keep README instructions clear
- Track issues and bugs with concise evidence

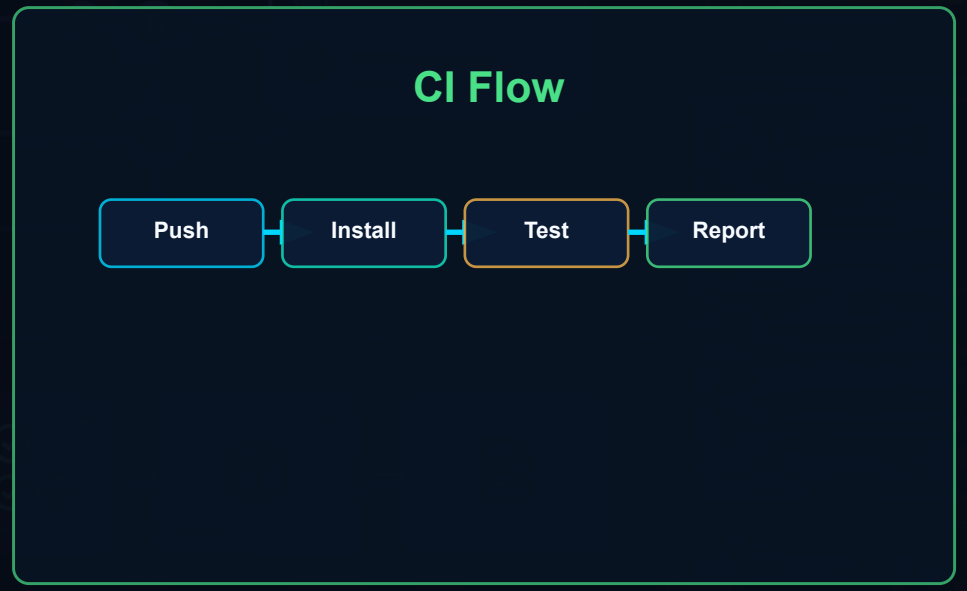
Portfolio signal

A clean repository shows how you think, not only what you wrote.

GitHub Actions CI/CD Pipeline

Prove your tests can run automatically after code changes.

- Create a workflow file under .github/workflows
- Trigger tests on push or pull request
- Install dependencies and browsers in the runner
- Run pytest, Playwright, API tests, or selected suites
- Upload reports, screenshots, traces, and logs when possible



Modern QA Productivity with AI

Use AI as an assistant, not as an unchecked automation author.

Useful AI assistance

Generate test ideas, explain failures, draft edge cases, summarize logs, and improve README wording.

Human validation

Review every locator, assertion, test data assumption, and generated code before using it.

Never expose secrets

Do not paste credentials, customer data, private source code, or confidential logs into unapproved tools.

AI can speed up thinking, but the SDET still owns correctness, maintainability, and risk.

Portfolio Capstone Architecture

Your portfolio should show one complete testing system, not random snippets.

- README with setup and execution instructions
- Clean folder structure and Page Object Model
- UI and API tests with clear assertions
- GitHub Actions workflow and test report artifacts
- Short portfolio summary explaining design decisions

Interview and Resume Positioning

Explain your work honestly and practically.

Common interview topics

- Framework architecture and POM
- Locator strategy and flaky-test handling
- API validation and test data
- GitHub Actions pipeline
- Bug lifecycle and communication

Live conversation

Interviewer: “How do you handle flaky tests?”
Candidate: “I first check whether the issue is timing, locator stability, test data, environment, or a real product defect. Then I fix the root cause instead of hiding the failure.”

Final Action Plan and Next Step

Understand. Build. Prove. Apply.

Understand

Python, testing concepts, web, API, CI/CD

Build

One complete automation framework with UI + API tests

Prove

Reports, README, GitHub Actions, portfolio explanation

Apply

Target QA Automation, SDET Intern/Junior, Test Automation Engineer roles

Start with one project. Make it clean. Document it clearly. Then use it to explain how you think as an automation engineer.

For structured technology learning and career-focused guidance, explore ITLearn360.