The purpose of this document is coalesce the terminology we use around testing to support collaboration, whilst at the same time encouraging good practice, particularly around following a layered approach. This document also acts as a useful template for your projects Testing Strategy. The Test Types are drawn from a holistic model of an application that sits inside an ‘eco system’, but views the application as an independent component rather than part of a monolith or pyramid.



NB. For UI applications, the state is the DOM, and this is subject to external mutation which makes it very difficult to control :(

# 

# Layer ‘A’

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Type** | **Why** | **How** | **Where** |
| **Define my application under test** | So I can support independent testing and fail fast and fail true | Look at commonality of; deployment, technology, state, contract definition, delivery responsibility and delivery cycles etc. | Collaborate |
| **Unit**  **(white box)** | Verify the technical design,  support a TDD approach, , living documentation, and fail fast | Test each method in each end-point, mocking dependent methods. | CI env |
| **Code Coverage** | Demonstrate robustness.  Missed tests | Unit &  Functional |  |
| **Static analysis** | Verify code style, design, correctness and efficiency, exposures,dead code | Tools such as Sonar, Jacoco, Codacy, Veracode, | CI env  GitHub |
| **Peer Review** |  | On GitHub pull request |  |
| **Functional**  **(black box)** | Verifying the deployed app:   1. is meeting ACs 2. is transitioning local state 3. calling 3rd parties ‘**smart** **stubs’** correctly 4. for APIs, is it consistent with swagger contract and swagger reflects the API 5. For UIs, layout testing | 1. BDD tests against my service or BDD tests against UI components. Using ‘smart’ stubs of dependencies. Can use; Cucumber for API, Chai for F-E JS (BDD tools available for Angular and React), and Appium+Cucumber for mobile. 2. Verify domain state transition with journeys across endpoints of UI components 3. Verify the content of calls to 3rd parties 4. For APIs and SmartStubs, run same tests against the stub of my service and check same validations and payloads 5. For UIs, Visual comparisons across devices | CI env  Should run in <10mins  Also trigerred when contract changes for third parties |
| **Performance**  **Load** | Will I meet my SLA under peak prod load  Load testing can verify scalability. | Run typical user journeys in CI, with stubs for dependent services (having SLA latency) under peak prod load. Capture response times, requests per second.  Tools include Gatling, JMeter, WebPageTest | Prod like for the app <10mins |
| **Performance**  **Stress** | Will I continue to meet my SLA unexpected peaks. Can I recover from the peaks. | Run the Load test, with extreme loa, ramping up from normal load, to extreme load causing errors, then back to normal load | Prod like env |
| **Performance**  **Soak** | Will I continue to meet my SLA over extended time | Run the Load test over a long period. Capture response times, requests per second, threads, cpu, memory etc  Bear in mind some envs restart |  |
| **Static Security** | Can I bypass the interface to expose private assets or perform invalid actions | Using tools such Qualys |  |
| **Dynamic Security** | Find bugs  Visual checks  Ethical hacking | Bring domain knowledge and QE ingenuity to identify bugs and sense-check the app | QA/PreProd  ad hoc |
| **Infrastructure** | Resilience | Chaos testing. Eg take down a DB | Ad Hoc |
| **Configuration Validity** | Validate config meets ACs | Lint or comparisons | CI |
| **Heart beat** | Prod health check | Verify the health of the application by running key scenarios in Prod | Prod |
| **Browser & device compatibility** | Does my app function across my distribution channels | Re-run functional tests across browsers and devices | BrowserStack |
| **SEO** | Do we have an acceptable SEO score |  | Eg DeepCrawl |
| **Exploratory** | Can I break the app | Human ingenuity and experience.  Innovative ways of testing; bug bashes, crowd sourcing etc |  |
| **Fallback & Recovery** | Can I recover from fallback, do I need to clean-up | Install new version, run tests, fallback and install previous version without clearing local and persisted data, run previous version tests | CI |
| **Accessibility Testing** | Is application accessible to disabled person? | Open source tools |  |

# Layer B

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Type** | **Why** | **How** | **Where** |
| **Dependency Integration** | Check integration with 3rd party dependencies  i.e. services you rely on.  **Only needed if you change or contracts change.** | Either... A small set of tests checking the integration with real 3rd party dependencies..  Or  if you are using SmartStubs and they represent tight contracts and your service verifiers that it honours the contract...Nothing to do as, Your ‘’Functional’’ test should be triggered on a dependent service contract change. Or If you are using client sdks then again nothing as your integration is checked at compile time |  |

# Layer C

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Type** | **Why** | **How** | **Where** |
| **Downstream** | If the contract is not tight then we need to check the payload validity to APIs and downstream | Maybe manual or triggering the downstream test passing in your data |  |

# 

# 

# Layer D

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Type** | **Why** | **How** | **Where** |
| **Consumer Integration** | Check integration with 3rd party consumers, services or apps that rely  on you.  **Only needed if contract changes.** | Either…  Run your PACT tests or trigger the consumer’s Dependency Integration tests  Or  if you are using SmartStubs and they represent tight contracts and the service verifies it is honoring the contract ...  Nothing to do as, Consumers’ Functional’’ test **should be triggered on your contract change.** Or If your consumers are using client sdks then nothing to do as, Consumers’ Functional’’ test should be triggered on your contract change.  Or  Kick of Consumer ‘B’/’C’ |  |

# 

# Where To Start

A good starting implementation would be to build in CI:

* level ‘A’ functional (stubbed back-ends) + performance load (if needed) + static security tests
* level ‘B’ dependency integration test (subset of functional with real dependencies)

N.b. Plus manual exploratory testing