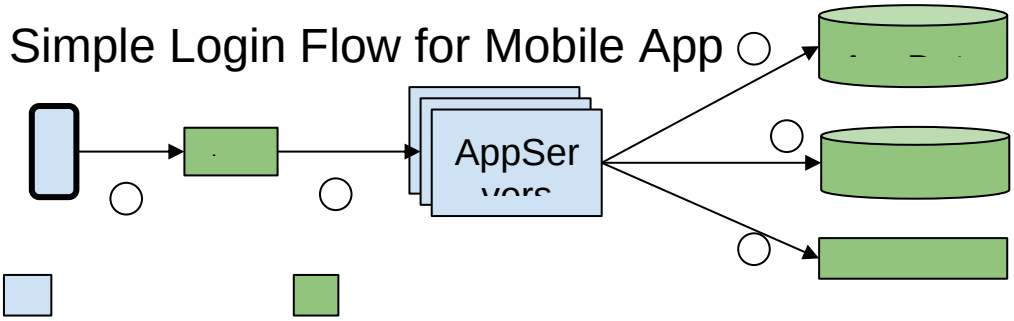
Testing on the Toilet

What Makes a Good End-to-End Test?

An end-to-end test tests your entire system from one end to the other, treating everything in between as a black box. **End-to-end tests can catch bugs that manifest *across* your entire system**. In addition to unit and integration tests, they are a critical part of a balanced testing diet, providing confidence about the health of your system in a near production state. Unfortunately, end-to-end tests are **slower, more flaky, and more expensive to maintain** than unit or integration tests. Consider carefully whether an end-to-end test is warranted, and if so, how best to write one.

Let's consider how an end-to-end test might work for the following "login flow":



In order to be cost effective, an end-to-end test should **focus on aspects of your system that cannot be reliably evaluated with smaller tests**, such as resource allocation, concurrency issues and API compatibility. More specifically:

- **For each important use case, there should be one corresponding end-to-end test.** This should include one test for each important class of error. The goal is the keep your total end-to-end count low.
- Be prepared to **allocate at least one week a quarter per test to keep your end-to-end tests stable** in the face of issues like slow and flaky dependencies or minor UI changes.
- **Focus your efforts on verifying overall system behavior instead of specific implementation details**; for example, when testing login behavior, verify that the process succeeds independent of the exact messages or visual layouts, which may change frequently.
- **Make your end-to-end test easy to debug** by providing an overview-level log file, documenting common test failure modes, and preserving all relevant system state information (e.g.: screenshots, database snapshots, etc.).

End-to-end tests also come with some important caveats:

- System components that are owned by other teams may change unexpectedly, and break your tests. This increases overall maintenance cost, but can highlight incompatible changes
- **It may be more difficult to make an end-to-end test fully hermetic**; leftover test data may alter future tests and/or production systems. Where possible keep your test data ephemeral.
- An end-to-end test often necessitates multiple test doubles (fakes or stubs) for underlying dependencies; they can, however, have a high maintenance burden as they drift from the real implementations over time.

More information, discussion, and archives:

testing.googleblog.com