Puppeteer unit tests

Unit tests in Puppeteer are written using Mocha as the test runner and Expect as the assertions library.

Test state

We have some common setup that runs before each test and is defined in `mocha-utils.js`.

You can use the `getTestState` function to read state. It exposes the following that you can use in your tests. These will be reset/tidied between tests automatically for you:

- `puppeteer`: an instance of the Puppeteer library. This is exactly what you'd get if you ran `require('puppeteer')`.
- `puppeteerPath`: the path to the root source file for Puppeteer.
- `defaultBrowserOptions`: the default options the Puppeteer browser is launched from in test mode, so tests can use them and override if required.
- `server`: a dummy test server instance (see `utils/testserver` for more).
- `httpsServer`: a dummy test server HTTPS instance (see `utils/testserver` for more).
- `isFirefox`: true if running in Firefox.
- `isChrome`: true if running Chromium.
- `isHeadless`: true if the test is in headless mode.

If your test needs a browser instance, you can use the `setupTestBrowserHooks()` function which will automatically configure a browser that will be cleaned between each test suite run. You access this via `getTestState()`.

If your test needs a Puppeteer page and context, you can use the `setupTestPageAndContextHooks()` function which will configure these. You can access `page` and `context` from `getTestState()` once you have done this.

The best place to look is an existing test to see how they use the helpers.

Skipping tests in specific conditions

Tests that are not expected to pass in Firefox can be skipped. You can skip an individual test by using `itFailsFirefox` rather than `it`. Similarly you can skip a describe block with `describeFailsFirefox`.

There is also `describeChromeOnly` and `itChromeOnly` which will only execute the test if running in Chromium. Note that this is different from `describeFailsFirefox`: the goal is to get any `FailsFirefox` calls passing in Firefox, whereas `describeChromeOnly` should be used to test behaviour that will only ever apply in Chromium.

There are also tests that assume a normal install flow, with browser binaries ending up in `.local-<browser>`, for example. Such tests are skipped with `itOnlyRegularInstall` which checks `BINARY` and `PUPPETEER_ALT_INSTALL` environment variables.

Running tests

Despite being named 'unit', these are integration tests, making sure public API methods and events work as expected.

- To run all tests:

```
npm run unit
```

- **Important**: don't forget to first run TypeScript if you're testing local changes:

```
npm run tsc && npm run unit
```

- To run a specific test, substitute the `it` with `it.only`:

```
  ...
  it.only('should work', async function() {
    const {server, page} = getTestState();
    const response = await page.goto(server.EMPTY_PAGE);
    expect(response.ok).toBe(true);
  });
```

- To disable a specific test, substitute the `it` with `xit` (mnemonic rule: '*cross it*'):

```
  ...
  // Using "xit" to skip specific test
  xit('should work', async function({server, page}) {
    const {server, page} = getTestState();
    const response = await page.goto(server.EMPTY_PAGE);
    expect(response.ok).toBe(true);
  });
```

- To run tests in non-headless mode:

```
HEADLESS=false npm run unit
```

- To run tests with custom browser executable:

```
BINARY=<path-to-executable> npm run unit
```