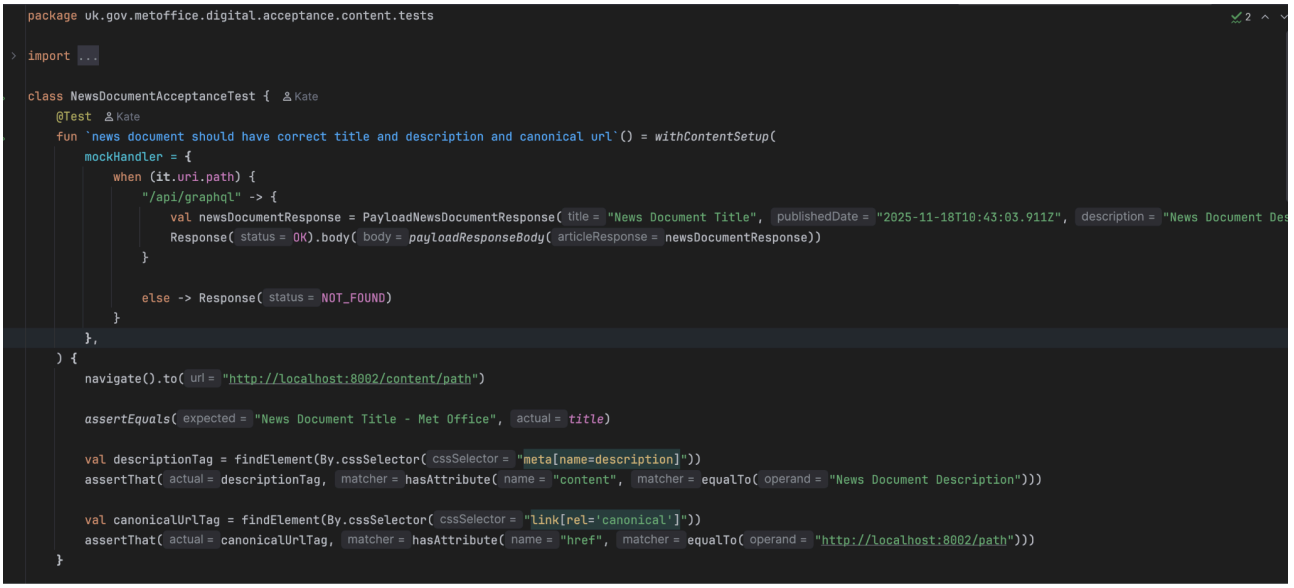
Demonstrate use of a Mocking Library or use of an E2E Framework on you Delivery Project

This is an example of using mock testing combined with acceptance/E2E testing using Selenium on my Delivery Project.

In this test, the real backend GraphQL API is replaced with a mock HTTP handler that returns a fabricated Payload response. The benefits of this approach are that no Payload CMS needs to be running, no real database is used, no real API calls happen, and all test data comes from the mocked GraphQL call.

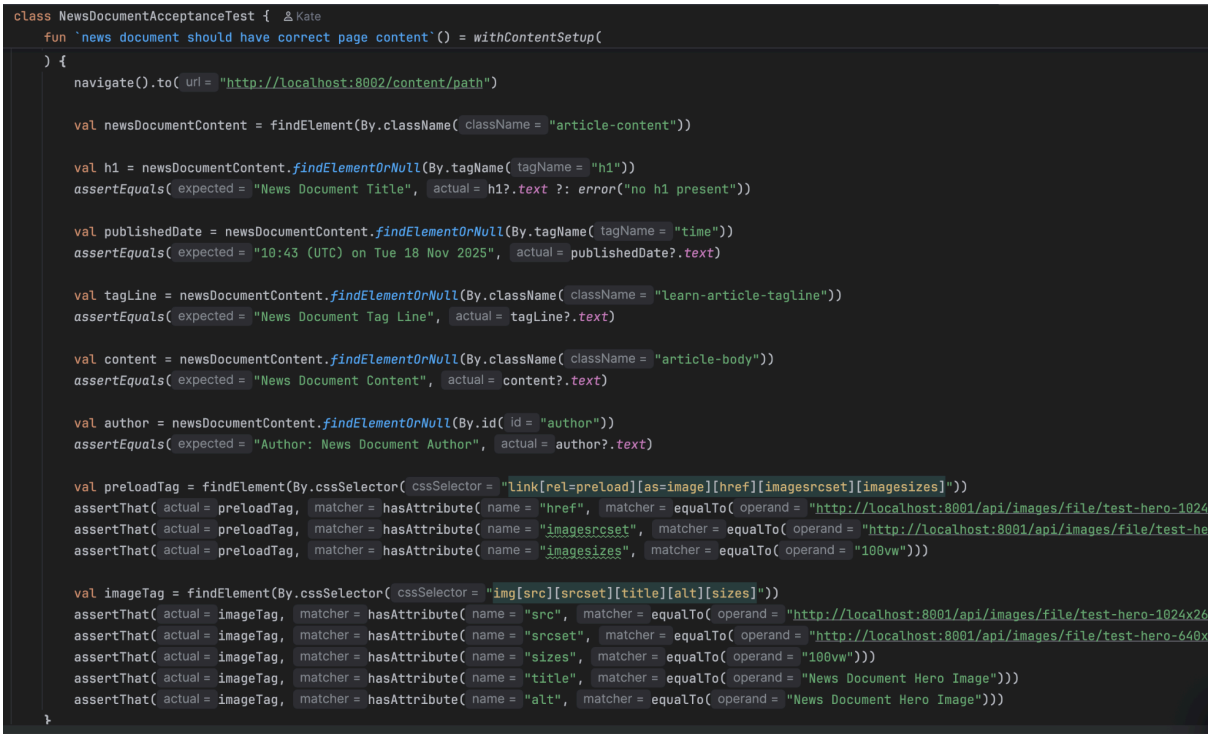
It is considered an acceptance test because it validates real browser behaviour, testing the full HTML, DOM, CSS selectors, and simulated user interactions. However, it is not a full E2E test, as the database and backend are mocked rather than real.



```
package uk.gov.metoffice.digital.acceptance.content.tests

import ...

class NewsDocumentAcceptanceTest {  Kate
    @Test  Kate
    fun `news document should have correct title and description and canonical url`() = withContentSetup(
        mockHandler = {
            when (it.uri.path) {
                "/api/graphql" -> {
                    val newsDocumentResponse = PayloadNewsDocumentResponse( title = "News Document Title", publishedDate = "2025-11-18T10:43:03.911Z", description = "News Document Des
                    Response( status = OK).body( body = payloadResponseBody( articleResponse = newsDocumentResponse))
                }

                else -> Response( status = NOT_FOUND)
            }
        },
    ) {
        navigate().to( url = "http://localhost:8002/content/path")

        assertEquals( expected = "News Document Title - Met Office", actual = title)

        val descriptionTag = findElement(By.cssSelector( cssSelector = "meta[name=description]"))
        assertThat( actual = descriptionTag, matcher = hasAttribute( name = "content", matcher = equalTo( operand = "News Document Description")))

        val canonicalUrlTag = findElement(By.cssSelector( cssSelector = "link[rel='canonical']"))
        assertThat( actual = canonicalUrlTag, matcher = hasAttribute( name = "href", matcher = equalTo( operand = "http://localhost:8002/path")))
    }
```

```
class NewsDocumentAcceptanceTest {  Kate

    @Test  Kate
    fun `news document should have correct page content`() = withContentSetup(
        mockHandler = {
            when (it.uri.path) {
                "/api/graphql" -> {
                    val newsDocumentResponse = PayloadNewsDocumentResponse(
                        title = "News Document Title",
                        publishedDate = "2025-11-18T10:43:03.911Z",
                        tagLine = "News Document Tag Line",
                        content = Jackson.mapper.readTree(
                            content = """
                            {
                              "root": {
                                "type": "root",
                                "format": "",
                                "indent": 0,
                                "version": 1,
                                "children": [
                                  {
                                    "type": "paragraph",
                                    "format": "",
                                    "indent": 0,
                                    "version": 1,
                                    "children": [
                                      {
                                        "mode": "normal",
                                        "text": "News Document Content",
                                        "type": "text",
                                        "style": "",
                                        "detail": 0,
                                        "format": 0,
                                        "version": 1
```

```
class NewsDocumentAcceptanceTest {  Kate
    fun `news document should have correct page content`() = withContentSetup(
        mockHandler = {
                                      }
                                    ],
                                    "direction": "ltr",
                                    "textStyle": "",
                                    "textFormat": 0
                                  }
                                ],
                                "direction": "ltr"
                              }
                            }
                            """.trimIndent(),
                        ),
                        authorName = "News Document Author",
                        heroImage = PayloadResponseFormat.Images(
                            alt = "News Document Hero Image",
                            sizes = mapOf(
                                "heroXSmall" to Image( url = "/api/images/file/test-hero-640x207.jpg", width = 640),
                                "heroMedium" to Image( url = "/api/images/file/test-hero-1024x263.jpg", width = 1024),
                                "heroXLarge" to Image( url = "/api/images/file/test-hero-1600x300.jpg", width = 1600),
                            ),
                        ),
                    )
                    Response( status = OK).body( body = payloadResponseBody( articleResponse = newsDocumentResponse))
                }

                else -> Response( status = NOT_FOUND)
            }
        },
```



```
class NewsDocumentAcceptanceTest {  Kate
    fun `news document should have correct page content`() = withContentSetup(
    ) {
        navigate().to( url = "http://localhost:8002/content/path")

        val newsDocumentContent = findElement(By.className( className = "article-content"))

        val h1 = newsDocumentContent.findElementOrNull(By.tagName( tagName = "h1"))
        assertEquals( expected = "News Document Title", actual = h1?.text ?: error("no h1 present"))

        val publishedDate = newsDocumentContent.findElementOrNull(By.tagName( tagName = "time"))
        assertEquals( expected = "10:43 (UTC) on Tue 18 Nov 2025", actual = publishedDate?.text)

        val tagLine = newsDocumentContent.findElementOrNull(By.className( className = "learn-article-tagline"))
        assertEquals( expected = "News Document Tag Line", actual = tagLine?.text)

        val content = newsDocumentContent.findElementOrNull(By.className( className = "article-body"))
        assertEquals( expected = "News Document Content", actual = content?.text)

        val author = newsDocumentContent.findElementOrNull(By.id( id = "author"))
        assertEquals( expected = "Author: News Document Author", actual = author?.text)

        val preloadTag = findElement(By.cssSelector( cssSelector = "link[rel=preload][as=image][href][imagesrcset][imagesizes]"))
        assertThat( actual = preloadTag, matcher = hasAttribute( name = "href", matcher = equalTo( operand = "http://localhost:8001/api/images/file/test-hero-1024
        assertThat( actual = preloadTag, matcher = hasAttribute( name = "imagesrcset", matcher = equalTo( operand = "http://localhost:8001/api/images/file/test-he
        assertThat( actual = preloadTag, matcher = hasAttribute( name = "imagesizes", matcher = equalTo( operand = "100vw")))

        val imageTag = findElement(By.cssSelector( cssSelector = "img[src][srcset][title][alt][sizes]"))
        assertThat( actual = imageTag, matcher = hasAttribute( name = "src", matcher = equalTo( operand = "http://localhost:8001/api/images/file/test-hero-1024x26
        assertThat( actual = imageTag, matcher = hasAttribute( name = "srcset", matcher = equalTo( operand = "http://localhost:8001/api/images/file/test-hero-640x
        assertThat( actual = imageTag, matcher = hasAttribute( name = "sizes", matcher = equalTo( operand = "100vw")))
        assertThat( actual = imageTag, matcher = hasAttribute( name = "title", matcher = equalTo( operand = "News Document Hero Image")))
        assertThat( actual = imageTag, matcher = hasAttribute( name = "alt", matcher = equalTo( operand = "News Document Hero Image")))
    }
```

On the parts I have been working on within one of the delivery repositories, there are:

- **Acceptance/E2E Tests:** 26 total tests, 7 test files
- **Unit Tests:** 121 total tests, 12 test files

This gives a ratio of approximately **5:1** in favour of unit tests (121 unit vs 26 acceptance tests), which aligns well with the testing pyramid guidelines (roughly 70% Unit, 20% Integration, 10% E2E). Unit tests form the bulk of the testing suite because they are fast, cheap to run, and easy to maintain. Acceptance/E2E tests are slower and more brittle, so having fewer of them is appropriate.

I would not significantly change this ratio, as it represents a healthy balance. However, it could be beneficial to add more acceptance tests when new features introduce user journeys with a higher likelihood of edge cases. But overall, I think the current distribution is appropriate and does not require major adjustment.