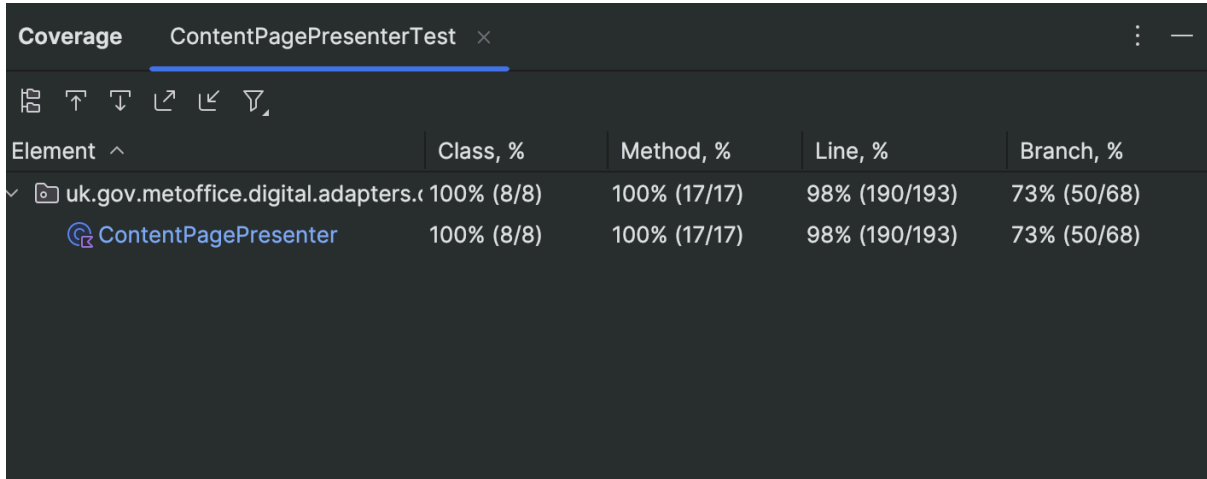


Increasing Code Coverage of Part of my Delivery's Code Base

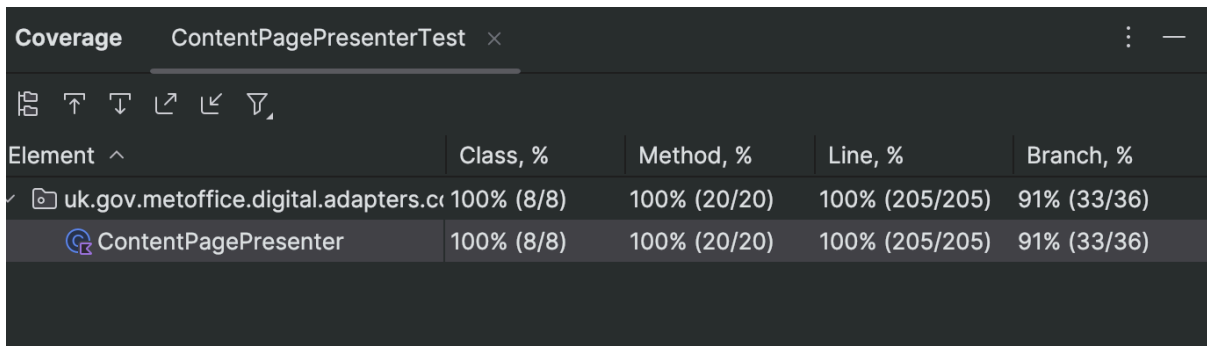
I ran the Kotlin Code Coverage tool on the Content Page Presenter in one of our repositories within my delivery. Initially, the tool reported a code coverage of **73%**:



The screenshot shows the IntelliJ IDEA Coverage tool interface. The title bar indicates 'Coverage' and 'ContentPagePresenterTest'. Below the title bar is a toolbar with icons for coverage actions. The main table displays coverage data for the 'uk.gov.metoffice.digital.adapters.cr' package and the 'ContentPagePresenter' class. The 'ContentPagePresenter' class shows 100% coverage for Class, Method, and Line, but only 73% for Branch.

Element ^	Class, %	Method, %	Line, %	Branch, %
✓ uk.gov.metoffice.digital.adapters.cr	100% (8/8)	100% (17/17)	98% (190/193)	73% (50/68)
ContentPagePresenter	100% (8/8)	100% (17/17)	98% (190/193)	73% (50/68)

To improve this, I refactored the code and added additional tests targeting areas that were previously untested. After these changes, the overall coverage increased to **91%**:



The screenshot shows the IntelliJ IDEA Coverage tool interface after refactoring and adding tests. The title bar indicates 'Coverage' and 'ContentPagePresenterTest'. The main table displays coverage data for the 'uk.gov.metoffice.digital.adapters.cr' package and the 'ContentPagePresenter' class. The 'ContentPagePresenter' class now shows 100% coverage for Class, Method, and Line, and 91% for Branch.

Element ^	Class, %	Method, %	Line, %	Branch, %
✓ uk.gov.metoffice.digital.adapters.cr	100% (8/8)	100% (20/20)	100% (205/205)	91% (33/36)
ContentPagePresenter	100% (8/8)	100% (20/20)	100% (205/205)	91% (33/36)

The Code Coverage tool was particularly useful because it highlighted parts of the code that were not being tested. This allowed me to focus my efforts on creating tests that would have the most impact on overall coverage.

Here is an example of part of the code that was initially highlighted as not being covered:



```
content = renderSafeHtml( node = result.content ),
authors = when (result.authors.size) {
    0 -> null
    1 -> Pair( first = "Author", second = result.authors.first() )
    else -> Pair( first = "Authors", second = result.authors.joinToString( separator = ", " ) )
},
),
headlineTitle = title
```

I then added tests specifically targeting this code:

```
class ContentPagePresenterTest { @ Kate +1 *
    fun `no authors results in null`() = runBlocking {
        templateCompiler,
        config = object : EmptyConfig() {
            override val payloadConfig = PayloadConfig(
                host = "https://www.payload.com",
                assetHost = "https://www.hero-images.com",
            )
        },
    ) { Response( status = NOT_IMPLEMENTED) }

    presenter.present(
        result = Found(
            url = "https://www.metoffice.gov.uk/path",
            content = BlogPost(
                title = "Blog Post Title",
                publishedDate = "2025-11-25T16:25:20.204Z",
                description = "Blog Post Description",
                introduction = "Blog Post Introduction",
                content = Root( children = listOf(Text( text = "Blog Post Content"))),
                authors = emptyList(),
                heroImageTitle = null,
                heroImageSizes = emptyMap(),
            ),
            HeroImages(
                title = "Blog Post Hero Image Title",
                preferred = HeroImage( size = "200w", path = "/img2"),
                sizes = listOf(
                    HeroImage( size = "100w", path = "/img1"),
                    HeroImage( size = "200w", path = "/img2"),
                    HeroImage( size = "300w", path = "/img3"),
                ),
            ),
        ),
    ),
)

presenter.present( menus = emptyMap())

val body = presenter.response().bodyString()
assertTrue( actual = !body.contains( other = "id=\"authors\""))
}
```

```

class ContentPagePresenterTest { @Kate +1
    fun `single author results in Author label and name in blog posts`() = runBlocking {
        val presenter = ContentPagePresenter(
            templateCompiler,
            config = object : EmptyConfig() {
                override val payloadConfig = PayloadConfig(
                    host = "https://www.payload.com",
                    assetHost = "https://www.hero-images.com",
                )
            },
        ) {
            Response( status = NOT_IMPLEMENTED)
        }

        presenter.present(
            result = Found(
                uri = "https://www.metoffice.gov.uk/path",
                content = BlogPost(
                    title = "Blog Post Title",
                    publishedDate = "2025-11-25T16:25:20.204Z",
                    description = "Blog Post Description",
                    introduction = "Blog Post Introduction",
                    content = Root( children = listOf(Text( text = "Blog Post Content"))),
                    authors = listOf("Blog Post Author"),
                    heroImageTitle = null,
                    heroImageSizes = emptyMap(),
                ),
                HeroImages(
                    title = "Blog Post Hero Image Title",
                    preferred = HeroImage( size = "200w", path = "/img2"),
                    sizes = listOf(
                        HeroImage( size = "100w", path = "/img1"),
                        HeroImage( size = "200w", path = "/img2"),
                        HeroImage( size = "300w", path = "/img3"),
                    ),
                ),
            ),
        )

        presenter.present( menus = emptyMap())

        val body = presenter.response().bodyString()
        assertContains( charSequence = body, other = """<p><strong>Author:</strong> Blog Post Author</p>""")
    }
}

```

```

class ContentPagePresenterTest { @Kate +1
    fun `multiple authors result in Authors label and comma separated names`() = runBlocking {
        val presenter = ContentPagePresenter(
            templateCompiler,
            config = object : EmptyConfig() {
                override val payloadConfig = PayloadConfig(
                    host = "https://www.payload.com",
                    assetHost = "https://www.hero-images.com",
                )
            },
        ) {
            Response(status = NOT_IMPLEMENTED)
        }
        presenter.present(
            result = Found(
                url = "https://www.metoffice.gov.uk/path",
                content = BlogPost(
                    title = "Blog Post Title",
                    publishedDate = "2025-11-25T16:25:20.204Z",
                    description = "Blog Post Description",
                    introduction = "Blog Post Introduction",
                    content = Root(children = listOf(Text(text = "Blog Post Content"))),
                    authors = listOf("Blog Post Author", "Other Blog Post Author"),
                    heroImageTitle = null,
                    heroImageSizes = emptyMap(),
                ),
                HeroImages(
                    title = "Blog Post Hero Image Title",
                    preferred = HeroImage(size = "200w", path = "/img2"),
                    sizes = listOf(
                        HeroImage(size = "100w", path = "/img1"),
                        HeroImage(size = "200w", path = "/img2"),
                        HeroImage(size = "300w", path = "/img3"),
                    ),
                ),
            ),
        )
        presenter.present(menus = emptyMap())

        val body = presenter.response().bodyString()
        assertContains(charSequence = body, other = ""<p><strong>Authors:</strong> Blog Post Author, Other Blog Post Author</p>""")
    }
}

```

Additionally, I refactored some of the existing code to create helper functions, which made it easier to write clear and maintainable tests:

```
private fun renderSafeHTML(node: Node?): Handlebars.SafeString? = node?.let { Handlebars.SafeString( content = textModelPresenter.html( node = it)) } 6 Usages & Kate

private fun mergeChildDocuments( 1 Usage & Kate
    childDocuments: List<ChildDocument>?,
    externalChildDocuments: List<ChildDocument>?,
    assetHost: String,
): List<PreviewBlock> = ((childDocuments ?: emptyList()) + (externalChildDocuments ?: emptyList()))
    .map { PreviewBlock.fromChildDocument( childDocument = it, assetHost) }

private fun mergeRelatedLinks( 2 Usages & Kate
    internalLinks: List<RelatedLink>?,
    externalLinks: List<RelatedLink>?,
): List<RelatedLink> = (internalLinks ?: emptyList()) + (externalLinks ?: emptyList())

private fun getHeroImageProperties(heroImages: Found.HeroImages?) = if (heroImages == null) { 4 Usages & Kate
    Triple( first = null, second = null, third = null)
} else {
    Triple(
        first = heroImages.title,
        second = heroImages.run { preferred.let { "$assetHost${it.path}" } },
        third = heroImages.run { sizes.joinToString( separator = ", " ) { "$assetHost${it.path} ${it.size}" } },
    )
}
```

Here are a few examples of the tests I created (note that there were additional tests beyond these examples):

```
class ContentPagePresenterTest { @ Kate +1 *

    @Test @ Kate
    fun `renders introduction and content when both are present`() = runBlocking {
        val presenter = ContentPagePresenter(
            templateCompiler,
            config = object : EmptyConfig() {
                override val payloadConfig = PayloadConfig(
                    host = "https://www.payload.com",
                    assetHost = "https://www.hero-images.com",
                )
            },
        ) {
            Response( status = NOT_IMPLEMENTED)
        }

        presenter.present(
            result = Found(
                uri = "https://www.metoffice.gov.uk/path",
                content = Article(
                    title = "Article Title",
                    description = "Article Description",
                    tagLine = "Article TagLine",
                    introduction = Root( children = listOf(Text( text = "Article Introduction"))),
                    content = Root( children = listOf(Text( text = "Article Content"))),
                    relatedInternalLinks = null,
                    relatedExternalLinks = null,
                    heroImageTitle = null,
                    heroImageSizes = emptyMap(),
                ),
                heroImages = null,
            ),
        )

        presenter.present( menus = emptyMap())

        val body = presenter.response().bodyString()

        assertContains( charSequence = body, other = "<div>Article Introduction</div>")
        assertContains( charSequence = body, other = "<div>Article Content</div>")
    }
}
```

```

class ContentPagePresenterTest { @Kate +1 *

    @Test @Kate
    fun `renders content when introduction is null`() = runBlocking {
        val presenter = ContentPagePresenter(
            templateCompiler,
            config = object : EmptyConfig() {
                override val payloadConfig = PayloadConfig(
                    host = "https://www.payload.com",
                    assetHost = "https://www.hero-images.com",
                )
            },
        ) {
            Response(status = NOT_IMPLEMENTED)
        }

        presenter.present(
            result = Found(
                uri = "https://www.metoffice.gov.uk/path",
                content = Article(
                    title = "Article Title",
                    description = "Article Description",
                    tagLine = "Article TagLine",
                    introduction = null,
                    content = Root(children = listOf(Text(text = "Article Content"))),
                    relatedInternalLinks = null,
                    relatedExternalLinks = null,
                    heroImageTitle = null,
                    heroImageSizes = emptyMap(),
                ),
                heroImages = null,
            ),
        )

        presenter.present(menus = emptyMap())

        val body = presenter.response().bodyString()

        assertFalse(actual = body.contains(other = "<div>Article Introduction</div>"))
        assertContains(charSequence = body, other = "<div>Article Content</div>")
    }
}

```

Overall, this process allowed me to improve the reliability of the Content Page Presenter by ensuring more of the code was tested and the overall code coverage was higher. It also gave me the opportunity to refactor the code base to make it less DRY.