

Narzędzie do tworzenia testów *e2e* dla widżetów

Cel testu

- **forWidget** - definiuje, dla jakiego całego widżetu ma zostać wykonany test
- **forComponent** - definiuje, dla jakiego elementu z atrybutem `data-component` (komponent stworzony w ramach projektu) oraz w jakim widżecie ma zostać wykonany test
- **forElement** - definiuje, dla jakiego elementu z atrybutem `data-testid` oraz w jakim widżecie ma zostać wykonany test

Typ testu

- **forClients** - dla jakiego klienta
- **forViewports** - na jaką rozdzielczość
- **forColorSchemes** - na jaki schemat kolorystyczny

Konfiguracja środowiska/wykonania testu

- **withWidgetProps** - parametry widżetu
- **withRouteMock** - konfiguracja odpowiedzi zapytań API
- **withWaitFor** - oczekiwanie testu na szczególne typy wydarzeń (np. załadowanie wykresu)
- **setPageInteraction** - interakcja z widżetem z pomocą obiektu `page playwrighta`

Pozostałe

- **only** - wykonanie wyłącznie testu z danego pliku
- **withTitle** - ustawienie przedrostka do nazwy testu (reszta jest generowana automatycznie)

Przykładowe nazwy wygenerowanych obrazów

Dla widżetu `widget1`:

- `widget1-client1-desktop-chromium.png`
- `widget1-client1-desktop-firefox.png`
- `widget1-client1-desktop-loading-chromium.png`
- `widget1-client1-desktop-loading-firefox.png`
- `widget1-client1-desktop-loading-webkit.png`
- `widget1-client1-desktop-no-data-chromium.png`
- `widget1-client1-desktop-no-data-firefox.png`
- `widget1-client1-desktop-no-data-webkit.png`
- `widget1-client1-desktop-webkit.png`
- `widget1-client1-mobile-chromium.png`
- `widget1-client1-mobile-firefox.png`
- `widget1-client1-mobile-loading-chromium.png`
- `widget1-client1-mobile-loading-firefox.png`
- `widget1-client1-mobile-loading-webkit.png`

- widget1-client1-mobile-no-data-chromium.png
- widget1-client1-mobile-no-data-firefox.png
- widget1-client1-mobile-no-data-webkit.png
- widget1-v-mobile-webkit.png

Dla komponentu `component1`:

- component1-client1-desktop-variant2-chromium.png
- component1-client1-desktop-variant2-firefox.png
- component1-client1-desktop-variant2-webkit.png
- component1-client1-desktop-variant1-chromium.png
- component1-client1-desktop-variant1-firefox.png
- component1-client1-desktop-variant1-webkit.png
- component1-client1-mobile-variant2-chromium.png
- component1-client1-mobile-variant2-firefox.png
- component1-client1-mobile-variant2-webkit.png
- component1-client1-mobile-variant1-chromium.png
- component1-client1-mobile-variant1-firefox.png
- component1-client1-mobile-variant1-webkit.png
- component1-client2-desktop-variant2-chromium.png
- component1-client2-desktop-variant2-firefox.png
- component1-client2-desktop-variant2-webkit.png
- component1-client2-desktop-variant1-chromium.png
- component1-client2-desktop-variant1-firefox.png
- component1-client2-desktop-variant1-webkit.png
- component1-client2-mobile-variant2-chromium.png
- component1-client2-mobile-variant2-firefox.png
- component1-client2-mobile-variant2-webkit.png
- component1-client2-mobile-variant1-chromium.png
- component1-client2-mobile-variant1-firefox.png
- component1-client2-mobile-variant1-webkit.png

Dla elementu `element1` w widżecie `widget2`:

- widget2-client1-desktop-element1-chromium.png
- widget2-client1-desktop-element1-firefox.png
- widget2-client1-desktop-element1-webkit.png
- widget2-client1-mobile-element1-chromium.png
- widget2-client1-mobile-element1-firefox.png
- widget2-client1-mobile-element1-webkit.png

Kod programu

```
1 import { test as t, expect } from "@playwright/test";
2
3 const DEFAULT_API_URL = "https://widgets.api.pl/test/api/equities/widgets";
4
5 /**
6  * @typedef {Object} ViewPortResolution
7  * @property {number} width
8  * @property {number} height
9  */
10
11 class Builder {
12   /** @type {string|null} */
13   #widgetId = null;
14
15   /** @type {string[]} */
16   #clients = ["default_client"];
17
18   /** @type {('desktop' | 'mobile')[]} */
19   #viewport = ["desktop", "mobile"];
20
21   /** @type {('default' | 'dark')[]} */
22   #colorSchemes = ["default"];
23
24   /** @type {string|null} */
25   #componentName = null;
26
27   /** @type {Object|null} */
28   #widgetProps = null;
29
30   /** @type {Array<Object>} */
31   #apiMocks = [];
32
33   /** @type {function|null} */
34   #pageInteraction = null;
35
36   /** @type {boolean} */
37   #onlyThis = false;
38
39   /** @type {Array<"canvas" | "timeout">} */
40   #waitFor = [];
41
42   /** @type {string} */
43   #widgetState = "default";
44
45   /** @type {string|null} */
46   #elementTestId = null;
47
48   /** @type {string|null} */
49   #title = null;
50
51   /** @type {Object<string, ViewPortResolution>} */
52   #viewPortResolution = {
53     desktop: {
54       width: 1396,
55       height: 480,
56     },
57     mobile: {
58       width: 600,
59       height: 480,
60     },
61   };
62
63   /**
64    * @param {string} widgetId
65    * @returns {this}
66    */
67   forWidget(widgetId) {
68     this.#widgetId = widgetId;
69     this.#elementTestId = null;
70     return this;
71   }
```

```

72
73 /**
74  * @param {string} componentName
75  * @param {string} widgetId
76  * @returns {this}
77  */
78 forComponent(componentName, widgetId) {
79   this.#componentName = componentName;
80   this.#widgetId = widgetId;
81   return this;
82 }
83
84 /**
85  * @param {string} elementTestId
86  * @returns {this}
87  */
88 forElement(elementTestId) {
89   this.#elementTestId = elementTestId;
90   return this;
91 }
92
93 /**
94  * @returns {this}
95  */
96 only() {
97   this.#onlyThis = true;
98   return this;
99 }
100
101 /**
102  * @param {string[]} clients
103  * @returns {this}
104  */
105 forClients(clients) {
106   this.#clients = clients;
107   return this;
108 }
109
110 /**
111  * @param {('desktop' | 'mobile')[]} viewport
112  * @returns {this}
113  */
114 forViewports(viewport) {
115   this.#viewport = viewport;
116   return this;
117 }
118
119 /**
120  * @param {('default' | 'dark')[]} colorSchemes
121  * @returns {this}
122  */
123 forColorSchemes(colorSchemes) {
124   this.#colorSchemes = colorSchemes;
125   return this;
126 }
127
128 /**
129  * @param {Object|function(Object):Object} newPropsOrCallback
130  * @returns {this}
131  */
132 withWidgetProps(newPropsOrCallback) {
133   if (typeof newPropsOrCallback === "function") {
134     this.#widgetProps = newPropsOrCallback(this.#widgetProps);
135   } else {
136     this.#widgetProps = newPropsOrCallback;
137   }
138   return this;
139 }
140
141 /**
142  * @param {string} affix
143  * @param {Object|string} data
144  * @param {string} contentType

```

```

145 * @returns {this}
146 */
147 withRouteMock(affix, data, contentType) {
148   this.#apiMocks = this.#apiMocks.filter((mock) => mock.endpoint !== affix);
149
150   this.#apiMocks = [
151     ...this.#apiMocks,
152     {
153       endpoint: affix,
154       mockData: data,
155       contentType: contentType,
156     },
157   ];
158   return this;
159 }
160
161 /**
162  * @param {Array<"canvas" | "timeout">} waitFor
163  * @returns {this}
164  */
165 withWaitFor(waitFor) {
166   this.#waitFor = waitFor;
167   return this;
168 }
169
170 /**
171  * @param {function(import('playwright').Page):Promise<void>} [pageInteraction]
172  * @returns {this}
173  */
174 setPageInteraction(pageInteraction) {
175   this.#pageInteraction = pageInteraction;
176   return this;
177 }
178
179 /**
180  * Sets the widget state for the test.
181  * @param {'no-data' | 'loading' | 'no-response' | 'default'} widgetState
182  * @returns {this}
183  */
184 setWidgetState(widgetState) {
185   this.#widgetState = widgetState;
186   return this;
187 }
188
189 /**
190  * Sets title prefix for the test.
191  * @param {string} title
192  * @returns {this}
193  */
194 withTitle(title) {
195   this.#title = title;
196   return this;
197 }
198
199 /**
200  * Runs the test with the given variant name.
201  * @param {string} [variantName]
202  * @returns {this}
203  */
204 test(variantName) {
205   if (!this.#widgetId) throw new Error("Widget ID is not set");
206   const playwrightTest = this.#onlyThis ? t.only : t;
207
208   const testState = {
209     widgetProps: this.#widgetProps,
210     pageInteraction: this.#pageInteraction,
211     widgetState: this.#widgetState,
212     elementTestId: this.#elementTestId,
213     waitFor: this.#waitFor,
214   };
215
216   for (const client of this.#clients) {
217     for (const viewport of this.#viewport) {

```

```

218     for (const colorScheme of this.#colorSchemes) {
219         playwrightTest(
220             this.#getTestDescriptionFor(
221                 client,
222                 viewport,
223                 colorScheme,
224                 testState.widgetState,
225                 variantName,
226                 testState.elementTestId,
227                 this.#title
228             ),
229             async ({ page }) => {
230                 await this.#mockCurrentDate(page);
231
232                 await this.#mockApiCall(page, testState.widgetState);
233
234                 await this.#setViewportFor(viewport, page);
235
236                 await this.#addWidgetToPage(
237                     client,
238                     page,
239                     testState.widgetProps,
240                     colorScheme
241                 );
242                 await this.#loadPage(page, colorScheme, testState.waitFor);
243
244                 if (testState.pageInteraction) {
245                     await testState.pageInteraction(page);
246                 }
247
248                 expect(
249                     await this.#takeWidgetScreenshot(page, testState.elementTestId)
250                 ).toMatchSnapshot(
251                     this.#getReferenceFileFor([
252                         client,
253                         viewport,
254                         colorScheme !== "default" && colorScheme,
255                         testState.widgetState !== "default" && testState.widgetState,
256                         testState.elementTestId,
257                         variantName,
258                     ])
259                 );
260             }
261         );
262     }
263 }
264 }
265
266 this.#resetState();
267 return this;
268 }
269
270 /**
271  * @private
272  * @param {string} client
273  * @param {string} viewport
274  * @param {string} colorScheme
275  * @param {string} widgetState
276  * @param {string} [variantName]
277  * @param {string|null} elementTestId
278  * @param {string|null} title
279  * @returns {string}
280  */
281 #getTestDescriptionFor(
282     client,
283     viewport,
284     colorScheme,
285     widgetState,
286     variantName,
287     elementTestId,
288     title
289 ) {
290     return [

```

```

291     ${title ? title : ""},
292     this.#componentName || this.#widgetId,
293     for client @${client},
294     in @${viewport} viewport,
295     colorScheme !== "default" && , @${colorScheme} color scheme,
296     widgetState !== "default" && , @${widgetState} state,
297     variantName && , @${variantName} variant,
298     elementTestId && , @${elementTestId} element,
299 ]
300 .filter(Boolean)
301 .join("");
302 }
303
304 /**
305  * @private
306  * @param {import('playwright').Page} page
307  * @returns {Promise<void>}
308  */
309 async #mockCurrentDate(page) {
310     const mockDate = new Date(Date.UTC(2023, 7, 4)).valueOf();
311     await page.addInitScript({
312         Date = class extends Date {
313             constructor(...args) {
314                 if (args.length === 0) {
315                     super(${mockDate});
316                 } else {
317                     super(...args);
318                 }
319             }
320         }
321         const __DateNowOffset = ${mockDate} - Date.now();
322         const __DateNow = Date.now;
323         Date.now = () => __DateNow() + __DateNowOffset;
324     });
325 }
326
327 /**
328  * @private
329  * @param {import('playwright').Page} page
330  * @param {string} widgetState
331  * @returns {Promise<void>}
332  */
333 async #mockApiCall(page, widgetState) {
334     if (widgetState === "no-response") return;
335
336     let mockApi;
337
338     try {
339         mockApi = await import(../../mock-api/${this.#widgetId}.mock);
340     } catch (error) {
341         return;
342     }
343
344     const { mockApiPresets } = mockApi;
345
346     for (const {
347         endpoint,
348         data,
349         contentType,
350         customQuery = "",
351         apiUrl,
352     } of mockApiPresets.e2e[widgetState === "no-data" ? "noData" : "default"]) {
353         await page.route(
354             ${apiUrl || DEFAULT_API_URL}/${endpoint}${customQuery}*,
355             async (route) => {
356                 if (widgetState === "loading") {
357                     return;
358                 } else if (contentType === "text/html") {
359                     await route.fulfill({
360                         contentType: "text/html",
361                         body: data,
362                     });
363                 } else {

```

```

364         await route.fulfill({ json: data });
365     }
366 }
367 );
368 }
369 }
370
371 /**
372  * @private
373  * @param {string} viewport
374  * @param {import('playwright').Page} page
375  * @returns {Promise<void>}
376  */
377 async #setViewportFor(viewport, page) {
378     await page.setViewportSize(this.#viewportResolution[viewport]);
379 }
380
381 /**
382  * @private
383  * @param {string} client
384  * @param {import('playwright').Page} page
385  * @param {Object|null} widgetProps
386  * @param {string} colorScheme
387  * @returns {Promise<void>}
388  */
389 async #addWidgetToPage(client, page, widgetProps, colorScheme) {
390     await page.addInitScript({
391         content:
392             window.widget = ${JSON.stringify({
393                 "widget-id": this.#widgetId,
394                 "no-animations": "1",
395                 "dark-mode": colorScheme === "dark" ? "on" : "off",
396                 ...widgetProps,
397             })};
398             window.CONFIG_CLIENT_ID = ${JSON.stringify(client)};
399     },
400     ));
401 }
402
403 /**
404  * @private
405  * @param {import('playwright').Page} page
406  * @param {string} colorScheme
407  * @param {Array<"canvas" | "timeout">} waitFor
408  * @returns {Promise<void>}
409  */
410 async #loadPage(page, colorScheme, waitFor) {
411     await page.goto("./");
412     if (colorScheme === "dark") {
413         await page.addStyleTag({
414             content:
415                 body {
416                     background-color: #1f2124 !important;
417                 }
418             ,
419         });
420     }
421     await page.evaluate(() => document.fonts.ready);
422     await page.waitForLoadState("load");
423     if (waitFor.includes("canvas")) {
424         await page.waitForSelector("canvas");
425     }
426     if (waitFor.includes("timeout")) {
427         await new Promise((resolve) => setTimeout(resolve, 2000));
428     }
429 }
430
431 /**
432  * @private
433  * @param {import('playwright').Page} page
434  * @param {string|null} elementTestId
435  * @returns {Promise<Buffer>}
436  */

```



```

437   async #takeWidgetScreenshot(page, elementTestId) {
438       const element = await page
439         .locator(
440           this.#componentName
441             ? [data-component=${this.#componentName}]
442             : elementTestId
443             ? [data-testid=${elementTestId}]
444             : [widget-id=${this.#widgetId}]
445         )
446         .first();
447
448       if (elementTestId) {
449         const boundingBox = await element.boundingBox();
450         const padding = 10;
451         const screenshotOptions = {
452           clip: {
453             x: boundingBox.x - padding,
454             y: boundingBox.y - padding,
455             width: boundingBox.width + 2 * padding,
456             height: boundingBox.height + 2 * padding,
457           },
458         };
459
460         return await page.screenshot(screenshotOptions);
461       } else {
462         return await element.screenshot();
463       }
464     }
465
466     /**
467      * @private
468      * @param {Array<string|boolean>} parts
469      * @returns {Array<string>}
470      */
471     #getReferenceFileFor(parts) {
472       return [
473         this.#componentName || this.#widgetId,
474         [this.#componentName || this.#widgetId, ...parts.filter(Boolean)].join(
475           "_"
476         ),
477       ];
478     }
479
480     /**
481      * Resets the state of the builder.
482      * @private
483      */
484     #resetState() {
485       this.#widgetState = "default";
486     }
487   }
488
489   export default Builder;
490
491   // STB screenshot test builder v. 1.0

```