Reliability Testing for Meta-Frameworks

LENNART JÖRGENS

Front-End Test Fest
June 7, 2023

Lennart Jörgens

Core maintainer of Gatsby

Working at Netlify

lekoarts.de - @lekoarts

Table of contents

Why?

“I can rebuild this framework in a weekend”

What could possibly go wrong?

- Inconsistencies between operating systems & browsers
- Browser extensions like Ad-Blockers
- Flaky, slow internet
- Bugs in your framework 🐞

Building a reliable framework is hard

But you can automate a lot of testing

Foundations

Foundations

- Unit test core functionalities
- Build out test utilities and patterns
- Run your tests on all supported versions and platforms

Enjoy your testing setup

Operating Systems

Different path separators:

```
# Linux
/some/path

# Windows
\\some\\path
```

You can use libraries like pathe for path normalization.

ESM loader:

```
/* Linux */
await import(somePath)

/* Windows */
await import(pathToFileURL(somePath).href)
```

Operating Systems

```yaml
jobs:
  unit-tests:
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest]
    runs-on: ${{ matrix.os }}
    steps:
    - run: 'Your tests'
```

Node.js versions

```yaml
jobs:
  build:
    strategy:
      matrix:
        node: [16, 18, 20]
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Setup node
        uses: actions/setup-node@v3
        with:
          node-version: ${{ matrix.node }}
      - run: 'Your tests'
```

Different browsers

```yaml
jobs:
  build:
    strategy:
      matrix:
        browser: [chrome, firefox]
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: cypress-io/github-action@v5
        with:
          browser: ${{ matrix.browser }}
```

Never assume things behave the same everywhere &
test your core logic extensively

Configurations

Configurations

- Behavior might depend on:
  - `` `NODE_ENV` ``
  - Environment variables
  - JS runtime
- Different ways of defining a configuration
- Handling deprecated/legacy APIs

Testing environment variables

```
describe('Testing TRAILING_SLASH env var', () => {
  beforeAll(() => {
    process.env.TRAILING_SLASH = 'true'
  })

  it('should add trailing slash to url', () => {#
    expect(modifyUrl('http://example.com')).toBe('http://example.com/')
  })

  afterAll(() => {
    delete process.env.TRAILING_SLASH
  })
})
```

How would you test all variations in an E2E test suite?

```
type Config = {
  trailingSlash: 'always' | 'never'
}

const config: Config =  {
  trailingSlash: 'always',
}

export default config
```

The shell is your friend

```
const trailingSlash = process.env.TRAILING_SLASH || 'always'

const config: Config =  {
  trailingSlash,
}

export default config
```

Set the environment variable inside the scripts you run:

```
{
  "scripts": {
    "test": "npm-run-all -c -s test:always test:never",
    "test:always": "cross-env OPTION=always npm run test-script",
    "test:never": "cross-env OPTION=never npm run test-script",
    "//": "Other scripts...",
    "cy:config": "cross-env-shell cypress run --config-file \"cypress/configs/$OPTION.ts\"",
    "build:opt": "cross-env-shell TRAILING_SLASH=$OPTION npm run build",
  }
}
```

Add test coverage for each new feature & programatically run your tests e.g. via shell scripts

Expected behaviors

Expected behaviors

- Client-side navigation. Did it reload?
- ``console.log`` information
- Hot Module Replacement (HMR)
- Accessibility checks (a11y)
- Built-in components
- Generated artifacts

Did it reload?

```
describe('Client-side navigation', () => {
  it('did not reload', () => {
    cy.visit('/')

    cy.window().then(win => {
      win.__didNotReload = true
    })

    cy.findByText('Page 2').click()

    cy.window().its('__didNotReload').should('equal', true)
  })
})
```

`console.log` information

```
Cypress.Commands.overwrite('visit', (orig, url, options = {}) => {
  const newOptions = {
    ...options,
    onBeforeLoad: win => {
      if (options.onBeforeLoad) {
        options.onBeforeLoad(win)
      }

      cy.spy(win.console, 'log').as('hmrConsoleLog')
    },
  }

  return orig(url, newOptions)
})

Cypress.Commands.add('waitForHmr', (message = 'App is up to date') => {
  cy.get('@hmrConsoleLog').should('be.calledWithMatch', message)
  cy.wait(1000)
})
```

Hot Module Replacement

React component:

```
export default function Title() {
  return <h1 data-testid="title">{'%TITLE%'}</h1>
}
```

Test:

```
describe('HMR: React components', () => {
  it('updates on change', () => {
    const text = `Hello World`
    cy.exec(
      `npm run update -- --file src/components/title.tsx --replacements "TITLE:${text}"`
    )

    cy.waitForHmr()
    cy.findByTestId('title').should('have.text', text)
  })
})
```

Example `update` script

Accessibility checks

There are many great talks and articles about this! Please watch/read those to learn more.

For automated testing I can recommend:

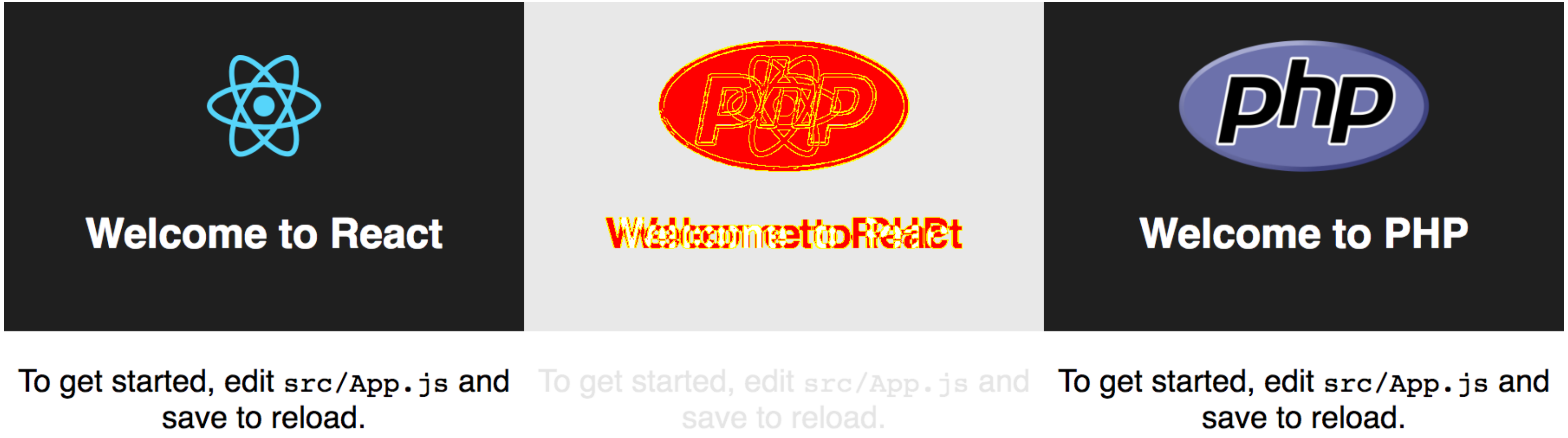
- cypress-axe
- cypress-real-events

But automated testing should only be a **part** of your a11y strategy.

Built-in components

Use a tool like @simonsmith/cypress-image-snapshot to visually test components.

Example test suite



Welcome to React

To get started, edit `src/App.js` and save to reload.

Welcome to PHP

To get started, edit `src/App.js` and save to reload.

Generated artifacts

```
describe(`Webpack Assets`, () => {
  beforeEach(() => {
    cy.intercept("/static/font-name-**.woff2").as("font-regular")
    cy.intercept("/image-file.png").as("static-folder-image")
    cy.visit(`/assets`)
  })

  it(`should create font file`, () => {
    cy.wait("@font-regular").should(req => {
      expect(req.response.url).to.match(/font-name-/i)
    })
  })

  it(`should load static folder asset`, () => {
    cy.wait("@static-folder-image").should(req => {
      expect(req.response.statusCode).to.be.gte(200).and.lt(400)
    })
  })
})
```

Spend time learning advanced features of your testing tool & mimic your users’ behavior

Unknowns 👻

Unknowns 👻

- Ad-Blocker
- Deploys between page navigations
- Internet speed
- Bots

Missing/Blocked page resources

- Ad-Blockers can arbitrarily block deployed files e.g. because they contain banned words
- Visitors can browse the website while a new deploy happened in the background
- Deployment of files might be misbehaving

These are all edge-cases but you can still make your framework resilient against those things.

```
/* --- Cypress Configuration --- */

import { defineConfig } from "cypress"
// Utilties later used on "task"
import { blockResourcesUtils } from "./cypress/utils/block-resources"

export default defineConfig({
  e2e: {
    setupNodeEvents(on) {
      on(`task`, {
        ...blockResourcesUtils
      })
    },
  },
})

/* --- Test File --- */

const runBlockedScenario = ({ filter, pagePath }) => {
  beforeEach(() => {
    // "getAssetsForPage" is our own utility
    cy.task("getAssetsForPage", { pagePath, filter }).then(urls => {
      for (const url of urls) {
        cy.intercept(url, {
          statusCode: 404,
          body: "",
```

Internet speed

Your front-end can have checks like these:

```
const isSlow = () => {
  if ('connection' in navigator && typeof navigator.connection !== 'undefined') {
    if ((navigator.connection.effectiveType || '').includes('2g')) {
      return true
    }
    if (navigator.connection.saveData) {
      return true
    }
  }
  return false
}

class Loader {
  shouldPrefetch(pagePath) {
    if (isSlow()) {
      return false
    }

    return true
  }
}
```

Internet speed

```
Cypress.Commands.add('visitWithType', (url, effectiveType) => {
  cy.visit(url, {
    onBeforeLoad(win) {
      const connection = {
        effectiveType,
        addEventListener: () => {},
      }
      cy.stub(win.navigator, 'connection', connection);
    },
  })
})

describe('Loading indicator', () => {
  beforeEach(() => {
    cy.visitWithType('/', '2g')
  })

  it('shown on 2G speed', () => {
    cy.findByTestId('loading-indicator').should('be.visible')
  })
})
```

Bots

Similarly you can also check for bots:

```
const BOT_REGEX = /bot|crawler|spider|crawling/i

class Loader {
  shouldPrefetch(pagePath) {
    if (navigator.userAgent && BOT_REGEX.test(navigator.userAgent)) {
      return false
    }

    return true
  }
}
```

Bots

```
{
  "scripts": {
    "cy:run:bot": "CYPRESS_CONNECTION_TYPE=bot cypress run",
  }
}
```

```
{
  setupNodeEvents(on) {
    on('before:browser:launch', (browser = {}, opts) => {
      if (process.env.CYPRESS_CONNECTION_TYPE === `bot`) {
        opts.args.push('--user-agent="Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"')
      }

      return opts
    })
  },
},
```

SUMMARY

- Test your core logic on different platforms & versions
- Consider your complete API surface when testing features, including environment variables
- Read up on advanced features like network request interception, stubbing, script execution to simulate and test behaviors
- Use community packages for things like a11y and snapshot testing
- Leverage bug reports to add tests for edge-cases

Thank You!

Slides on lekoarts.de