TESTING

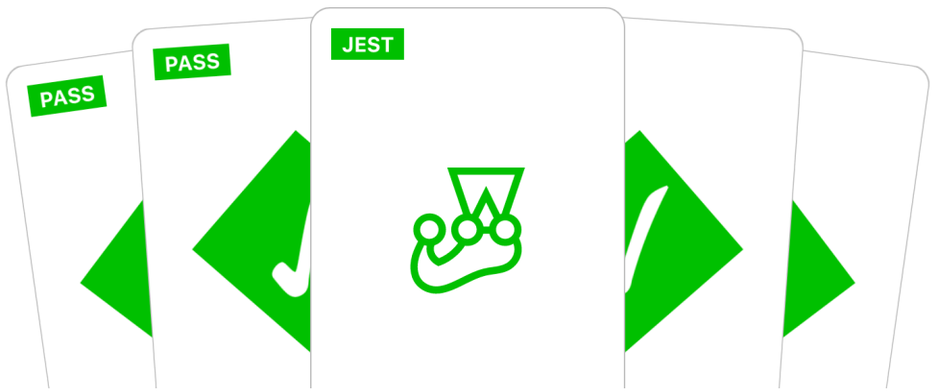
REACT APPS

Bettina Helgenlechner
hallo@bettina.tech
chemmedia AG
2019-02-28

Speaking about my experience testing React apps, but a lot of the tools and ideas work with most if not any framework.

(Some) Things to Test

- Business logic: unit tests
- Presentation(al logic): component tests
- Features and workflows: end-to-end tests

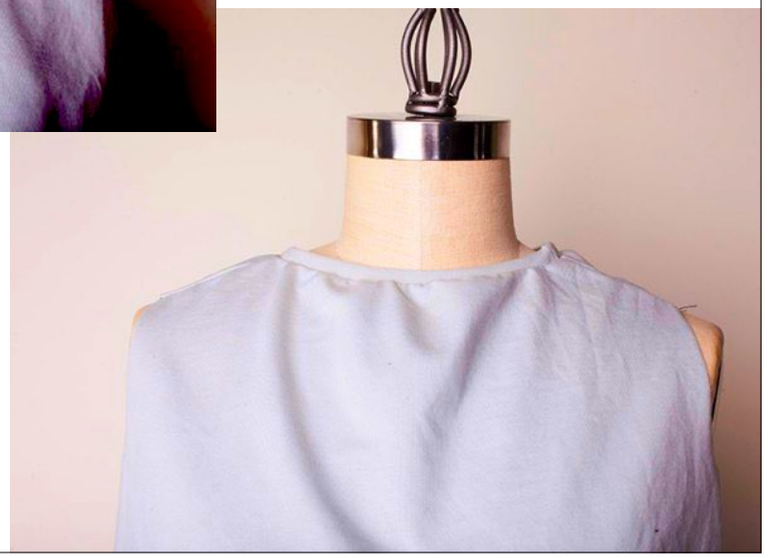


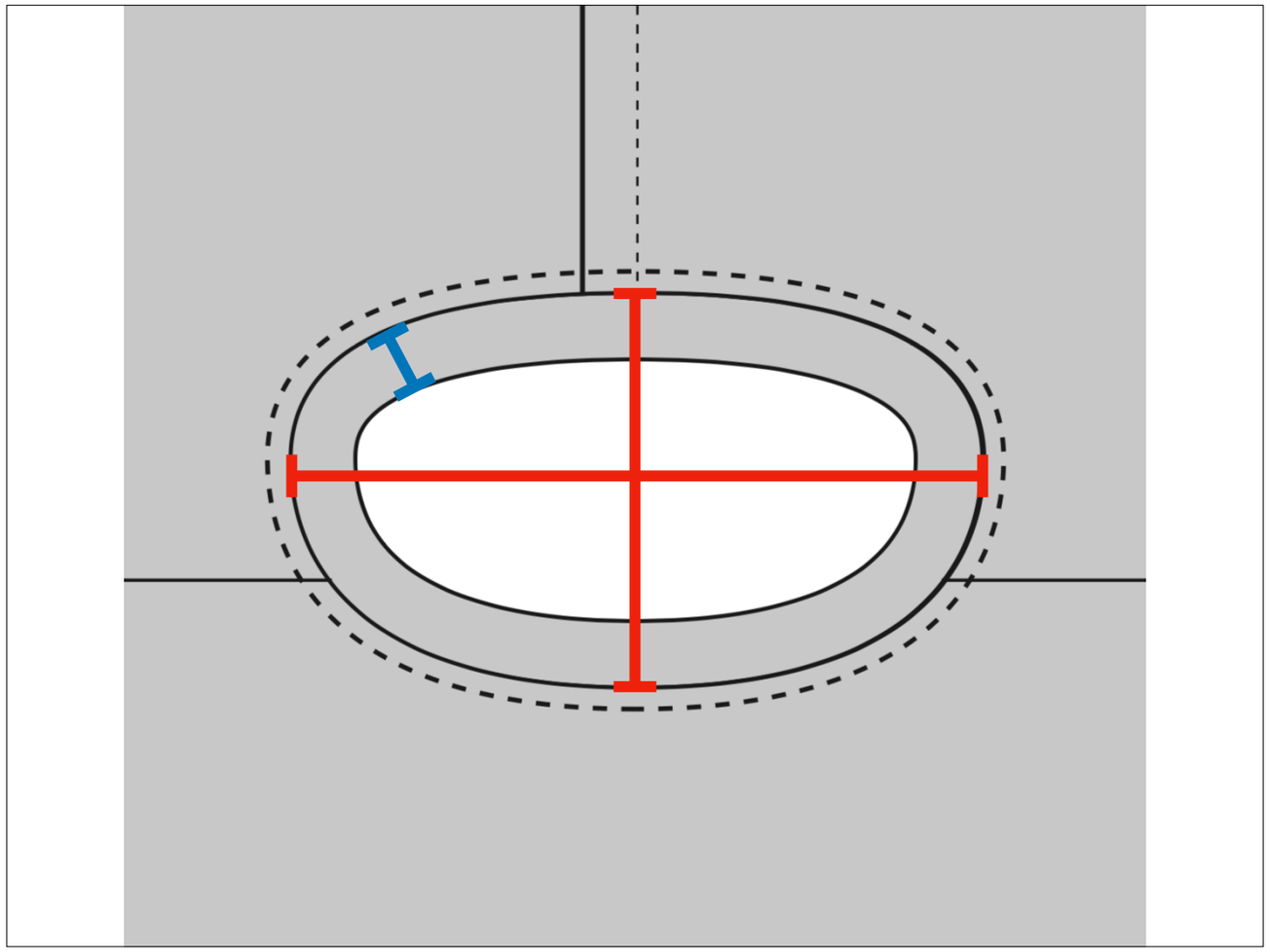
Unit Testing With Jest

- Test logic and components
- Comes standard with `create-react-app`
- Node, not browser
- `describe()`, `it()`, `expect()`
- Entire testing toolkit in one place, including coverage reports

- Jest is a JavaScript testing framework
- Designed to ensure correctness of any JavaScript codebase.
- Write tests with an approachable, familiar and feature-rich API.
- Well-documented, requires little configuration and can be extended to match your requirements.
- Used to test logic and components in isolation
- Included when creating app with CRA, but should work with any framework
- Runs in Node environment, not browser => fast and stable; window etc. are mocked
- By default, tests run in parallel => isolated and better performance

- Show idea/problem, then DEMO: caclulateLength.test.ts





1. Set up the application state.
2. Take an action.
3. Make an assertion about the resulting application state.

- Keep all setup etc. contained within the test
- Each test needs to be able to pass independently
- Keep test files next to units being tested
- Make code testable:
  - As little business logic as possible inside components
  - As many pure functions as possible
  - Inject dependencies
  - Heavy/difficult test setup often indicates bad code design
- Tests need to be part of code review and CI builds

- Keep everything within the test: IMO, DRY does not apply in unit tests. Tests are much easier to understand, if all set up is right within the test, even if it's repeated multiple times
- Independence: Don't couple tests together so that they build on each other.
- Heavy test setup: test setup pain is a bad code smell, means there is a lack of modularity, brittleness

Property-Based Testing

- Usually: example-based testing
- Limitations:
  - small amount of examples is usually checked
  - developer’s bias in writing input-output scenarios
- Solution: property-based testing

- Usually, unit tests are example-based. The unit under test is being called with predefined input and output is compared to predefined expected output
- This has two big limitations: only a small number of inputs are usually tested, the developer has to come up with meaningful scenarios and not miss anything important
- Solution: Use random value generators to generate inputs for the unit being tested and verify the result by deriving invariant properties out of the code (properties that will always be true, no matter the input)

- Hardest part: define properties that are true for all possible valid inputs

- Example: sorting an array of numbers

Can you come up with properties that can be used to see if sorting was done correctly?

- each member is GTE than the one before it
- Input and output have the same length
- Keeps each member of the original array
- Doesn’t add new numbers to the array

=> DEMO: Desktop/property-based

Component Testing

Jest Snapshots

- Ensure that the UI does not change unexpectedly
- NOT screenshots
- Treat snapshots as code
- Tests should be deterministic

- Snapshot tests can be used to ensure the UI of a component does not change unexpectedly
- It does not take a screenshot of the rendered output, but saves the serialized output to a file that is stored alongside the test
- When the test is run again, the new output is compared to the stored output
- Test will fail if the output changes => either fix the component or update the snapshot to the new desired output
- Treat snapshots as code: part of code review and version control, enforce stylistic rules => make it easy to review snapshots
- Tests should be deterministic: if necessary, use mocks to ensure the output is always the same (like mocking Date.now() when it is part of the output)

DEMO:
Snapshot test: result.test.tsx, switch to Inch units to show

Enzyme

- Testing utility used to assert, manipulate and traverse React components' output
- Can be used with various test runners, including Jest
- Capabilities:
  - node traversal,
  - simulate events (click etc.),
  - simulate component errors,
  - trigger lifecycle events,
  - change props, state and context

- Similar to snapshot testing with Jest itself, but the component can be manipulated

DEMO: ShapeSelector

Storybook

- UI development and testing environment
- Render and interact with components in isolation
- Write stories describing various use cases of components
- Ecosystem of add-ons to extend functionality

DEMO: storybook in the browser

- Many available add-ons:
  - accessibility standards compliance
  - links for interactive prototypes
  - viewports
  - router integration
  - props combinations
  - open API to write your own add-ons

Automated Testing with Storybook

- Structural Testing: Storyshots (Jest Snapshots inside Stories)
- Interaction Testing: Storybook Add-on Specifications (Add specs inside stories)

DEMO:

1. Structural: storyshots.test.ts
2. Interaction: shapeSelector.stories.tsx, input.stories.tsx

Component Testing
Best Practices

- Avoid brittle selectors (id, class, tag, text content) => Use data-attributes
- Write tests for text content only if you want the test to fail when the text changes

- Avoid brittle selectors: class and ID are usually coupled to CSS styling and are fairly likely to change. Tags are too generic and may stop working when another one of the same tag is added to the component. Text content may change, and what about i18n? Consider using data-attributes specifically for targeting elements in component testing (e.g. data-testid="submit")
- Don't care if the button label changes from Submit to Save => don't make tests depend on it

End-to-End Testing

Cypress

- Time Travel
- Debuggability
- Automatic Waiting
- Spies, Stubs, and Clocks
- Network Traffic Control
- Consistent Results
- Native access

- Time Travel: Cypress takes snapshots as your tests run. Simply hover over commands in the Command Log to see exactly what happened at each step.
- Debuggability: Stop guessing why your tests are failing. Debug directly from familiar tools like Chrome DevTools. Our readable errors and stack traces make debugging lightning fast.
- Automatic Waiting: Never add waits or sleeps to your tests. Cypress automatically waits for commands and assertions before moving on. No more async hell.
- Spies, Stubs, and Clocks: Verify and control the behavior of functions, server responses, or timers. The same functionality you love from unit testing is right at your fingertips.
- Network Traffic Control: Easily control, stub, and test edge cases without involving your server. You can stub network traffic however you like.
- Consistent Results: Our architecture doesn't use Selenium or WebDriver. Say hello to fast, consistent and reliable tests that are flake-free.
- Behind Cypress is a Node.js server process. Cypress and the Node.js process constantly communicate, synchronize, and perform tasks on behalf of each other. Having access to both parts (front and back) gives Cypress the ability to respond to your application's events in real time, while at the same time work outside of the browser for tasks that require a higher privilege.

Cypress

- Framework-agnostic (React, Angular, Vue…)
- Currently, only supports Chrome and friends ☹️
- Really good documentation
- Doesn’t use Selenium 🎉
- Uses jQuery internally for DOM traversal, but is not synchronous:
  - Chain commands together using Promise-like API
- Uses Chai and Sinon internally for assertions

- Cypress wraps all DOM queries with robust retry-and-timeout logic that better suits how real web apps work. To match the behavior of web applications, Cypress is asynchronous and relies on timeouts to know when to stop waiting on an app to get into the expected state. Timeouts can be configured globally, or on a per-command basis.
- All DOM based commands automatically wait for their elements to exist in the DOM.
- You don’t have to assert (explicitly) to have a useful test. Even without assertions, a few lines of Cypress can ensure thousands of lines of code are working properly across the client and server!

DEMO: cypress folder

E2E Testing
Best Practices

- Don’t visit external sites (OAuth, Email…)
- Multiple assertions are desirable
- Reset state before tests run, not afterwards

- Don’t visit UI of external sites. Instead, use APIs or stubs/mock, server-side testing (“forgot password” email was sent)
- Multiple assertions: test runner shows you exactly which point failed. Creating the test state is usually expensive
- Reset: Dangling state can be used to debug tests and write partial tests as you write code. Inspect spies, routes etc. after the test ran or failed

Testing Redux Apps

- Action Creators: only when necessary
- Reducers: yes, check that state isn’t mutated
- Selectors: yes!
- Epics/Async action creators: do it! (call epic like any other fn, assert output actions)
- Middleware: yes (lots of stuff to fake)
- MapState/DispatchToProps/MergeProps: yes

Review

- Unit Tests
  - Jest
  - Property-based testing with testcheck
- Component Tests
  - Jest Snapshots
  - Enzyme
  - Storybook
- End-to-end Tests
  - Cypress

Further Reading

- Jest
  - https://jestjs.io/docs/en/getting-started
  - https://facebook.github.io/create-react-app/docs/running-tests
  - https://jestjs.io/docs/en/next/architecture
- Property-based testing
  - http://leebyron.com/testcheck-js/
  - http://jsverify.github.io/
  - https://www.propertesting.com/book_what_is_a_property.html
- Snapshot testing
  - https://jestjs.io/docs/en/snapshot-testing
  - https://benmccormick.org/2016/09/19/testing-with-jest-snapshots-first-impressions/
- Enzyme
  - https://airbnb.io/enzyme/
  - https://jestjs.io/docs/en/tutorial-react#enzyme
- Storybook
  - https://storybook.js.org/
  - https://github.com/storybooks/storybook/tree/master/addons/storyshots/storyshots-core
  - https://github.com/mthuret/storybook-addon-specifications
- Cypress
  - https://www.cypress.io/

Sources

- “Testing” Animation: https://www.behance.net/gallery/64580653/ASAP-Rocky-Testing-(Design-Direction)
- Neckband Images: https://blog.colettehq.com/tutorials/how-to-bind-knit-edges-the-ultimate-guide