

# FRONT-END TESTING

(DOESN'T HAVE TO SUCK)

**Marie Hooper**

# AGENDA

- Why test?
- What to test
- Types of tests
- How to test
  - Component tests
  - E2E tests
  - Examples

WHY TEST?

# GOALS FOR FRONT-END TESTING

- Ensure components are working properly
  - Does the UI look like we expect it to?
  - Can users interact with it?
- More confidence for later refactoring
  - Make sure functionality remains even when implementation changes
- Catch edge cases
  - Can be hard to make sure you have local data that catches these cases

SO, WHAT DO I TEST?

# TESTING APPROACH

## What **not** to test

- Implementation details
- Lifecycle methods
- Element event handlers
- Internal Component State



## What to test

- User interactions
- State changes that update UI
- Changes in props



TEST USE CASES, NOT CODE.

- *Kent C Dodds*

# EXAMPLES

- Can a user create a new patient?
- Does the user see the correct button based on status?
- Is validation working on that form?
- Is the user seeing accurate success/error messaging?



# TYPES OF TESTS

We want to spend some time here with cypress tests



#### THE FOUR TYPES OF TESTS

### End to End

A helper robot that behaves like a user to click around the app and verify that it functions correctly.

Sometimes called "functional testing" or e2e.

### Integration

Verify that several units work together in harmony.

### Unit

Verify that individual, isolated parts work as expected.

### Static

Catch typos and type errors as you write the code.

We want to spend **most** of our time here using component tests



Image credit to [Maggie Appleton](#)  
More to read on [KCD's Blog post](#)

# INTEGRATION TESTS


- Think of these like user stories
  - A user can enroll a new patient
- Test chunks of functionality as they are developed
  - Fill out the form, click the button, and then...
  - Test the payload sent with a vuex action or mocked Axios call
- Think about how the UI should reflect changes
  - Don't test the filter method itself, test that a user can click the filter and see the correct output in the UI

# E2E TESTS

- Think of these like whole workflows
  - We want to test an entire happy path for a user from logging into the app to accomplishing a goal.
- These can be modeled after the 'scripts' that QA uses

BUT, HOW??

# TESTING LIBRARY

- Tool to help test UI components in a user-centric way
- Allows you to write tests that closely resemble how a user might interact with your application
- Available for most modern FE frameworks
  - We are using  [Vue Testing Library](#) in eko\_web\_client

# HIGHLIGHTS

- Devs love testing-library (according to state of js survey)
- Use it with Cypress for more user-centric e2e testing
- ESLint plugin makes suggestions for best practices

**Highest Satisfaction**

Awarded to the technology with the highest percentage of satisfied users.

## Testing Library

With a sky-high satisfaction rating of **97%**, Testing Library is 2020's overall most beloved tool.

[Share](#)

**Runner Ups**

2. **Jest**: 96%
3. **GraphQL**: 94%

Read more at: <https://2020.stateofjs.com/en-US/awards/>

# WHAT'S SO GREAT ABOUT IT?

- Selectors help reinforce good a11y habits
  - Recommended by how accessible they are to users
  - Example: `getByRole` uses a role set by the browser and allows you to search by the 'accessible' name (e.g. the `label` or `aria-label`)
- Mocking user events is super easy
- Changing vuex store and props between tests is simple



# BEST PRACTICES

- Picking the “right” query for your test
  - [More about that here](#)
- Less is more
  - Mock only what you need for the component (store, props, etc)
  - Test “success criteria” for a user story

# EXAMPLE - USER INTERACTIONS

```
test('Search returns matched result', async () => {  
  render(Enrollment, { store });  
  const [search] = await screen.findAllByPlaceholderText('Filter by name, phone, or MRN ...');  
  
  await fireEvent.update(search, 'mick');  
  
  screen.getByText('Mickey Mouse');  
  screen.getByText('(123456)');  
});
```

# EXAMPLE - VUEX ACTION

```
test('Can successfully archive patient', async () => {
  render(EpisodePatientDetails, {
    props: { episode },
    store
  });

  await fireEvent.click(screen.getByRole('button', { name: 'Stop the episode & archive' }));
  screen.getByText('Are you sure you want to stop and archive this episode for Patrick Hooper?');

  await fireEvent.click(screen.getByRole('button', { name: 'Archive' }));

  expect(actions.archiveEpisode).toBeCalledWith(expect.anything(), uuid);
});
```

# EXAMPLE - AXIOS MOCK

```
test('User can add enrollment_code', async () => {  
  render(Enrollment, { store });  
  
  const [, editBtn] = await screen.findAllByTitle('Edit');  
  await fireEvent.click(editBtn);  
  
  await fireEvent.update(screen.getByPlaceholderText('Enter code'), 'new-kit-id');  
  await fireEvent.click(screen.getByTitle('Save'));  
  
  expect(axiosMock.put).toHaveBeenCalledWith('/home/episodes/54321', {  
    enrollment_code: 'new-kit-id'  
  });  
});
```