Build Youtube in React 05: polishing sticky side bar

BY [JAN GOEBEL](https://productioncoder.com/author/admin3917/) · JANUARY 21, 2019

1 Youtube Sidebar Intermediate Result

Back in the sattle again… Our sidebar is supposed to look like this:

*Click on the image to see it in high resolution*.

Our current application looks like this:

So let’s get to work and close the gap.

2 Improving side bar

2.1. Adding horizontal dividers

You have certainly already noticed that the mockup contains horizontal divider lines. Implementing them is easy as pie. Let’s use [Semantic UI’s divider component](https://react.semantic-ui.com/elements/divider/) for that. It’s dead simple, I promise.

Change the src/SideBar/SideBar.js file like so:

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {SideBarItem} from './SideBarItem/SideBarItem'; |
|  | import {Menu, Divider} from 'semantic-ui-react'; |
|  | import './SideBar.scss'; |
|  |  |
|  | export class SideBar extends React.Component { |
|  | render() { |
|  | return ( |
|  | <Menu borderless vertical stackable fixed='left' className='side-nav'> |
|  | <SideBarItem highlight={true} label='Home' icon='home'/> |
|  | <SideBarItem label='Trending' icon='fire'/> |
|  | <SideBarItem label='Followers' icon='spy'/> |
|  | <Divider/> |
|  | <SideBarItem label='History' icon='history'/> |
|  | <SideBarItem label='Watch later' icon='clock'/> |
|  | <SideBarItem label='Liked videos' icon='thumbs up'/> |
|  | <Divider/> |
|  | <SideBarItem label='Movies and Shows' icon='film'/> |
|  | <Divider/> |
|  | <SideBarItem label='Report history' icon='flag'/> |
|  | <SideBarItem label='Help' icon='help circle'/> |
|  | <SideBarItem label='Send feedback' icon='comment'/> |
|  | <Divider/> |
|  | </Menu> |
|  | ); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/e8a0662af1db1548012c0dcd6fdf4381/raw/c746fc1c8364269be5a7d7265e1f592ef8766d11/SideBar.js)[**SideBar.js**](https://gist.github.com/productioncoder/e8a0662af1db1548012c0dcd6fdf4381#file-sidebar-js) hosted with  by [**GitHub**](https://github.com/)

Note that we have added the <Divider/> component two times. Please don’t forget to import it from Semantic UI.

Ok, that one was really easy.

2.3. Headings for side bar

In addition to dividers, we also need headers so that the sections in the sidebar are easy to distinguish.

Let’s create a simple component for that.

1. create a new directory within src/containers/SideBar called SideBarHeader
2. create SideBarHeader.js and SideBarHeader.scss inside the src/containers/SideBar directory

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {Menu} from 'semantic-ui-react'; |
|  | import './SideBarHeader.scss'; |
|  |  |
|  | export function SideBarHeader(props) { |
|  | const heading = props.title ? props.title.toUpperCase() : ''; |
|  | return ( |
|  | <Menu.Item> |
|  | <Menu.Header className='side-bar-header'>{heading}</Menu.Header> |
|  | </Menu.Item> |
|  | ) |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/f7d726dd022024d1538fef68b32b431f/raw/c2251026b448b86bc72f240a88cb4e9e4f571ee1/SideBarHeader.js)[**SideBarHeader.js**](https://gist.github.com/productioncoder/f7d726dd022024d1538fef68b32b431f#file-sidebarheader-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | .side-bar-header { |
|  | color: #6d6d6d; |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/f7d726dd022024d1538fef68b32b431f/raw/c2251026b448b86bc72f240a88cb4e9e4f571ee1/SideBarHeader.scss)[**SideBarHeader.scss**](https://gist.github.com/productioncoder/f7d726dd022024d1538fef68b32b431f#file-sidebarheader-scss) hosted with  by [**GitHub**](https://github.com/)

Since the SideBarHeader will live inside a [Semantic UI Menu](http://react.semantic-ui.com/collections/menu/) component, it is basically just a [Menu.Item](https://react.semantic-ui.com/collections/menu/#null) component that contains a [Menu.Header](https://react.semantic-ui.com/collections/menu/) component.

This component expects to receive a prop called title and will display it in caps only (because that’s how Youtube does it).

We’ve added the .side-bar-header SCSS class to specify the text’s color.

Let’s make use of it.

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {SideBarItem} from './SideBarItem/SideBarItem'; |
|  | import {Menu, Divider} from 'semantic-ui-react'; |
|  | import './SideBar.scss'; |
|  | import {SideBarHeader} from './SideBarHeader/SideBarHeader'; |
|  |  |
|  | export class SideBar extends React.Component { |
|  | render() { |
|  | return ( |
|  | <Menu borderless vertical stackable fixed='left' className='side-nav'> |
|  | <SideBarItem highlight={true} label='Home' icon='home'/> |
|  | <SideBarItem label='Trending' icon='fire'/> |
|  | <SideBarItem label='Followers' icon='spy'/> |
|  | <Divider/> |
|  | <SideBarHeader title='Library'/> |
|  | <SideBarItem label='History' icon='history'/> |
|  | <SideBarItem label='Watch later' icon='clock'/> |
|  | <SideBarItem label='Liked videos' icon='thumbs up'/> |
|  | <Divider/> |
|  | <SideBarHeader title='More from Youtube'/> |
|  | <SideBarItem label='Movies and Shows' icon='film'/> |
|  | <Divider/> |
|  | <SideBarItem label='Report history' icon='flag'/> |
|  | <SideBarItem label='Help' icon='help circle'/> |
|  | <SideBarItem label='Send feedback' icon='comment'/> |
|  | <Divider/> |
|  | </Menu> |
|  | ); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/eb3e4862149b75647ebce0fbf3845c16/raw/a94f7155bcae0fc0991126d7bb2eef4aff402965/SideBar.js)[**SideBar.js**](https://gist.github.com/productioncoder/eb3e4862149b75647ebce0fbf3845c16#file-sidebar-js) hosted with  by [**GitHub**](https://github.com/)

We now have the a heading in our side bar. Nice

2.4. Subscriptions component for side bar

As far as the sidebar is concerned, there are two more things missing: The subscriptions section and the footer.

1. Create a new directory inside src/containers/SideBar and name it Subscriptions
2. Create a Subscriptions.js file inside src/containers/SideBar/Subscriptions

The Subscriptions (notice the plural) component will display a list of the channels the user has subscribed to. To display a single Subscription (notice singular), let’s create a Subscription component.

1. Inside src/containers/SideBar/Subscriptions create a new directory called Subscription
2. Inside src/containers/SideBar/Subscriptions/Subscription create a Subscription.js and a Subscription.scss file

We need a circular image, the channel’s title and the amount of new videos next to it. In case the channel is just live, we want to display a special broadcasting icon:

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {Icon, Image, Menu} from "semantic-ui-react"; |
|  | import './Subscription.scss'; |
|  |  |
|  | export function Subscription(props) { |
|  |  |
|  | let rightElement = null; |
|  | const {broadcasting, amountNewVideos} = props; |
|  | if (broadcasting) { |
|  | rightElement = <Icon name='signal'/>; |
|  | } else if (amountNewVideos) { |
|  | rightElement = <span className='new-videos-count'>{amountNewVideos}</span>; |
|  | } |
|  |  |
|  | return ( |
|  | <Menu.Item> |
|  | <div className='subscription'> |
|  | <div> |
|  | <Image src='http://via.placeholder.com/28x28' avatar/> |
|  | <span>{props.label}</span> |
|  | </div> |
|  | {rightElement} |
|  | </div> |
|  | </Menu.Item> |
|  | ); |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/c991094fbdbb468b45c1fadaffb87865/raw/0aef729bedccd7c1242e18e4ccb1a19c394e4627/Subscription.js)[**Subscription.js**](https://gist.github.com/productioncoder/c991094fbdbb468b45c1fadaffb87865#file-subscription-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | @import '../../../../styles/shared.scss'; |
|  |  |
|  | .subscription { |
|  | width: 100%; |
|  | display: flex; |
|  | justify-content: space-between; |
|  | align-items: center; |
|  |  |
|  | i.icon { |
|  | color: $red; |
|  | } |
|  |  |
|  | .new-videos-count { |
|  | color: $grey; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/c991094fbdbb468b45c1fadaffb87865/raw/0aef729bedccd7c1242e18e4ccb1a19c394e4627/Subscription.scss)[**Subscription.scss**](https://gist.github.com/productioncoder/c991094fbdbb468b45c1fadaffb87865#file-subscription-scss) hosted with  by [**GitHub**](https://github.com/)

Ok, so let’s talk about it. Since the right element varies depending on whether the channel is currently live, we create a rightElement variable at the beginning. If we are broadcasting then we display the respective icon using the [Semantic UI Icon](https://react.semantic-ui.com/elements/icon/) component. If the channel is currently not live and has published new videos, our right element is just a spanelement that displays a number.

For layout purposes we wrap the content on the left into **a .subscription div** element that serves as a flex container. We force the flex container to maximise the space between its elements by setting justify-content: space-between. The align-items: center property centers the div’s content vertically. Since we only have two elements in our flex container, both elements are 'glued' to the left and right because of our SCSS magic. For now, we just use a placeholder image for the channel’s icon because we haven’t connected our app to the Youtube backend yet.

2.5. Wiring up subscriptions component to side bar

Now that we have designed and implemented the Subscription component, let’s also use it.

Add the following code to your src/SideBar/Subscription.js

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {Subscription} from "./Subscription/Subscription"; |
|  | import {Divider} from "semantic-ui-react"; |
|  | import {SideBarHeader} from '../SideBarHeader/SideBarHeader'; |
|  |  |
|  | export class Subscriptions extends React.Component { |
|  | render() { |
|  | return ( |
|  | <React.Fragment> |
|  | <SideBarHeader title='Subscriptions'/> |
|  | <Subscription label='MusicChannel' broadcasting/> |
|  | <Subscription label='Coursea' amountNewVideos={10}/> |
|  | <Subscription label='TEDx Talks' amountNewVideos={23}/> |
|  | <Subscription label='Stanford iOS' amountNewVideos={4}/> |
|  | <Subscription label='Udacity' amountNewVideos={114}/> |
|  | <Divider/> |
|  | </React.Fragment> |
|  | ); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/2615451bce6536ba49efd1a4cb4d585a/raw/d624413303f2b837ce7b0363c0cdc82835140387/Subscriptions.js)[**Subscriptions.js**](https://gist.github.com/productioncoder/2615451bce6536ba49efd1a4cb4d585a#file-subscriptions-js) hosted with  by [**GitHub**](https://github.com/)

For now, we are just adding a few subscription elements to our sidebar. Notice that the channel with the label MusicChannel is broadcasting and will therefore be displayed with a respective icon. We use a [React.Fragment](https://reactjs.org/docs/fragments.html) element so that we don’t need a wrapper div.

3 Side bar footer

Oh man, that was a lot of work for a side bar. Let’s finish it off by creating a footer so our application looks more similar to the original Youtube app.

1. Create a new directory inside src/containers/SideBar and name it SideBarFooter
2. Create a a SideBarFooter.js and a SideBarFooter.scss file in the SideBarFooter directory you have just created.

This component is super easy, it’s basically just a little bit of text.

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import './SideBarFooter.scss' |
|  |  |
|  | export function SideBarFooter() { |
|  | return ( |
|  | <React.Fragment> |
|  | <div className='footer-block'> |
|  | <div>About Press Copyright</div> |
|  | <div>Creators Advertise</div> |
|  | <div>Developers +MyTube</div> |
|  | <div>Legal</div> |
|  | </div> |
|  | <div className='footer-block'> |
|  | <div>Terms Privacy</div> |
|  | <div>Policy & Safety</div> |
|  | <div>Test new features</div> |
|  | </div> |
|  | <div className='footer-block'> |
|  | <div>All prices include VAT</div> |
|  | </div> |
|  | <div className='footer-block'> |
|  | <div>© Productioncoder.com - A Youtube clone for educational purposes under fair use.</div> |
|  | </div> |
|  | </React.Fragment> |
|  | ); |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/9ff6060f5466e357a0efdda9c9962425/raw/0a86f207110c8f442fac7426baeddee1438e91f7/SideBarFooter.js)[**SideBarFooter.js**](https://gist.github.com/productioncoder/9ff6060f5466e357a0efdda9c9962425#file-sidebarfooter-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | .footer-block { |
|  | padding-bottom: 10px; |
|  | padding-left: 16px; |
|  | color: #918888; |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/9ff6060f5466e357a0efdda9c9962425/raw/0a86f207110c8f442fac7426baeddee1438e91f7/SideBarFooter.scss)[**SideBarFooter.scss**](https://gist.github.com/productioncoder/9ff6060f5466e357a0efdda9c9962425#file-sidebarfooter-scss) hosted with  by [**GitHub**](https://github.com/)

Now, I don’t think it needs much explanation here, we’re just creating a little bit of text.

I’ve just included this example for the the sake of completeness.

As a last step, add the SideBarFooter component to the end of the SideBar component.

|  |  |
| --- | --- |
|  | /\*...\*/ |
|  | import {SideBarFooter} from './SideBarFooter/SideBarFooter'; |
|  |  |
|  | export class SideBar extends React.Component { |
|  | render() { |
|  | return ( |
|  | <Menu borderless vertical stackable fixed='left' className='side-nav'> |
|  | {/\*rest unchanged\*/} |
|  | <SideBarItem label='Send feedback' icon='comment'/> |
|  | <Divider/> |
|  | <SideBarFooter/> |
|  | </Menu> |
|  | ); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/100f4dd05d3a823266c5dd37af02b7b1/raw/b4bea0a736e474676520b4f4b054a96855169089/SideBar.js)[**SideBar.js**](https://gist.github.com/productioncoder/100f4dd05d3a823266c5dd37af02b7b1#file-sidebar-js) hosted with  by [**GitHub**](https://github.com/)

And here’s how our footer looks like (c*lick on the image to see it in high resolution)*:

4 Wrap Up

That looks good. In the previous tutorials we’ve been dealing with a lot of Semantic UI's components. This was the last episode where we’ll make extensive use of it. All the other components are pretty much built from scratch. So in case you don’t like Semantic UI, congrats, you’re pretty much through . However, we will make use of Semantic UI's

# Build Youtube in React 06: Snapshot Testing using Jest and Enzyme

BY [JAN GOEBEL](https://productioncoder.com/author/admin3917/) · JANUARY 22, 2019

## 1. Testing React components in Jest and Enzyme

Just one more thing before we go on. We have already quite a few components and as part of our a professional software development approach, we would also like to test them as well.

Right now, our components are basically just markup and don’t contain any logic. But we can still test them!

### 1.1. Snapshot Testing

Creating a snapshot can be compared with taking a photo in the real world. When you create a snapshot, you basically create a photo of the component, i.e. you somehow store, how the component is supposed to look. Now if the component is changed later on, the photo (the snapshot) will look differently than before. When you run the available snapshot tests, you can compare the last stored snapshot with the current component. Therefore, it is immediately apparent what has changed. This approach is way easier and faster to do than other types of frontend testing we can do. These snapshot tests are unit tests. They don’t test how the different components interact, but are focused on one single component.

Fortunately there is a variety of tools out there that make our snapshot testing life much easier. One of the most popular snapshot testing packages is [Enzyme](https://github.com/airbnb/enzyme) which is developed and maintained by AirBnB. [Enzyme](https://github.com/airbnb/enzyme) itself is responsible for creating and storing component snapshots. However, to set up a test bench, we need more functionality than creating snapshots. We need a package that allows us to assertions and can compare different snapshots. Therefore, [Enzyme](https://github.com/airbnb/enzyme) is often used together with [Jest](https://jestjs.io/),  one of the most popular React/Javascript testing frameworks out there.

We’re also going to use them together.

### 1.2 Setting up Jest and Enzyme in create-react-app

First, we need to install Jest and Enzyme via npm or yarn.

The nice thing about create-react-app is that Jest is already pre-installed. So we only need [Enzyme](https://github.com/airbnb/enzyme).

|  |  |
| --- | --- |
|  | yarn add --dev enzyme enzyme-adapter-react-16 react-test-renderer enzyme-to-json |

[**view raw**](https://gist.github.com/productioncoder/d63e86a681165e5f766166622906fd38/raw/a79f80a9aefaf72d5091ce46f079884f3db73a87/install-enzyme.sh)[**install-enzyme.sh**](https://gist.github.com/productioncoder/d63e86a681165e5f766166622906fd38#file-install-enzyme-sh) hosted with  by [**GitHub**](https://github.com/)

So the thing is that Enzyme for itself is not sufficient. Depending on which React version you use, the overall render process is different. React 15 renders component differently than React 16 and so on. Since we want to use one universal API for [Enzyme](https://github.com/airbnb/enzyme), we need an adapter that connects [Enzyme](https://github.com/airbnb/enzyme) to the React version we are using. As we are using React 16.x we install the [enzyme-adapter-react-16 component](https://github.com/airbnb/enzyme/tree/master/packages/enzyme-adapter-react-16).

Why do we also need the [react-test-renderer](https://github.com/facebook/create-react-app/issues/2021) package? The thing is that React typically renders your component to the DOM. However, when we are testing, we don’t want to render to a DOM, we just want to render to Javascript objects because then it is easy to create snapshots.

Finally, we also install the [enzyme-to-json](https://github.com/adriantoine/enzyme-to-json) package. [Jest](https://jestjs.io/) and [Enzyme](https://github.com/airbnb/enzyme) are basically two separate libraries, so per default Jest does not understand [Enzyme](https://github.com/airbnb/enzyme) objects. Therefore, we need a package so that we [Enyzme](https://github.com/airbnb/enzyme) wrappers are compatible with [Jest](https://jestjs.io/).

### 1.3. So why again do we need this adapter?

I’ll give an analogy on why [Enzyme](https://github.com/airbnb/enzyme) needs a React adapter. Maybe that’s easier to understand.

Suppose you want to connect some periphery to your computer. Your machine only has USB-C ports. So if you want to a device with USB-A, you need a USB-C to USB-A adapter. If you want to connect to a monitor, you need a USB-C to HDMI / display port cable. So by having USB-C only, you have a universal physical interface that you can use to connect other devices to. So no matter what physical interface your periphery has, you can always connect to it as long as you have an adapter.

In this analogy, USB-C is the [Enzyme API](https://github.com/airbnb/enzyme) you use to write your testing code. We don’t want this API to change because then the [Enzyme API](https://github.com/airbnb/enzyme) would differ depending on which React version we use. Continuing with the analogy, the different peripheral devices that have USB-A / Display port /HDMI interfaces are the different React versions that you can write your tests against.

### 1.4 Configure Testing in create-react-app

Wow that was a lot of explanation. But don’t worry – since we have all the packages above, setting up testing in our create-react-app based application is really easy. create-react-app has [documentation on how to setup up testing](https://facebook.github.io/create-react-app/docs/running-tests). Its test setup allows us to create a file in src/setupTests where we can configure our testing.

1. Create a new file in src and name it setupTests.js. The naming here is important.

Paste the following code into this file to configure your test setup:

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import Enzyme from 'enzyme'; |
|  | import Adapter from 'enzyme-adapter-react-16'; |
|  | import {createSerializer} from 'enzyme-to-json'; |
|  |  |
|  | expect.addSnapshotSerializer(createSerializer({mode: 'deep'})); |
|  |  |
|  | // React 16 Enzyme adapter |
|  | Enzyme.configure({adapter: new Adapter()}); |

[**view raw**](https://gist.github.com/productioncoder/a29ee73240b3d9e6f5a4d0391401abfb/raw/64fe73dc0f12d12729a1bb5935ef80fe1ae0e587/setupTests.js)[**setupTests.js**](https://gist.github.com/productioncoder/a29ee73240b3d9e6f5a4d0391401abfb#file-setuptests-js) hosted with  by [**GitHub**](https://github.com/)

What do we do here? Remember that [enzyme-to-json](https://github.com/adriantoine/enzyme-to-json) provides a way so that [Jest](https://jestjs.io/) can understand [Enzyme](https://github.com/airbnb/enzyme) wrappers. We basically tell [expect](https://github.com/mjackson/expect) (an assertion tool that is part of [Jest](https://jestjs.io/)): look, we have a little bit of additional logic here so in case you encounter an [Enzyme](https://github.com/airbnb/enzyme) wrapper and you don’t know what to do with it, we give you an additional serializer so that it works anyway.

As already mentioned, we also have to configure [Enzyme](https://github.com/airbnb/enzyme) to our current React version. In our case this is React 16 and we just use and adapter we imported from [enzyme-adapter-react-16](https://github.com/airbnb/enzyme).

### 1.5. Testing our setup

Let’s see if our setup works.

Head over to your console and run

|  |  |
| --- | --- |
|  | yarn test |

[**view raw**](https://gist.github.com/productioncoder/e88c8989d9209373c51694c52e1d4cbc/raw/bc8354002c3d54324a771cbe637b6d981c0bba00/testing-test-setup.sh)[**testing-test-setup.sh**](https://gist.github.com/productioncoder/e88c8989d9209373c51694c52e1d4cbc#file-testing-test-setup-sh) hosted with  by [**GitHub**](https://github.com/)

If you encounter an error like this:

Cannot find module '/Users/jgoebel/Documents/dev/productioncoder/youtube-react/node\_modules/jest-cli'

at Function.Module.\_resolveFilename (internal/modules/cjs/loader.js:581:15)

at Function.Module.\_load (internal/modules/cjs/loader.js:507:25)

at Module.require (internal/modules/cjs/loader.js:637:17)

...

[**view raw**](https://gist.github.com/productioncoder/943ad1ca8d56e07953c891c4d6c5d9d3/raw/d43bf1d1b5936e3bf21fa954778d5ecd93f7d2ba/jest-cli-missing.md)[**jest-cli-missing.md**](https://gist.github.com/productioncoder/943ad1ca8d56e07953c891c4d6c5d9d3#file-jest-cli-missing-md) hosted with  by [**GitHub**](https://github.com/)

you have probably mixed npm with yarn commands. Just delete your node\_modules and your yarn.lock file and reinstall them again using

|  |  |
| --- | --- |
|  | yarn install |

[**view raw**](https://gist.github.com/productioncoder/7c78bdff21c9ce01d996b7498a7a7e3b/raw/709175a3d05de614bce7350fcd3e9c22c2a9d8b1/install-dependencies-yarn.sh)[**install-dependencies-yarn.sh**](https://gist.github.com/productioncoder/7c78bdff21c9ce01d996b7498a7a7e3b#file-install-dependencies-yarn-sh) hosted with  by [**GitHub**](https://github.com/)

Apart from that everything should just work out fine and you a file watcher should have been started.

### 1.6. Writing tests in create-react-app

The [create-react-app docs clearly list how our tests should be organized](https://github.com/facebook/create-react-app/blob/master/packages/react-scripts/template/README.md#filename-conventions) so that our test runner executes them:

1. .js suffix in \_\_tests\_\_ folders.
2. .test.js suffix.
3. .spec.js suffix.

We are going to colocate our snapshot tests with our components because they are unit tests.

We will also create \_\_tests\_\_ directories for each component because otherwise our logic is mixed with tests. I typically give unit tests a .unit.test.js suffix because then it is immediately apparent whether it is a unit or an integration test.

Let’s test the components we already have.

## 2 Testing our HeaderNav component

1. Create a new directory in src/containers/HeaderNav and name it \_\_tests\_\_ (the directory’s name is important)
2. Create a new file called HeaderNav.unit.test.js inside the directory you just created.

Let’s create our first snapshot test:

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {shallow} from 'enzyme'; |
|  | import HeaderNav from '../HeaderNav'; |
|  |  |
|  | test('renders HeaderNav', () => { |
|  | const wrapper = shallow( |
|  | <HeaderNav /> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |

[**view raw**](https://gist.github.com/productioncoder/a318f5db838923a28a0428db59a3f1e5/raw/44bb7f5dae8283ad788790b857c95ccb554ba0b9/HeaderNav.unit.test.js)[**HeaderNav.unit.test.js**](https://gist.github.com/productioncoder/a318f5db838923a28a0428db59a3f1e5#file-headernav-unit-test-js) hosted with  by [**GitHub**](https://github.com/)

Head over to your console and run

|  |  |
| --- | --- |
|  | yarn test |

[**view raw**](https://gist.github.com/productioncoder/0f77a127b0b4662f9bc7a93c78756e77/raw/6af153a6033c568a8d33b0dff3b1b290e6f779fa/yarn-test.sh)[**yarn-test.sh**](https://gist.github.com/productioncoder/0f77a127b0b4662f9bc7a93c78756e77#file-yarn-test-sh) hosted with  by [**GitHub**](https://github.com/)

Notice that a folder called \_\_snapshots\_\_ within our \_\_tests\_\_ directory has been created that contains .snap files. .snap files contain the markup, the render function returns from our components. Let’s illustrate this really quick.

### 2.1. Shallow Rendering in Jest and Enzyme

Let’s just talk about shallow rendering real quick. Shallow rendering means that we are executing the render method of a component, but don’t step down the entire view hierarchy. We’re just getting the markup that the render method returns and write this to a snapshot file. Here’s an example that is really easy:

|  |  |
| --- | --- |
|  | export function MyCustomComponent(props) { |
|  | return ( |
|  | <div> |
|  | <p>MyCustomComponent paragraph 1</p> |
|  | <p>MyCustomComponent paragraph 2</p> |
|  | </div> |
|  | ) |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/619453166418a2db706ab1b981172fe2/raw/f51a3fdbb71cd4f6d2effafe388cc0d64e80cc84/MyCustomComponent.js)[**MyCustomComponent.js**](https://gist.github.com/productioncoder/619453166418a2db706ab1b981172fe2#file-mycustomcomponent-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | export function ParentComponent(props) { |
|  | return ( |
|  | <div> |
|  | <p>Just a paragraph</p> |
|  | <MyCustomComponent/> |
|  | </div> |
|  | ); |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/619453166418a2db706ab1b981172fe2/raw/f51a3fdbb71cd4f6d2effafe388cc0d64e80cc84/ParentComponent.js)[**ParentComponent.js**](https://gist.github.com/productioncoder/619453166418a2db706ab1b981172fe2#file-parentcomponent-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | // Jest Snapshot v1, https://goo.gl/fbAQLP |
|  |  |
|  | exports[`renders ParentComponent`] = ` |
|  | <div> |
|  | <p>Just a paragraph</p> |
|  | <MyCustomComponent/> |
|  | </div> |
|  | `; |

[**view raw**](https://gist.github.com/productioncoder/619453166418a2db706ab1b981172fe2/raw/f51a3fdbb71cd4f6d2effafe388cc0d64e80cc84/ParentComponent.unit.test.js.snap)[**ParentComponent.unit.test.js.snap**](https://gist.github.com/productioncoder/619453166418a2db706ab1b981172fe2#file-parentcomponent-unit-test-js-snap) hosted with  by [**GitHub**](https://github.com/)

The ParentComponent.unit.test.snap file above shows content of the snapshot file, our test produces. And here you can see the **difference between mounting and shallow rendering**. Shallow rendering will not execute the rendermethod of custom components that are used within the current component. This is actually really good because our test for the ParentComponent should only cover the ParentComponent and not the elements it makes use of in its rendermethod. Like so, we are only testing one component. This is ok because each component will have its own snapshot test, so nothing remains untested.

## 3 Snapshot test all the components we currently have in our project!

Now that we have understood the basic concepts of snapshot testing, let’s apply this knowledge to our existing components.

Go ahead an create snapshot tests similar to the one we have created in section 2 for

1. SideNav
2. Footer
3. SideBarItem
4. SideNavHeader
5. Subscriptions
6. Subscription

For brevity, I won’t show all the code here because it would be very repetitive. Try to do this on your own. You can also head over to [this project’s git repository](https://github.com/productioncoder/youtube-react) if you need help.

Just one more thing. Some components take different sets of props. E.g. SideBarItem takes highlighted, icon and label as props. You can snapshot the component with different props like so.

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {shallow} from 'enzyme'; |
|  | import {SideBarItem} from '../SideBarItem'; |
|  |  |
|  | describe('SideBarItem', () => { |
|  | test('renders empty SideBarItem', () => { |
|  | const wrapper = shallow( |
|  | <SideBarItem/> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  |  |
|  | test('renders highlighted SideBarItem that navigates to /feed/trending', () => { |
|  | const wrapper = shallow( |
|  | <SideBarItem highlighted icon='fire' label='Trending'/> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  |  |
|  | test('render non-highlighted SideBarItem that navigates to /feed/trending', () => { |
|  | const wrapper = shallow( |
|  | <SideBarItem icon='fire' label='Trending'/> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  |  |
|  | test('Renders highlighted SideBarItem with no navigation', () => { |
|  | const wrapper = shallow( |
|  | <SideBarItem highlighted icon='fire' label='Trending'/> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/96a760f36e3511cac3bcc715faf543a9/raw/63a71fce12114493cb2de1b9fb0bfae5e34d294e/SideBarItem.unit.test.js)[**SideBarItem.unit.test.js**](https://gist.github.com/productioncoder/96a760f36e3511cac3bcc715faf543a9#file-sidebaritem-unit-test-js) hosted with  by [**GitHub**](https://github.com/)

You can also do that for the SideBarHeader and the Subscription component:

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {shallow} from 'enzyme'; |
|  | import {SideBarHeader} from '../SideNavHeader'; |
|  |  |
|  | describe('SideBarHeader', () => { |
|  | test('renders SideBarHeader with props.title=null', () => { |
|  | const wrapper = shallow( |
|  | <SideBarHeader/> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  |  |
|  | test('renders SideBarHeader with props.title=\'\'', () => { |
|  | const wrapper = shallow( |
|  | <SideBarHeader title=''/> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  |  |
|  | test('renders SideBarHeader with sample title', () => { |
|  | const wrapper = shallow( |
|  | <SideBarHeader title='Sample Title'/> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  | }); |

[**view raw**](https://gist.github.com/productioncoder/6213d7341156c62b056ec680d0ae8a67/raw/06d7013393aa6817e3e6e958504be3d1bbe96720/SideBarHeader.unit.test.js)[**SideBarHeader.unit.test.js**](https://gist.github.com/productioncoder/6213d7341156c62b056ec680d0ae8a67#file-sidebarheader-unit-test-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {shallow} from 'enzyme'; |
|  | import {Subscription} from '../Subscription'; |
|  |  |
|  | describe('Subscription', () => { |
|  | test('renders empty subscription', () => { |
|  | const wrapper = shallow( |
|  | <Subscription/> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  |  |
|  | test('renders broadcasting subscription', () => { |
|  | const wrapper = shallow( |
|  | <Subscription broadcasting label='Productioncoder'/> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  |  |
|  | test('renders non-broadcasting subscription with new videos', () => { |
|  | const wrapper = shallow( |
|  | <Subscription amountNewVideos={4} label='Productioncoder'/> |
|  | ); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  | }); |

[**view raw**](https://gist.github.com/productioncoder/6213d7341156c62b056ec680d0ae8a67/raw/06d7013393aa6817e3e6e958504be3d1bbe96720/Subscription.unit.test.js)[**Subscription.unit.test.js**](https://gist.github.com/productioncoder/6213d7341156c62b056ec680d0ae8a67#file-subscription-unit-test-js) hosted with  by [**GitHub**](https://github.com/)

In case you were wondering about the describe function: we use it to group tests for one component or one functionality together. It’s just best-practise.

## 4 Wrap Up

I know, testing is probably not your favourite activity. I get it.

But you can have greater confidence in the stuff you actually build. That’s why we do it in the first place.

# Build Youtube in React 07: Home Feed and Video Preview

BY [JAN GOEBEL](https://productioncoder.com/author/admin3917/) · JANUARY 23, 2019

## 1 Youtube Home Feed Mockups

We’re finally about to develop one of Youtube’s main components: its home feed. Let’s look at our mock to see what we’re up to.

Click the image to get it in high resolution.

*Video thumbnails are free stock images from https://www.pexels.com/ that explicitly don’t require any attribution*

We can see that we need some grid like component that shows us videos for different categories.

## 2 Adding VideoPreview component

### 2.1 Mockups

Let’s start with the component that holds the video thumbnail, its description and metadata. We will call this component VideoPreview. Here’s how it looks when we zoom in a little bit.

Click the image to get it in high resolution.

### 2.2 Adding VideoPreview and Home component skeletons

The first question – as always – is: is this a stateful or a presentational component? It makes sense to assume that probably some parent component will fetch data from an endpoint and then create a VideoPreview component for each fetched video. Therefore, we can presume that our VideoPreview component will get its information via props. We will therefore put it into the src/components directory.

1. Create a new directory inside src/components and name it VideoPreview
2. Inside this directory, create a VideoPreview.js and a VideoPreview.scss file

The typical workflow in a React project is to break the app down into smaller components and then start building them bottom up. However, since we want to actually see the results of our efforts as we are gradually adding more components, we will create the Home feed component skeleton now. The Home component will certainly fetch some information from the Youtube API and is therefore a stateful component.

1. Create a new directory inside src/containers and name it Home
2. Inside this directory, create a Home.js and a Home.scss file

We also make use of our Home component in src/App.js for better testing

|  |  |
| --- | --- |
|  | import React, {Component} from 'react'; |
|  | import HeaderNav from './containers/HeaderNav/HeaderNav'; |
|  | import {SideBar} from './containers/SideBar/SideBar'; |
|  | import {Home} from './containers/Home/Home'; |
|  | import './Home.scss'; |
|  |  |
|  | class App extends Component { |
|  | render() { |
|  | return ( |
|  | <React.Fragment> |
|  | <HeaderNav/> |
|  | <SideBar/> |
|  | <Home/> |
|  | </React.Fragment> |
|  | ); |
|  | } |
|  | } |
|  |  |
|  | export default App; |

[**view raw**](https://gist.github.com/productioncoder/1da96d59337b410de9a5dede4b61b5f2/raw/2664fef9f111482fa5b5bc6460615345cace7cc3/App.js)[**App.js**](https://gist.github.com/productioncoder/1da96d59337b410de9a5dede4b61b5f2#file-app-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {VideoPreview} from '../../components/VideoPreview/VideoPreview'; |
|  |  |
|  | export class Home extends React.Component { |
|  | render() { |
|  | return ( |
|  | <VideoPreview/> |
|  | ) |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/1da96d59337b410de9a5dede4b61b5f2/raw/2664fef9f111482fa5b5bc6460615345cace7cc3/Home.js)[**Home.js**](https://gist.github.com/productioncoder/1da96d59337b410de9a5dede4b61b5f2#file-home-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import './VideoPreview.scss'; |
|  |  |
|  | export class VideoPreview extends React.Component { |
|  | render() { |
|  | return ( |
|  | <div>This is must a VideoPreview placeholder</div> |
|  | ); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/1da96d59337b410de9a5dede4b61b5f2/raw/2664fef9f111482fa5b5bc6460615345cace7cc3/VideoPreview.js)[**VideoPreview.js**](https://gist.github.com/productioncoder/1da96d59337b410de9a5dede4b61b5f2#file-videopreview-js) hosted with  by [**GitHub**](https://github.com/)

### 2.3 Adjusting Youtube Home feed styling

But wait, what is that? Apart from the side bar and the top bar, we don’t see anything. How did that happen? Remember how we implemented SideBar?We added a  display: fixed CSS property somewhere. If you add an element like this, you basically glue it to a specific position of the view port and you remove it from the normal document flow. Therefore, we need to give the Home component a left margin that is equal to its side bar’s width.

Also remember that the top bar is fixed as well. Therefore, we also need a to give our Home component a margin-topproperty. Fortunately we have already put these widths in our styles/shared.scss file. So let’s just make use of it.

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {VideoPreview} from '../../components/VideoPreview/VideoPreview'; |
|  | import './Home.scss'; |
|  |  |
|  | export class Home extends React.Component { |
|  | render() { |
|  | return ( |
|  | <div className='home'> |
|  | <VideoPreview /> |
|  | </div> |
|  | ); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/2f68afedcd778021b38917e1b77d3845/raw/4449a280846a10115e7c536ba9e4c7011e63d827/Home.js)[**Home.js**](https://gist.github.com/productioncoder/2f68afedcd778021b38917e1b77d3845#file-home-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | @import '../../styles/shared.scss'; |
|  |  |
|  | .home { |
|  | margin-top: $header-nav-height; |
|  | margin-left: $sidebar-left-width; |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/2f68afedcd778021b38917e1b77d3845/raw/4449a280846a10115e7c536ba9e4c7011e63d827/Home.scss)[**Home.scss**](https://gist.github.com/productioncoder/2f68afedcd778021b38917e1b77d3845#file-home-scss) hosted with  by [**GitHub**](https://github.com/)

### 2.4 Displaying the thumbnail image in VideoPreview

Let’s build out the VideoPreview component step by step and take care of the video thumbnail image first.

Here is our markup:

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {Image} from 'semantic-ui-react'; |
|  | import './VideoPreview.scss'; |
|  |  |
|  | export class VideoPreview extends React.Component { |
|  | render() { |
|  | return ( |
|  | <div className='video-preview'> |
|  | <div className='image-container'> |
|  | <Image src='http://via.placeholder.com/210x118'/> |
|  | <div className='time-label'> |
|  | <span>05:22</span> |
|  | </div> |
|  | </div> |
|  | </div> |
|  | ); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/a7c3e5c22143a81d04031982f1359147/raw/025569efd3430a35b625c0c1fb439efb41200bc4/VideoPreview.js)[**VideoPreview.js**](https://gist.github.com/productioncoder/a7c3e5c22143a81d04031982f1359147#file-videopreview-js) hosted with  by [**GitHub**](https://github.com/)

Now you might have asked yourself why the markup is nested the way it is. Well, that’s because of our SCSS.

We will use CSS-Grid to accomplish the layout we want. If you are not familiar with CSS Grid, I highly recommend reading [this excellent article](https://css-tricks.com/snippets/css/complete-guide-grid/). CSS-Grid is currently the most powerful lay-outing system that CSS offers. It is more powerful than Flexbox because it allows us to create two-dimensional layouts. Flexbox only allows us to create one dimensional layouts unless we nest several flex containers in one another.

### 2.5 Lay-outing VideoPreview with CSS-Grid

Let’s revisit our VideoPreview mockup again and let’s specifically think about how we would lay it out in a grid.

We would probably go with a grid that has one column and two rows. The first grid cell (here in green) will display the video image thumbnail. The second grid cell displays some meta information like the video’s title, the amount of views etc.

Now you might be wondering why the grid lines are marked in different colours. The horizontal grid lines are all marked in red and get assigned a number starting form 1. The same is true for the vertical lines which are marked in black. However, since we only have one column, we only have two vertical lines instead of three. Why is this that useful? Well, because that’s how CSS-Grid (or Grid in short) works.

Let me show you the code that will create the layout we want.

|  |  |
| --- | --- |
|  | @import '../../styles/shared.scss'; |
|  |  |
|  | .video-preview { |
|  | display: grid; |
|  | grid: 118px auto / 210px; |
|  | } |
|  |  |
|  | .image-container { |
|  | position: relative; |
|  | grid-row: 1 / 2; |
|  | grid-column: 1 / 2; |
|  |  |
|  | /\* Video duration label at bottom right \*/ |
|  | .time-label { |
|  | /\* ... \*/ |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/3649700e20f2a5cc8ae6654cf64983db/raw/c261c3bb360f5703df20c88a4796c650683c2c67/VideoPreview.scss)[**VideoPreview.scss**](https://gist.github.com/productioncoder/3649700e20f2a5cc8ae6654cf64983db#file-videopreview-scss) hosted with  by [**GitHub**](https://github.com/)

If we want to use CSS-Grid we have to make one element a so called grid-container. Similar to Flexbox where we make an element a flex-container by adding display: flex, we’re now adding display:grid to make the element a grid container. The actual magic lies in the grid: 118px auto / 210px statement.

This means that our grid will have two rows and that the first row will have a height of 118px. That means that the first and the second red line will be 118px away from each other. Why? Well because that’s the height of a video thumbnail in Youtube. At least that’s what I saw in the Chrome Dev tools. The second keyword is auto. This basically means: size the second row automatically and make sure that the content fits in somehow. Note that this is a rather vague and intuitive explanation of what auto does. But I won’t cover the CSS-Grid specification here in detail.

The text after the / describes the columns we want to have. As shown in our little image above, we will only have one column. This column will have a width of 210px because that’s how Youtube sizes its video thumbnails. Feel free to open up your developer console in Youtube and to check it out for yourself.

The .image-container div contains the video preview image. By setting grid-row to 1 / 2, we instruct CSS Grid to place it between the first second horizontal grid line, i.e. between the first two red horizontal lines. By setting grid-column to 1 / 2, we also force  to put our div into the first column. Well that’s kind of clear, we only have one column after all.

### 2.6 Displaying video duration by using absolute positioning

I haven’t shown you the SCSS for displaying the video duration yet. That is, because we are using absolute positioning and because that would have distracted you from understanding how we use CSS-Grid.

Now you also come to understand why we use position:relative inside our .image-container class. We need it for the positioning of our video duration label.

So here’s our SCSS magic.

|  |  |
| --- | --- |
|  | /\*...\*/ |
|  | $text-color-dark: #111111; |

[**view raw**](https://gist.github.com/productioncoder/e471816e4a5c0799aa3d5ef75efd4130/raw/775b2cad228509bfa167c43b38efb222b7868a5d/shared.scss)[**shared.scss**](https://gist.github.com/productioncoder/e471816e4a5c0799aa3d5ef75efd4130#file-shared-scss) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | /\* ... \*/ |
|  | .time-label { |
|  | position: absolute; |
|  | background: $text-color-dark; |
|  | bottom: 0; |
|  | right: 0; |
|  | opacity: 0.8; |
|  | border-radius: 2px; |
|  | font-weight: 500; |
|  | color: white; |
|  | margin: 4px; |
|  | padding: 2px 4px; |
|  | line-height: 12px; |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/e471816e4a5c0799aa3d5ef75efd4130/raw/775b2cad228509bfa167c43b38efb222b7868a5d/VideoPreview.scss)[**VideoPreview.scss**](https://gist.github.com/productioncoder/e471816e4a5c0799aa3d5ef75efd4130#file-videopreview-scss) hosted with  by [**GitHub**](https://github.com/)

Let’s think again how we want our video duration label to be positioned. It should always stay at the bottom right corner. This is a perfect use case for absolute positioning. By using absolute positioning, we tell our .time-label div to be 0px away from both the bottom and the right. Absolute positioning basically allows us to glue an element somewhere relative to a parent element. But there is one more thing. When we use absolute positioning, the browser positions our element relative to the first positioned ancestor? What? Well that means our element gets positioned to the first ancestor which has either display: absolute or display: relative in its CSS. Since we always want to position our div relative to the .image-container, we’ve given it the position: relative property. Without this, our SCSS does not work.

Apart from that we do a little bit of cosmetics in our .time-label class, we add a little bit of margin and make the background dark and a little bit opaque. I guess this does not need much explanation.

If you are confused with how the file now actually looks, you can scroll down and look at all the code. You can also head over to our [Github repository](https://github.com/productioncoder/youtube-react/blob/master/src/components/VideoPreview/VideoPreview.scss) and have a look at the file directly. Note that there might be more CSS than we have here because we will add additional functionality to this component later on.

### 2.7 VideoPreview channel title, views and video title

Right now, we only have a placeholder image and a hard-coded video duration. But we still need to display the video’s title, its release date and the channel name.

|  |  |
| --- | --- |
|  | {/\*image container ends here\*/} |
|  | <div className='video-info'> |
|  | <div className='semi-bold show-max-two-lines'>Video title</div> |
|  | <div className='video-preview-metadata-container'> |
|  | <div className='channel-title'>Channel title</div> |
|  | <div><span>2.1M views • 2 days ago</span></div> |
|  | </div> |
|  | </div> |

[**view raw**](https://gist.github.com/productioncoder/7c2ae0b50c86d02a1f6619572e48221e/raw/9042469a229b990f901baec1d9ce396e0178413b/VideoPreview.js)[**VideoPreview.js**](https://gist.github.com/productioncoder/7c2ae0b50c86d02a1f6619572e48221e#file-videopreview-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | /\* |
|  | place this at the top level |
|  | Rest of the file unchanged |
|  | \*/ |
|  |  |
|  | .video-info { |
|  | color: $text-color-dark; |
|  |  |
|  | .video-preview-metadata-container { |
|  | padding-top: 5px; |
|  | font-size: 13px; |
|  | color: #6E6E6E; |
|  |  |
|  | .channel-title { |
|  | white-space: nowrap; |
|  | overflow: hidden; |
|  | text-overflow: ellipsis; |
|  | } |
|  | } |
|  |  |
|  | .show-max-two-lines { |
|  | overflow: hidden; |
|  | line-height: 1.4em; |
|  | max-height: 2.8em; |
|  | } |
|  |  |
|  | .semi-bold { |
|  | font-weight: 600; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/7c2ae0b50c86d02a1f6619572e48221e/raw/9042469a229b990f901baec1d9ce396e0178413b/VideoPreview.scss)[**VideoPreview.scss**](https://gist.github.com/productioncoder/7c2ae0b50c86d02a1f6619572e48221e#file-videopreview-scss) hosted with  by [**GitHub**](https://github.com/)

We create a new element with the class .video-info and place it directly below where our .image-container element ends. There are a few things are worth mentioning here in terms of CSS.

We only want our title to span two lines at maximum. Therefore, we added the .show-max-two-lines class. By making the overflowing content “invisible” and by setting the maximum height to two times the element’s line-height, we make sure, that we hide part of the title if it is too long.

In case of the channel’s name, things are a little bit different. Since we want the channel’s title to be on exactly one line, we add white-space: now-wrap. In case the channel’s title is too long, we abbreviate it with ... This is what text-overflow: ellipsis does.

### 2.8 Full VideoPreview code

Here’s the complete code for our component:

|  |  |
| --- | --- |
|  | /\* ... \*/ |
|  | $text-color-dark: #111111; |

[**view raw**](https://gist.github.com/productioncoder/a09347f195d0f8986c3af3ec30e2a4d8/raw/5d4ea0ff44ab408c3b3c43ab040187daf18007a2/shared.scss)[**shared.scss**](https://gist.github.com/productioncoder/a09347f195d0f8986c3af3ec30e2a4d8#file-shared-scss) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {Image} from 'semantic-ui-react'; |
|  | import './VideoPreview.scss'; |
|  |  |
|  | export class VideoPreview extends React.Component { |
|  | render() { |
|  | return ( |
|  | <div className='video-preview'> |
|  | <div className='image-container'> |
|  | <Image src='http://via.placeholder.com/210x118'/> |
|  | <div className='time-label'> |
|  | <span>05:22</span> |
|  | </div> |
|  | </div> |
|  |  |
|  | <div className='video-info'> |
|  | <div className='semi-bold show-max-two-lines'>Video title</div> |
|  | <div className='video-preview-metadata-container'> |
|  | <div className='channel-title'>Channel title</div> |
|  | <div><span>2.1M views • 2 days ago</span></div> |
|  | </div> |
|  | </div> |
|  | </div> |
|  | ); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/a09347f195d0f8986c3af3ec30e2a4d8/raw/5d4ea0ff44ab408c3b3c43ab040187daf18007a2/VideoPreview.js)[**VideoPreview.js**](https://gist.github.com/productioncoder/a09347f195d0f8986c3af3ec30e2a4d8#file-videopreview-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | @import '../../styles/shared.scss'; |
|  |  |
|  | .video-preview { |
|  | display: grid; |
|  | grid: 118px auto / 210px; |
|  | } |
|  |  |
|  | .video-info { |
|  | grid-row: 2 / 3; |
|  | grid-column: 1 / 2; |
|  | color: $text-color-dark; |
|  |  |
|  | .video-preview-metadata-container { |
|  | padding-top: 5px; |
|  | font-size: 13px; |
|  | color: #6E6E6E; |
|  |  |
|  | .channel-title { |
|  | white-space: nowrap; |
|  | overflow: hidden; |
|  | text-overflow: ellipsis; |
|  | } |
|  | } |
|  |  |
|  | .show-max-two-lines { |
|  | overflow: hidden; |
|  | line-height: 1.4em; |
|  | max-height: 2.8em; |
|  | } |
|  |  |
|  | .semi-bold { |
|  | font-weight: 600; |
|  | } |
|  | } |
|  |  |
|  | .image-container { |
|  | position: relative; |
|  | grid-row: 1 / 2; |
|  | grid-column: 1 / 2; |
|  |  |
|  | /\* Video duration label at bottom right \*/ |
|  | .time-label { |
|  | position: absolute; |
|  | background: $text-color-dark; |
|  | bottom: 0; |
|  | right: 0; |
|  | opacity: 0.8; |
|  | border-radius: 2px; |
|  | font-weight: 500; |
|  | color: white; |
|  | margin: 4px; |
|  | padding: 2px 4px; |
|  | line-height: 12px; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/a09347f195d0f8986c3af3ec30e2a4d8/raw/5d4ea0ff44ab408c3b3c43ab040187daf18007a2/VideoPreview.scss)[**VideoPreview.scss**](https://gist.github.com/productioncoder/a09347f195d0f8986c3af3ec30e2a4d8#file-videopreview-scss) hosted with  by [**GitHub**](https://github.com/)

## 4 Tests

To wrap up, let’s just add a snapshot test. Since our component currently doesn’t accept any props, one snapshot test is sufficient.

1. Create a \_\_tests\_\_ directory inside your VideoPreview component
2. Create a VideoPreview.unit.test.js file inside the directory you have just created.

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {shallow} from 'enzyme'; |
|  | import {VideoPreview} from '../VideoPreview'; |
|  |  |
|  | describe('VideoPreview', () => { |
|  | test('renders', () => { |
|  | const wrapper = shallow(<VideoPreview/>); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  | }); |

[**view raw**](https://gist.github.com/productioncoder/28cc15374b70837c57bcfb6855a68c1f/raw/921796c1e92652c5c7b0f0160d307c7eb6be33c5/VideoPreview.unit.test.js)[**VideoPreview.unit.test.js**](https://gist.github.com/productioncoder/28cc15374b70837c57bcfb6855a68c1f#file-videopreview-unit-test-js) hosted with  by [**GitHub**](https://github.com/)

## 5 Wrap up

That was a lot of code for one seemingly easy component. But it works. Let’s have a look at the result in our browser.

Now you might have asked yourself why we wrote it in CSS-Grid. Well, first of all it’s a good exercise and second, we will make this component more flexible in the future by adding more lay-outing options to it. I think CSS-Grid makes this a little bit easier, but we could have also achieved the same thing in Flexbox.

# Build Youtube in React 08: Home Feed and Video Grid

BY [JAN GOEBEL](https://productioncoder.com/author/admin3917/) · JANUARY 24, 2019

## 1 Youtube Home Feed video grid

In the last tutorial, we started building out Home feed components. Let’s look at our mockup again.

Click on the image to see it in high resolution.

Video thumbnails are free stock images from https://www.pexels.com/ that explicitly don’t require any attribution

As of now, we have a component which can show video video preview elements.

Click on the image to see it in high resolution.

## 2 Introducing VideoGrid component

Youtube shows suggestions for interesting topics grouped by a certain topic. The videos are shown in a grid. Typically we have 12 videos per topic or category. Let’s create a component that creates a proper grid view of the available videos.

Because the VideoGrid component will only be responsible for lay-outing VideoPreview components properly, it will probably not be connected to our Redux state later on. Therefore, we consider it a presentational component and will put it into src/components.

### 2.1. Creating VideoGrid component files

1. Create a directory called VideoGrid inside src/components
2. Create a VideoGrid.js and VideoGrid.scss file inside the directory you have just created

### 2.2. Design VideoGrid component.

That’s any easy one. This component will show a list of available videos grouped by a specific topic (e.g. entertainment, cars). For each topic, 12 VideoPreview components are displayed. Since we haven’t connected our app to the Youtube API backend, we will show some placeholder VideoPreview elements. Later on we will of course generate them using loops and real data. Therefore, the current markup could look a little bit weird .

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import './VideoGrid.scss'; |
|  | import {VideoGridHeader} from "./VideoGridHeader/VideoGridHeader"; |
|  | import {Divider} from "semantic-ui-react"; |
|  | import {VideoPreview} from '../VideoPreview/VideoPreview'; |
|  |  |
|  | export function VideoGrid(props) { |
|  | const divider = props.hideDivider ? null : <Divider/>; |
|  | return ( |
|  | <React.Fragment> |
|  | <h4>Trending</h4> |
|  | <div className='video-grid'> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | </div> |
|  | {divider} |
|  | </React.Fragment> |
|  | ); |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/e1fc7d0850b9e6aef270e67de115bba2/raw/7bf715d6073dcae5b5c6555a9ede7cf28d64f346/VideoGrid.js)[**VideoGrid.js**](https://gist.github.com/productioncoder/e1fc7d0850b9e6aef270e67de115bba2#file-videogrid-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | .video-grid { |
|  | display: flex; |
|  | flex-wrap: wrap; |
|  |  |
|  | & > \* { |
|  | margin-left: 4px; |
|  | margin-bottom: 24px; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/e1fc7d0850b9e6aef270e67de115bba2/raw/7bf715d6073dcae5b5c6555a9ede7cf28d64f346/VideoGrid.scss)[**VideoGrid.scss**](https://gist.github.com/productioncoder/e1fc7d0850b9e6aef270e67de115bba2#file-videogrid-scss) hosted with  by [**GitHub**](https://github.com/)

Our CSS is pretty straight forward. The .video-grid element is a flex container hat aligns its elements horizontally. We’ve added flex-wrap: wrap so that in case there is not enough room, the VideoPreview elements will wrap to the next line. Without specifying this property, flexbox would try to cram all VideoPreview elements into one single row.

We also want our grid to add some spacing between the VideoPreview items. That’s why we give every direct descendant of the .video-grid element, i.e. every VideoPreview a margin on the left and a margin at the bottom.

One row in a video grid should hold at most six VideoPreview elements. We’re going to enforce that in our Homecomponent in the next section. The VideoGrid element should not have any max widths so that it remains flexible

### 2.3 Updating Home component

To see what we have achieved so far, we should update our Home component and make use of our VideoGridcomponents.

Update your src/containers/Home/Home.js to

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import './Home.scss'; |
|  | import {VideoGrid} from '../../components/VideoGrid/VideoGrid'; |
|  | import {SideBar} from '../SideBar/SideBar'; |
|  |  |
|  | export class Home extends React.Component { |
|  | render() { |
|  | return ( |
|  | <React.Fragment> |
|  | <SideBar/> |
|  | <div className='home'> |
|  | <div className="responsive-video-grid-container"> |
|  | <VideoGrid title='Trending'/> |
|  | <VideoGrid title='Autos & Vehicles' hideDivider={true}/> |
|  | </div> |
|  | </div> |
|  | </React.Fragment> |
|  | ); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/d5701167a3b58a9e021365d6b217720c/raw/7835ca94581eef15a741e04a903c58d6a5b06f17/Home.js)[**Home.js**](https://gist.github.com/productioncoder/d5701167a3b58a9e021365d6b217720c#file-home-js) hosted with  by [**GitHub**](https://github.com/)

Wow, wait what are these CSS classes? Since we want our Home feed to look similar to Youtube’s we must centre our video grids horizontally. Therefore, we need a little bit of CSS tweaking.

We’re making use of one of the most modern CSS modules: CSS Grid (or Grid in short). If you are not familiar with CSS grid, I highly recommend you read a little bit [about it here](https://css-tricks.com/snippets/css/complete-guide-grid/).

It’s basically Flexbox just for two dimensions. Flexbox and CSS Grid actually work really well together.

### 2.4 Updating Home component styling

Grid will save you so much time and effort once you have picked it up. I promise.

|  |  |
| --- | --- |
|  | @import '../../styles/shared.scss'; |
|  |  |
|  | .home { |
|  | margin-top: $header-nav-height; |
|  | margin-left: $sidebar-left-width; |
|  | display: grid; |
|  | grid: auto / auto; |
|  | justify-content: center; |
|  | } |
|  | @media all and (min-width: 476px) { |
|  | .responsive-video-grid-container { |
|  | max-width: 240px; |
|  | } |
|  | } |
|  |  |
|  | @media all and (min-width: 700px) { |
|  | .responsive-video-grid-container { |
|  | max-width: 472px; |
|  | } |
|  | } |
|  |  |
|  | @media all and (min-width: 900px) { |
|  | .responsive-video-grid-container { |
|  | max-width: 667px; |
|  | } |
|  | } |
|  |  |
|  | @media all and (min-width: 1096px) { |
|  | .responsive-video-grid-container { |
|  | max-width: 864px; |
|  | } |
|  | } |
|  |  |
|  | @media all and (min-width: 1370px) { |
|  | .responsive-video-grid-container { |
|  | max-width: 1096px; |
|  | } |
|  | } |
|  |  |
|  | @media all and (min-width: 1370px) { |
|  | .responsive-video-grid-container { |
|  | max-width: 1096px; |
|  | } |
|  | } |
|  |  |
|  | @media all and (min-width: 1560px) { |
|  | .responsive-video-grid-container { |
|  | max-width: 1284px; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/359b8f2528573e9eacbd899c162b8ed6/raw/e2c2e0fbbca1c15328e51caf6707849660eafbda/Home.scss)[**Home.scss**](https://gist.github.com/productioncoder/359b8f2528573e9eacbd899c162b8ed6#file-home-scss) hosted with  by [**GitHub**](https://github.com/)

What are we doing here? We are making our .home element a grid container by setting display: grid. We only want to have one row and one column because Grid makes it easy to horizontally center our elements. Therefore, we’re telling Grid we want one column and one row and it should figure out their widths on its own by setting grid: auto / auto.

We also force our grid to horizontally centre its children  by setting justify-content: center. And then we insert the only child of our grid with a class of .responsive-video-grid-container. This element is responsible for capping the maximum size of our VideoGrid elements. Our VideoGrid element will try to place all VideoPreview items in one row unless there is not enough space. If there’s not enough space, it will wrap. However, we want our VideoPreview items to be nicely centred. Therefore we limit the width of our VideoGrid elements by using media queries.

Try it on your own. The maximum amount of VideoPreview items should in one row is six. If you make make your browser window smaller you will notice that suddenly, you only see five elements in one row. We do this so our Home feed still looks nice on smaller screens.

And this is what we get.

Click on the image to see it in high resolution.

Looks ok, but the spacing is not quite right. Let’s fix that.

### 2.5 Adding VideoGridHeader element

If you have a closer look at each video section heading, you’ll notice that there are some spacing issues. Since we might want to add more content to our heading later on, let’s just put it into a simple component.

1. Create a new directory inside the VideoGrid directory and call it VideoGridHeader
2. Create a VideoGridHeader.js and a VideoGridHeader.scss file in the directory you just created

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import './VideoGridHeader.scss'; |
|  |  |
|  | export function VideoGridHeader(props) { |
|  | return ( |
|  | <div className='video-grid-header'> |
|  | <span className='title'>{props.title}</span> |
|  | </div> |
|  | ); |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/9f341c6a2d9bf4343e551d26590164cb/raw/a0dc0425459268a9f58ecedde3092084aff51ee5/VideoGridHeader.js)[**VideoGridHeader.js**](https://gist.github.com/productioncoder/9f341c6a2d9bf4343e551d26590164cb#file-videogridheader-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | .video-grid-header { |
|  | padding-bottom: 24px; |
|  | padding-top: 14px; |
|  | .title { |
|  | font-size: 1.2rem; |
|  | font-weight: bold; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/9f341c6a2d9bf4343e551d26590164cb/raw/a0dc0425459268a9f58ecedde3092084aff51ee5/VideoGridHeader.scss)[**VideoGridHeader.scss**](https://gist.github.com/productioncoder/9f341c6a2d9bf4343e551d26590164cb#file-videogridheader-scss) hosted with  by [**GitHub**](https://github.com/)

As you can see, it’s just a little bit of cosmetics.

Let’s also update our VideoGrid component. Just replace the <h4> tag with our new VideoGridHeader component like so:

|  |  |
| --- | --- |
|  | <div className='video-section'> |
|  | <VideoGridHeader title='Trending'/> |
|  | <div className='video-grid'> |
|  | {/\*...\*/} |

[**view raw**](https://gist.github.com/productioncoder/0b5ca659b88bc45ab7dd3f71d478070f/raw/c7f0ed1ad2b0ebacc1abc84f82ba794ca4068031/VideoGrid.js)[**VideoGrid.js**](https://gist.github.com/productioncoder/0b5ca659b88bc45ab7dd3f71d478070f#file-videogrid-js) hosted with  by [**GitHub**](https://github.com/)

This looks much better. But there’s one more thing with the video grid dividers

### 2.6 Make video grid dividers removable

Now you may have noticed that we get a grey divider line at the bottom of our video grids. While this is useful to better group videos of the same categories, it doesn’t really make sense for the last VideoGrid element that is rendered. Therefore, let’s fix this:

|  |  |
| --- | --- |
|  | export function VideoGrid(props) { |
|  | const divider = props.hideDivider ? null : <Divider/>; |
|  |  |
|  | return ( |
|  | <React.Fragment> |
|  | <h4>Trending</h4> |
|  | <div className='video-grid'> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | <VideoPreview/> |
|  | </div> |
|  | {divider} |
|  | </React.Fragment> |
|  | ); |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/d631a4cc21c62b7927eface03061c719/raw/1aff41e299fce8bef3137ae48bb1f56ff74fe6d0/VideoGrid.js)[**VideoGrid.js**](https://gist.github.com/productioncoder/d631a4cc21c62b7927eface03061c719#file-videogrid-js) hosted with  by [**GitHub**](https://github.com/)

We just introduce a hideDivider prop that can be used to hide the grey divider line at the end of each video grid.

Finally, we also have to update our home component.

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import './Home.scss'; |
|  | import {VideoGrid} from '../../components/VideoGrid/VideoGrid'; |
|  |  |
|  | export class Home extends React.Component { |
|  | render() { |
|  | return ( |
|  | <div className='home'> |
|  | <VideoGrid title='Trending'/> |
|  | <VideoGrid title='Autos & Vehicles' hideDivider={true}/> |
|  | </div> |
|  | ); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/productioncoder/379c9d9bb11719512dca97c7041cd8c8/raw/0521c92130cf355ec4bc5fffe5bab72956e13fb1/Home.js)[**Home.js**](https://gist.github.com/productioncoder/379c9d9bb11719512dca97c7041cd8c8#file-home-js) hosted with  by [**GitHub**](https://github.com/)

## 3 Adding tests

The only thing that’s missing are a few test cases.

1. Create a \_\_tests\_\_ directory inside the VideoGrid and inside VideoGridHeader directory
2. Add a VideoGrid.unit.test.js file inside VideoGrid/\_\_tests\_\_
3. Add a VideoGridHeader.unit.test.js file inside VideoGrid/VideoGridHeader/\_\_tests\_\_

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {shallow} from 'enzyme'; |
|  | import {VideoGrid} from '../VideoGrid'; |
|  |  |
|  | describe('VideoGrid', () => { |
|  | test('renders without props', () => { |
|  | const wrapper = shallow(<VideoGrid/>); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  | test('renders with title prop', () => { |
|  | const wrapper = shallow(<VideoGrid title='Trending'/>); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  | test('renders without divider', () => { |
|  | const wrapper = shallow(<VideoGrid hideDivider={true}/>); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  | }); |

[**view raw**](https://gist.github.com/productioncoder/0fb09562b1d5678185419b6f5c66bfd0/raw/01c1339008ab53b569f76fe7d255431f5f21c507/VideoGrid.unit.test.js)[**VideoGrid.unit.test.js**](https://gist.github.com/productioncoder/0fb09562b1d5678185419b6f5c66bfd0#file-videogrid-unit-test-js) hosted with  by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | import React from 'react'; |
|  | import {shallow} from 'enzyme'; |
|  | import {VideoGridHeader} from '../VideoGridHeader'; |
|  |  |
|  | describe('VideoGridHeader', () => { |
|  | test('renders without props', () => { |
|  | const wrapper = shallow(<VideoGridHeader/>); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  | test('renders with empty string header', () => { |
|  | const wrapper = shallow(<VideoGridHeader title=''/>); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  | test('renders with title', () => { |
|  | const wrapper = shallow(<VideoGridHeader title='Autos & Vehicles'/>); |
|  | expect(wrapper).toMatchSnapshot(); |
|  | }); |
|  | }); |

[**view raw**](https://gist.github.com/productioncoder/0fb09562b1d5678185419b6f5c66bfd0/raw/01c1339008ab53b569f76fe7d255431f5f21c507/VideoGridHeader.unit.test.js)[**VideoGridHeader.unit.test.js**](https://gist.github.com/productioncoder/0fb09562b1d5678185419b6f5c66bfd0#file-videogridheader-unit-test-js) hosted with  by [**GitHub**](https://github.com/)

To run the tests and to have a look at the snapshots, cd into your project and run

|  |  |
| --- | --- |
|  | yarn test |

[**view raw**](https://gist.github.com/productioncoder/e45063e66017bc41a6a65e0664bc82b7/raw/5655bd139e294f3578963432e214488e63a60bf7/run-tests.sh)[**run-tests.sh**](https://gist.github.com/productioncoder/e45063e66017bc41a6a65e0664bc82b7#file-run-tests-sh) hosted with  by [**GitHub**](https://github.com/)

## 4 Wrap Up

So here’s the result of our work. Currently we’ve just done the markup. Later on, we will fill the VideoGrid component with life and display videos we fetched from the Youtube API.