# **<h1>Зачем и как тестировать фронтенд на примере библиотеки Vue Test Utils</h1>**

### Тестирование кода – один из важнейших подходов к разработке, которым должен уметь владеть каждый разработчик.

### Длительное время фронтенд ~~не~~ нуждался в тестировании, так как подавляющие число задач на JavaScript состояло в манипуляции с DOM и добавления небольшой интерактивности. Сейчас же львиная доля логических операций выполняется на клиентской стороне и в условиях расширения функционала мы должны быть уверены в нашем коде и не бояться, что добавление нового метода или класса не нарушит работу какого-либо модуля в целом.

### В данной статье мы заострим внимание на [юнит-тестировании](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), как важнейшей части процесса [непрерывной интеграции(CI)](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D1%8F), а также неотъемлемой части современной разработки.

## **<h2>Зачем нужны юнит-тесты</h2>**

* чтобы быть более уверенным, что код выполняет то, что мы от него ожидаем
* чтобы определить проблемы и дефекты на ранних стадиях разработки
* чтобы улучшить читаемость кода
* тесты - хороший способ убедиться в том, что код готов к повторному использованию
* тесты - отличная документация для вас и вашей команды, посмотрев на которые вы сможете понять назначение того или иного метода

### 

При условии соблюдения “хороших практик”, “покрыть” тестами можно подавляющую часть функционала. Этими практиками являются:

* принцип единственной ответственности
* предсказуемость
* слабая связанность(coupling)

**<h2>Какие тесты являются хорошими</h2>**

* понятные, хорошо читаемые
* изолированные
* направленные на тестирование одного поведения

## **<h2>Как следует писать тесты</h2>**

* тесты важно писать во время разработки, не после.

Этот способ является самым “дешевым”, поскольку вы существенно снижаете вероятность возвращения к старому коду, соответственно потратив как минимум в 2 раза меньше времени

* тестами следует “покрывать” публичный интерфейс.

Учитывая, что в данной статье речь идет про фронтенд разработку, публичным интерфейсом является тот, с которым взаимодействует пользователь. Тесты должны воспроизводить то, как реальные пользователи работают с вашим сайтом или приложением. Вы должны убедиться, что публичный API не сломается. То, что происходит под капотом должно проверяться косвенно, но важно только то, чтобы ваш API оставался надежным.

* тестировать необходимо отдельно логику, отдельно представление

В нашей компании для разработки фронтенда мы используем фреймворк Vue.js, поэтому в данной статье я сделаю небольшой обзор о том, как тестировать логику и представление Vue.js компонентов - декларативных переиспользуемых сущностей, в которых слои представления и управления данными разделены.

Тестирование компонентов пользовательского интерфейса - не самая простая задача, но разделение слоев дает нам возможность более точечно контролировать этот процесс.

## **<h2>Что будем использовать для тестирования</h2>**

В качестве инструментов для тестирования мы будем использовать официальную библиотеку модульного тестирования для [Vue.js](https://vuejs.org/) - [Vue Test Utils](https://vue-test-utils.vuejs.org/ru/) и Javascript-фреймворк для тестирования от Facebook - [Jest](https://jestjs.io/ru/).

Vue Test Utils позволяет монтировать компоненты Vue изолированно и имитировать взаимодействие с пользователем. В нем есть все необходимые утилиты для тестирования однофайловых компонентов, в том числе использующих [Vue Router](https://router.vuejs.org/ru/) или [Vuex](https://vuex.vuejs.org/ru/guide/).

Jest - это полнофункциональная программа для запуска тестов, которая практически не требует настройки.

Документация Vue Test Utils не рекомендует покрывать тестами каждую строку кода, поскольку это приводит к слишком большому фокусированию на деталях внутренней реализации компонентов и может привести к созданию хрупких тестов.

Как уже отмечалось ранее, один конкретный тест должен проверять, что некоторые входные данные(взаимодействие пользователя со страницей или изменение входных параметров компонента), представляемые компоненту, будут приводить к ожидаемому результату.

В данной статье мы рассмотрим простую настройку окружения для тестирования с [Vue CLI](https://cli.vuejs.org/), рассмотрим способы тестирования отрисовки компонентов, а также несколько сценариев, которые возникают во время тестирования:

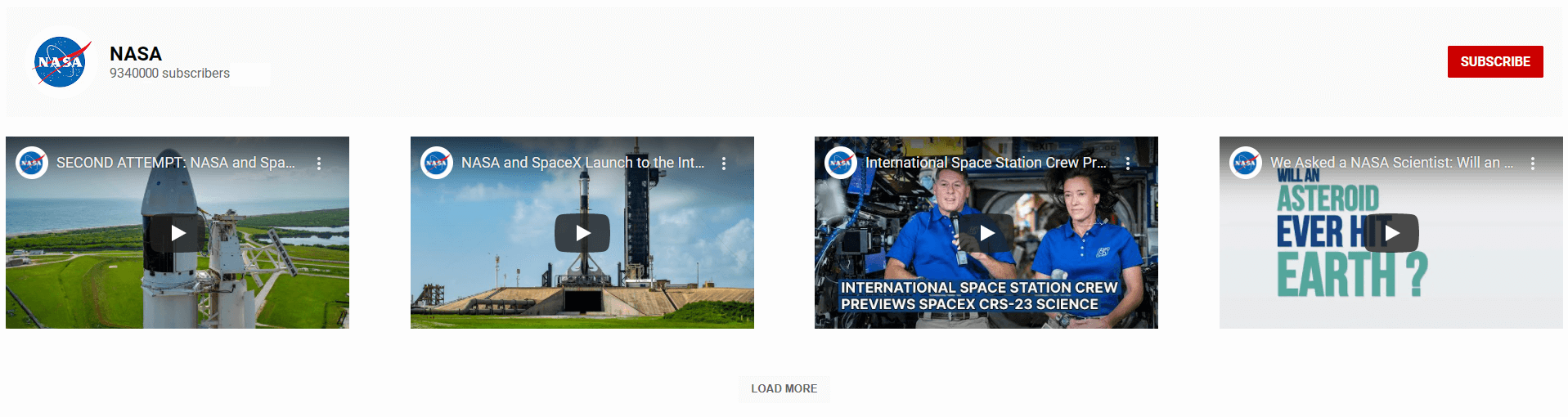
* принятие входных параметров;
* использование вычисляемых свойств;
* вызов методов;
* использование хуков жизненного цикла;
* симуляцию пользовательских действий;
* использование “моков” (фиктивных реализаций интерфейса) для API ответов.

Мы не будем напрямую тестировать вычисляемые свойства или методы. Их результат работы будет проверен путем тестирования общедоступного интерфейса.

## **<h2>Что будем тестировать</h2>**

В качестве примера будет представлен Vue.js компонент Youtube-виджета, который мы напишем с нуля. Кроме того, я вынес его на [GitHub](https://github.com/dmitriyakkerman/vue-youtube-widget), где вы можете его клонировать, а также на [NPM](https://www.npmjs.com/package/vue-youtube-widget).

Данный компонент является частичным клоном категории “Видео” внутри любого из Youtube-каналов.



Основные его функции:

* получение и возможность дополнительной загрузки видеозаписей нужного канала
* получение мета-информации канала: названия, лого, количества подписчиков
* возможность подписаться на канал

## **<h2>Установка и настройка окружения</h2>**

Данная статья подразумевает, что вы владеете базовыми знаниями и опытом настройки и создания нового проекта с использованием Vue CLI.

Давайте подготовим проект:

1. Установите глобально пакет [Vue CLI](https://cli.vuejs.org/ru/guide/installation.html), если он еще не был установлен:

npm install -g @vue/cli

1. Создайте [новый проект Vue](https://cli.vuejs.org/ru/guide/creating-a-project.html#vue-create)**:**

vue create <project-name>

1. Установите библиотеку модульного тестирования Vue приложений [Vue Test Utils](https://www.npmjs.com/package/@vue/test-utils):

npm i @vue/test-utils

1. Установите [Jest плагин](https://github.com/vuejs/vue-cli/tree/dev/packages/%40vue/cli-plugin-unit-jest) для юнит-тестирования Vue приложений:

vue add unit-jest

1. Сконфигурируйте Jest:

* в корне проекта создайте файл jest.config.js
* укажите в нем опции, которые дадут Jest понять, что для тестирования вашего приложения будет использоваться плагин для тестирования Vue приложений “@vue/cli-plugin-unit-jest”:

//*jest.config.js*

*module*.exports = {

preset: '@vue/cli-plugin-unit-jest'

}

1. Кроме того я установлю 2 пакета для использования препроцессоров SASS/SCSS:

npm install -D sass-loader node-sass

1. Так же нам необходим HTTP-клиент [axios](https://www.npmjs.com/package/axios) для создания HTTP-запросов к YouTube API 3.

npm i axios

Обратите внимание на ваш package.json файл в корне проекта.

Теперь, в опции “scripts” кроме стандартных команд “serve”, “lint” и “build” появилась еще одна - “test:unit”, именно ее мы будем использовать для тестирования.

"scripts": {

"serve": "vue-cli-service serve",

"build": "vue-cli-service build",

"test:unit": "vue-cli-service test:unit",

"lint": "vue-cli-service lint"

}

Кроме того в корне проекта сгенерировалась папка “tests/unit” с “example.spec.js” файлом для тестирования внутри.

Перед тем, как мы перейдем к написанию и тестированию нашего компонента, совершим еще несколько шагов:

* удалите компонент “HelloWorld.vue” в папке “src/components”
* удалите импорт данного компонента в файле “App.vue” в папке “src”:

//App.vue

<template>

<div id="app"></div>

</template>

<script>

export default {

name: 'App',

components: {}

}

</script>

* создайте в папке “src” файл “main.scss” со следующими стилями:

*//main.scss*

@import url('<https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap>');

.youtube-widget {

font-family: 'Roboto', sans-serif;

&\_\_header {

display: flex;

flex-wrap: wrap;

justify-content: space-between;

align-items: center;

padding: 24px;

background: #FAFAFA;

}

&\_\_info {

display: flex;

flex-wrap: wrap;

align-items: center;

}

&\_\_logo {

border-radius: 100%;

}

&\_\_data {

margin-left: 15px;

}

&\_\_title {

margin: 0;

}

&\_\_subscribers-count {

color: #606060;

}

&\_\_subscribe-btn,

&\_\_load-more-btn {

padding: 10px 16px;

border-radius: 2px;

border: none;

font-weight: bold;

text-transform: uppercase;

text-decoration: none;

}

&\_\_subscribe-btn {

background: #CC0000;

color: #FFFFFF;

}

&\_\_inner {

display: flex;

flex-wrap: wrap;

margin-top: 24px;

}

&\_\_load-more-btn {

display: block;

margin: 32px auto 0 auto;

background: #FAFAFA;

color: #606060;

cursor: pointer;

}

}

Напоминаю, что код всех файлов мы можете найти на [GitHub репозитории](https://github.com/dmitriyakkerman/vue-youtube-widget)

* импортируйте данный файл стилей в компонент

*//YouTubeWidget.vue*

<template>

<div id="app"></div>

</template>

<script>

export default {

name: 'App',

components: {}

}

</script>

<style lang="scss">

@import "../main.scss";

</style>

* удалите файл “example.spec.js” в папке “tests/unit”

## **<h2>Создание и тестирование компонента</h2>**

### **<h3>Создание компонента “YouTubeWidget”</h3>**

Приступим к созданию компонента:

1. Создайте компонент файл “YouTubeWidget.vue” в папке “src/components” со следующим содержимым:

*//YouTubeWidget.vue*

<template>

<div class="youtube-widget"></div>

</template>

<script>

export default {}

</script>

<style lang="scss"></style>

1. Импортируем его в главный файл приложения “App.vue”:

*//App.vue*

<template>

<div id="app">

<YouTubeWidget></YouTubeWidget>

</div>

</template>

<script>

import YouTubeWidget from "./components/YouTubeWidget";

export default {

name: 'App',

components: {

YouTubeWidget

}

}

</script>

### 

### 

### **<h3>Тестирование рендеринга “YouTubeWidget”</h3>**

На текущем этапе мы уже имеем возможность написать наш первый юнит-тест, который тестирует:

* является ли созданный компонент Vue-инстансом (создан ли он с помощью экземпляра объекта Vue.js)
* является ли данный компонент тем самым “YouTubeWidget.vue”

Нам необходимо создать файл с расширением “.spec.js”, только тогда Jest поймет, что данный файл создан для тестирования.

Создадим в папке “tests/unit” файл “YouTubeWidget.spec.js”.

Первое, что нам надо, импортировать 2 функции из библиотеки Vue Test Utils: [createLocalVue](https://vue-test-utils.vuejs.org/ru/api/createLocalVue.html) и [mount](https://vue-test-utils.vuejs.org/ru/api/#mount).

**createLocalVue** - возвращает класс Vue, чтобы вы могли добавлять компоненты, примеси и устанавливать плагины без загрязнения глобального класса Vue;

**mount** - создаёт обертку, которая в свою очередь содержит примонтированный и отрендеренный компонент Vue.

Мы также должны импортировать сам компонент “YouTubeWidget.vue” в “YouTubeWidget.spec.js”:

*//YouTubeWidget.spec.js*

import { *createLocalVue*, *mount* } from '@vue/test-utils';

import YouTubeWidget from '../../src/components/YouTubeWidget.vue';

Теперь мы готовы написать наш тест, соответствуя конвенции Jest и Vue Test Utils.

На данном этапе мы будет использовать 2 функции объекта-обертки [Wrapper](https://vue-test-utils.vuejs.org/ru/api/wrapper/#%D1%81%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%B0) - [isVueInstance](https://vue-test-utils.vuejs.org/ru/api/wrapper-array/#isvueinstance) и [is](https://vue-test-utils.vuejs.org/ru/api/wrapper/#is-selector).

**isVueInstance** - проверяет, что обертка является экземпляром Vue

**is** - проверяет, что обертка соответствует заданному селектору.

*//YouTubeWidget.spec.js*

import { *createLocalVue*, *mount* } from '@vue/test-utils';

import YouTubeWidget from '../../src/components/YouTubeWidget.vue';

*//используем функцию от Jest “describe”, которая группирует связанные по логике тесты в один блок, названный нами “YouTubeWidget.vue testing”*

describe('YouTubeWidget.vue testing', () => {

*//создаем новый экземпляр Vue приложения с помощью функции “createLocalVue”*

const vueInstance = createLocalVue();

*//создаем и помещаем в переменную “wrapper” обертку, в которую передаем наш компонент, дополнительно помещая в объект опций созданный экземпляр вью, чтобы смонтировать и отрендерить наш компонент во Vue-приложении*

const wrapper = *mount*(YouTubeWidget, {

vueInstance

});

*//используем функцию от Jest “it”, в которой описываем наш первый тест с двумя ожидаемыми результатами:*

*it*('initialized correctly', () => {

*//ожидаем, что созданная обертка является экземпляром Vue*

*expect*(wrapper.isVueInstance()).*toBe*(true);

*//ожидаем, что селектор компонента действительно является “YouTubeWidget”*

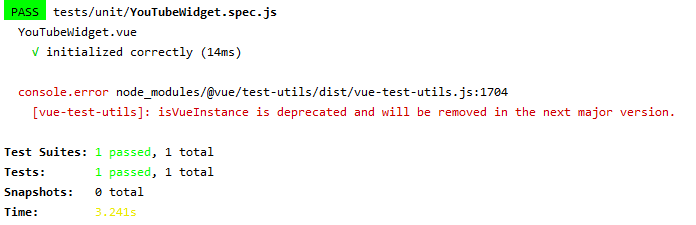
expect(wrapper.is(YouTubeWidget)).toBe(true);

});

});

Запускаем тест вводя в консоли команду:

npm run test:unit



Наш тест удачно пройден!

Кстати, с API Vue Test Utils и его методами вы можете ознакомиться [тут](https://vue-test-utils.vuejs.org/ru/api/#mount).

### **<h3>Тестирование входных параметров “YouTubeWidget”</h3>**

В предыдущей главе мы протестировали рендеринг компонента.

Перед тем, как приступить к тестированию публичного интерфейса, нам необходимо немного наполнить компонент разметкой и данными:

*//YouTubeWidget.vue*

<template>

<div class="youtube-widget">

<div class="youtube-widget\_\_header">

<div class="youtube-widget\_\_info">

<img :src="channelAvatar" alt="" class="youtube-widget\_\_logo">

<div class="youtube-widget\_\_data">

<h2 class="youtube-widget\_\_title">{{ channelTitle }}</h2>

<div class="youtube-widget\_\_subscribers-count">{{ subscribersCount }} {{ subscribersCountText }}</div>

</div>

</div>

<a :href="'http://www.youtube.com/channel/' + channelId + '?sub\_confirmation=1'" class="youtube-widget\_\_subscribe-btn">{{ subscribeBtnText }}</a>

</div>

<div class="youtube-widget\_\_inner"></div>

<button class="youtube-widget\_\_load-more-btn">{{ loadMoreBtnText }}</button>

</div>

</template>

<script>

export default {

props: {

channelId: {

type: *String*,

required: true

},

subscribersCountText: {

type: *String*,

required: false,

default: 'subscribers'

},

subscribeBtnText: {

type: *String*,

required: false,

default: 'Subscribe'

},

loadMoreBtnText: {

type: *String*,

required: false,

default: 'Load more'

}

},

data() {

return {

subscribersCount: 10000,

channelAvatar: 'https://i.ibb.co/wdbv1xY/imgonline-com-ua-Resize-w-FKj-Qa7-D6l-Y.png',

channelTitle: 'NASA',

videos: []

}

}

}

</script>

<style lang="scss">

@import "../main.scss";

</style>

В качестве канала для тестирования я выбрал канал [NASA](https://www.youtube.com/channel/UCLA_DiR1FfKNvjuUpBHmylQ). Пока мы не делаем запрос к YouTube API для получения списка видео и другой информации, а попробуем протестировать наш компонент с произвольными данными.

Для этого в методе “data” в переменные “subscribersCount”, “channelAvatar” и “channelTitle” я поместил произвольные данные, а массив “videos” оставил пустым.

*//YouTubeWidget.vue*

data() {

return {

subscribersCount: 10000,

channelAvatar: 'https://i.ibb.co/wdbv1xY/imgonline-com-ua-Resize-w-FKj-Qa7-D6l-Y.png',

channelTitle: 'NASA',

videos: []

}

}

Кроме того наш компонент принимает входные параметры, необходимые для работы компонента:

**channelId** - обязательный параметр, отвечающий за ID YouTube канала. Необходим для создания запроса к YouTube API, но пока что мы его используем только для кнопки подписки на канал, которая требует данный ID.

**subscribersCountText** - дополнительный текст количества подписчиков канала. Данный параметр я вынес в публичный интерфейс для тех случаев, когда понадобится локализировать дополнительный текст количества подписчиков.

**subscribeBtnText -** текст кнопки подписки на канал. Данный параметр я вынес в публичный интерфейс для тех случаев, когда понадобится локализировать текст кнопки подписки.

**loadMoreBtnText** - текст кнопки дополнительной загрузки видео. Данный параметр я вынес в публичный интерфейс для тех случаев, когда понадобится локализировать текст кнопки дополнительной загрузки.

Переходим в “App.vue” и передаем в компонент те параметры, которые мы ожидаем:

*//App.vue*

<template>

<div id="app">

<YouTubeWidget

channelId="UCLA\_DiR1FfKNvjuUpBHmylQ"

subscribersCountText=”подписчиков"

subscribeBtnText="Подписаться"

loadMoreBtnText="Загрузить еще"

></YouTubeWidget>

</div>

</template>

<script>

import YouTubeWidget from "./components/YouTubeWidget";

export default {

name: 'App',

components: {

YouTubeWidget

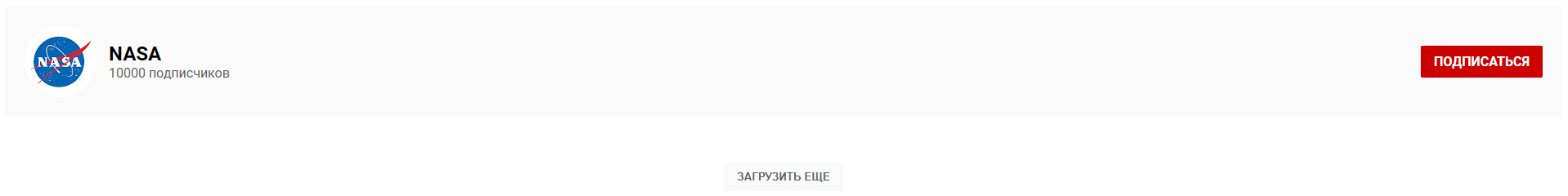
}

}

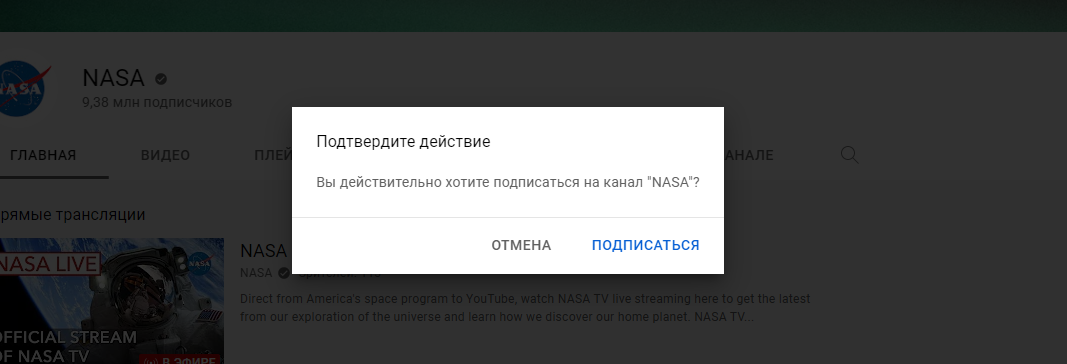
</script>

Сейчас наш компонент должен выглядеть примерно так.

Мы получили локализированные данные виджета,



а кнопка подписки при клике ведет на YouTube канал “NASA”, на котором всплывает окно подтверждения.



Теперь мы может протестировать публичный интерфейс компонента с входящими параметрами.

Первое, что нам необходимо - создать объект с моками, которые будем передавать в локально созданную во “YouTubeWidget.spec.js” обертку.

*//YouTubeWidget.spec.js*

const mockData = {

channelId: 'UCLA\_DiR1FfKNvjuUpBHmylQ',

resultsPerRequest: 1,

subscribersCountText: "подписчиков",

subscribeBtnText: "Подписаться",

loadMoreBtnText: "Загрузить еще"

};

Затем мы должны передать эти данные в объект [propsData](https://vue-test-utils.vuejs.org/ru/api/options.html#propsdata) обертки, который устанавливает входные параметры экземпляра компонента, когда он примонтирован:

*//YouTubeWidget.spec.js*

const wrapper = *mount*(YouTubeWidget, {

vueInstance,

propsData: {

channelId: mockData.channelId,

subscribersCountText: mockData.subscribersCountText,

subscribeBtnText: mockData.subscribeBtnText,

loadMoreBtnText: mockData.loadMoreBtnText

}

});

Напишем наш следующий тест, в котором используем 2 функции объекта-обертки [Wrapper](https://vue-test-utils.vuejs.org/ru/api/wrapper/#%D1%81%D0%B2%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%B0) - [props](https://vue-test-utils.vuejs.org/ru/api/wrapper/#props) и [find](https://vue-test-utils.vuejs.org/ru/api/wrapper/#find-selector):

**props** - возвращает объект с входными параметрами обертки.

**find** - возвращает первый DOM-узел или компонент Vue, соответствующего селектору.

Руководствуясь принципом единственной ответственности мы создаем новую функцию “it” внутри блока “describe”, которая будет направлена на тестирование входных параметров и их попаданию в разметку:

*//YouTubeWidget.spec.js*

*it*('renders props when passed', () => {

*//ожидаем, что входящий параметр “channelId” идентичен тому, что мы передали*

*expect*(wrapper.props().channelId).*toBe*(mockData.channelId);

*//ожидаем, что входящий параметр “subscribersCountText” идентичен тому, что мы передали*

*expect*(wrapper.props().subscribersCountText).*toBe*(mockData.subscribersCountText);

*//ожидаем, что входящий параметр “subscribeBtnText” идентичен тому, что мы передали*

*expect*(wrapper.props().subscribeBtnText).*toBe*(mockData.subscribeBtnText);

*//ожидаем, что входящий параметр “loadMoreBtnText” идентичен тому, что мы передали*

*expect*(wrapper.props().loadMoreBtnText).*toBe*(mockData.loadMoreBtnText);

*//Тут же проверим, отрисовываются ли переданные данные в разметке*

*//ожидаем, что атрибут “href” кнопки-подписки на канал идентичен тому, что мы передали*

expect(wrapper.find('.youtube-widget\_\_subscribe-btn').attributes('href')).toEqual('http://www.youtube.com/channel/' + mockData.channelId + '?sub\_confirmation=1');

*//ожидаем, что текст кнопки-подписки на канал идентичен тому, что мы передали*

*expect*(wrapper.find('.youtube-widget\_\_subscribe-btn').text()).*toEqual*(mockData.subscribeBtnText);

*//ожидаем, что текст кнопки дополнительной загрузки видео идентичен тому, что мы передали*

*expect*(wrapper.find('.youtube-widget\_\_load-more-btn').text()).*toEqual*(mockData.loadMoreBtnText);

});

Так выглядит в данный момент “YouTubeWidget.spec.js” в целом:

*//YouTubeWidget.spec.js*

import { *createLocalVue*, *mount* } from '@vue/test-utils';

import YouTubeWidget from '../../src/components/YouTubeWidget.vue';

const mockData = {

channelId: 'UCLA\_DiR1FfKNvjuUpBHmylQ',

resultsPerRequest: 1,

subscribersCountText: "подписчиков",

subscribeBtnText: "Подписаться",

loadMoreBtnText: "Загрузить еще"

};

*describe*('YouTubeWidget.vue', () => {

const vueInstance = *createLocalVue*();

const wrapper = *mount*(YouTubeWidget, {

vueInstance,

propsData: {

channelId: mockData.channelId,

subscribersCountText: mockData.subscribersCountText,

subscribeBtnText: mockData.subscribeBtnText,

loadMoreBtnText: mockData.loadMoreBtnText

}

});

*it*('initialized correctly', () => {

*expect*(wrapper.isVueInstance()).*toBe*(true);

*expect*(wrapper.is(YouTubeWidget)).*toBe*(true);

});

*it*('renders props when passed', () => {

*expect*(wrapper.props().channelId).*toBe*(mockData.channelId);

*expect*(wrapper.props().subscribersCountText).*toBe*(mockData.subscribersCountText);

*expect*(wrapper.props().loadMoreBtnText).*toBe*(mockData.loadMoreBtnText);

*expect*(wrapper.props().subscribeBtnText).*toBe*(mockData.subscribeBtnText);

*expect*(wrapper.find('.youtube-widget\_\_subscribe-btn').attributes('href')).*toEqual*('http://www.youtube.com/channel/' + mockData.channelId + '?sub\_confirmation=1');

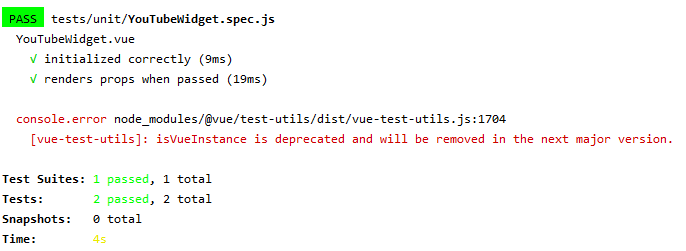
*expect*(wrapper.find('.youtube-widget\_\_subscribe-btn').text()).*toEqual*(mockData.subscribeBtnText);

*expect*(wrapper.find('.youtube-widget\_\_load-more-btn').text()).*toEqual*(mockData.loadMoreBtnText);

});

});

Запускаем тест:



Тесты проходят!

### **<h3>Тестирование асинхронной логики и хуков жизненного цикла</h3>**

Мы протестировали рендеринг компонента и его входные параметры, теперь настало время поработать с асинхронной логикой.

Сейчас мы займемся настройкой http запросов к YouTube API для получения видеозаписей канала, его названия, лого и количества подписчиков, а затем запустим тесты, чтобы удостовериться, что при определенном ответе сервера наш компонент будет работать так как мы ожидаем.

Для начала создайте в папке “src” другую папку под названием “api”, а в ней файл “YouTubeWidget.js” со следующим содержимым:

*//YouTubeWidget.js*

import axios from "axios";

export const *getChannelTitleAndAvatar* = async function(channelId, apiKey) {

const response = await axios.get(`https://www.googleapis.com/youtube/v3/channels?part=snippet&id=${channelId}&key=${apiKey}`);

return response.data;

};

export const *getSubscribersCount* = async function(channelId, apiKey) {

const response = await axios.get(`https://www.googleapis.com/youtube/v3/channels?part=statistics&id=${channelId}&key=${apiKey}`);

return response.data;

};

export const *getVideos* = async function(channelId, apiKey, resultsCount) {

const response = await axios.get(`https://www.googleapis.com/youtube/v3/search?key=${apiKey}&channelId=${channelId}&part=snippet,id&order=date&maxResults=${resultsCount}`);

return response.data;

};

export const *loadMoreVideos* = async function(channelId, apiKey, resultsCount, nextPageToken) {

const response = await axios.get(`https://www.googleapis.com/youtube/v3/search?key=${apiKey}&channelId=${channelId}&part=snippet,id&order=date&pageToken=${nextPageToken}&maxResults=${resultsCount}`);

return response.data;

};

Мы импортирует http-клиент “axios” для создания http-запросов к серверу YouTube API, а затем экспортируем во вне функции по получению:

* названия канала и его лого
* количества подписчиков канала
* видеозаписей канала
* дополнительных видеозаписей

Переходим к компоненту “YouTubeWidget.vue” и преобразуем его:

*//YouTubeWidget.vue*

<template>

<div class="youtube-widget">

<div class="youtube-widget\_\_header">

<div class="youtube-widget\_\_info">

<img :src="channelAvatar" alt="" class="youtube-widget\_\_logo">

<div class="youtube-widget\_\_data">

<h2 class="youtube-widget\_\_title">{{ channelTitle }}</h2>

<div class="youtube-widget\_\_subscribers-count">{{ subscribersCount }} {{ subscribersCountText }}</div>

</div>

</div>

<a :href="'http://www.youtube.com/channel/' + channelId + '?sub\_confirmation=1'" class="youtube-widget\_\_subscribe-btn">{{ subscribeBtnText }}</a>

</div>

<div class="youtube-widget\_\_inner">

<iframe v-for="video in videos" :key="video.id.videoId" :src="'https://www.youtube.com/embed/' + video.id.videoId" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

</div>

<button class="youtube-widget\_\_load-more-btn">{{ loadMoreBtnText }}</button>

</div>

</template>

<script>

import { getChannelTitleAndAvatar, getSubscribersCount, getVideos, loadMoreVideos } from "../api/YouTubeWidget";

export default {

props: {

apiKey: {

type: String,

required: true

},

channelId: {

type: String,

required: true

},

resultsPerRequest: {

type: Number,

required: false,

default: 1

},

subscribersCountText: {

type: String,

required: false,

default: 'subscribers'

},

subscribeBtnText: {

type: String,

required: false,

default: 'Subscribe'

},

loadMoreBtnText: {

type: String,

required: false,

default: 'Load more'

}

},

data() {

return {

subscribersCount: null,

channelAvatar: '',

channelTitle: '',

videos: [],

totalVideosCount: null,

nextPageToken: null,

isSending: false

}

},

methods: {

async getChannelTitleAndAvatar() {

this.isSending = true;

try {

const response = await getChannelTitleAndAvatar(this.channelId, this.apiKey);

this.channelAvatar = response["items"][0].snippet.thumbnails.default.url;

this.channelTitle = response["items"][0].snippet.localized.title;

}

catch(error) {

console.log(error);

}

finally {

this.isSending = false;

}

},

async getSubscribersCount() {

this.isSending = true;

try {

const response = await getSubscribersCount(this.channelId, this.apiKey);

this.subscribersCount = response["items"][0].statistics.subscriberCount;

this.totalVideosCount = response["items"][0].statistics.videoCount;

}

catch(error) {

console.log(error);

}

finally {

this.isSending = false;

}

},

async getVideos() {

this.isSending = true;

try {

const response = await getVideos(this.channelId, this.apiKey, this.resultsPerRequest);

this.videos = response.items;

this.nextPageToken = response.nextPageToken;

}

catch(error) {

console.log(error);

}

finally {

this.isSending = false;

}

}

},

mounted() {

this.getChannelTitleAndAvatar();

this.getSubscribersCount();

this.getVideos();

}

}

</script>

<style lang="scss">

@import "../main.scss";

</style>

Мы импортируем путем деструктуризации каждую из функций из “api/YouTubeWidget.js” и используем их в поле “methods”, записывая в переменные объекта “data” результат выполнения каждой из функций. Функцию “loadMoreVideos” мы так же импортируем, однако пока не используем.

Обратите внимание, во входных параметрах появились еще 2 новых:

**apiKey** - обязательный Google API ключ, необходимый для http-запросов к YouTube API

**resultsPerRequest** - то количество видеозаписей, которое мы хотели бы загрузить за 1 запрос

Отметим, что результат выполнения методов “getChannelTitleAndAvatar”, “getSubscribersCount” и

“getVideos” нам нужно получится сразу же при рендеринге компонента, именно поэтому мы их вызываем в хуке жизненного цикла [mounted](https://ru.vuejs.org/v2/api/#mounted).

Переходим в “App.vue” и преобразуем его к следующему содержимому:

//App.vue

<template>

<div id="app">

<YouTubeWidget

apiKey="ВАШ\_API\_КЛЮЧ"

channelId="UCLA\_DiR1FfKNvjuUpBHmylQ"

:resultsPerRequest="1"

subscribersCountText="подписчиков"

subscribeBtnText="Подписаться"

loadMoreBtnText="Загрузить еще"

></YouTubeWidget>

</div>

</template>

<script>

import YouTubeWidget from "./components/YouTubeWidget";

export default {

name: 'App',

components: {

YouTubeWidget

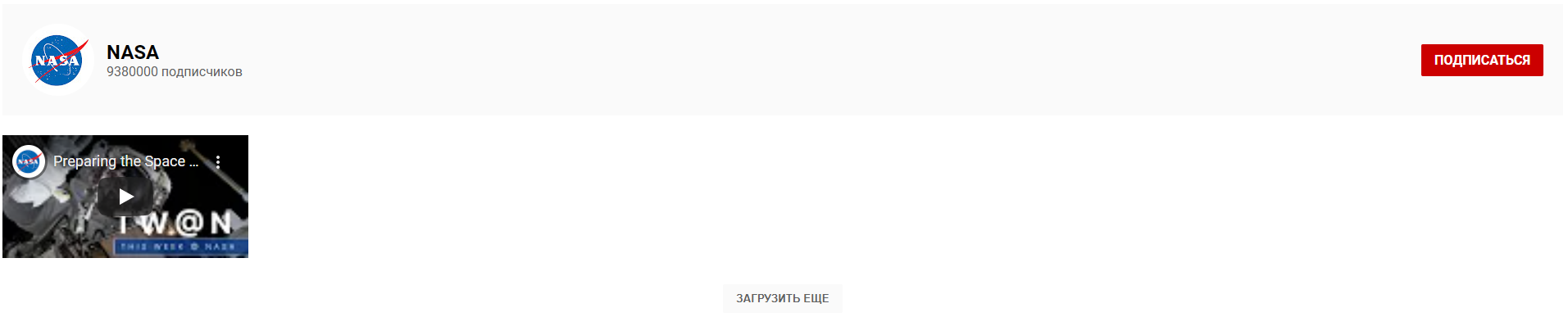
}

}

</script>

*\*Обратите внимание, что вам необходимо передать валидный Google API ключ в параметр “apiKey”*

Обратившись к YouTube API мы динамически получили название канала, лого, количество подписчиков и загрузили 1 видеозапись:



Переходим к тестированию.

Первое, что нам необходимо, преобразовать наш объект с моками, дополнив его новыми входными параметрами и данными, которые будут имитировать ответ сервера:

*//YouTubeWidget.spec.js*

const mockData = {

apiKey: 'yourapikey',

channelId: 'UCLA\_DiR1FfKNvjuUpBHmylQ',

subscribersCountToFixed: 1,

subscribersCountText: "подписчиков",

subscribeBtnText: "Подписаться",

loadMoreBtnText: "Загрузить еще",

httpResponse: {

data: {

items: [

{

id: {

videoId: '1'

},

snippet: {

thumbnails: {

default: {

url: 'avatar-url'

}

},

localized: {

title: 'NASA'

}

},

statistics: {

subscriberCount: 100,

videoCount: 100

}

}

],

nextPageToken: 'E7GFKSDGKLF22f'

}

}

};

Дополним объект входных параметров обертки:

*//YouTubeWidget.spec.js*

const wrapper = mount(YouTubeWidget, {

vueInstance,

propsData: {

apiKey: mockData.apiKey,

channelId: mockData.channelId,

subscribersCountToFixed: mockData.subscribersCountToFixed,

subscribersCountText: mockData.subscribersCountText,

subscribeBtnText: mockData.subscribeBtnText,

loadMoreBtnText: mockData.loadMoreBtnText

}

});

Далее нам необходимо задействовать специальную mock-функцию для библиотеки “axios” и ее метода “get”:

*//YouTubeWidget.spec.js*

jest.*mock*('axios', () => ({

get: () => *Promise*.resolve(mockData.httpResponse)

}));

Мы сообщаем локальному окружению, что все get-запросы должны возвращать ответ, который сохранили в объекте с моками.

Фактически, когда мы сделаем get-запрос с помощью “axios”, в ответ получим данный объект, данные которого впоследствии пробросим в представление:

*//YouTubeWidget.spec.js*

httpResponse: {

data: {

items: [

{

id: {

videoId: '1'

},

snippet: {

thumbnails: {

default: {

url: 'avatar-url'

}

},

localized: {

title: 'NASA'

}

},

statistics: {

subscriberCount: 100,

videoCount: 100

}

}

],

nextPageToken: 'E7GFKSDGKLF22f'

}

}

Перед написанием тестов отмечу важное примечание.

Во-первых, необходимо не забывать, что методы “getChannelTitleAndAvatar”, “getSubscribersCount” и “getVideos” являются асинхронными, соответственно и написание тестов должно содержать коллбэки с ключевым словом “async”.

Кроме того, важно понимать, что для таких операций как асинхронные вызовы или разрешение промисов мы должны использовать метод Vue.js [“nextTick”](https://ru.vuejs.org/v2/api/index.html#Vue-nextTick).

*//YouTubeWidget.spec.js*

import { *createLocalVue*, *mount* } from '@vue/test-utils';

import YouTubeWidget from '../../src/components/YouTubeWidget.vue';

const mockData = {

apiKey: 'ВАШ\_API\_КЛЮЧ',

channelId: 'UCLA\_DiR1FfKNvjuUpBHmylQ',

resultsPerRequest: 1,

subscribersCountToFixed: 1,

subscribersCountText: "подписчиков",

subscribeBtnText: "Подписаться",

loadMoreBtnText: "Загрузить еще",

httpResponse: {

data: {

items: [

{

id: {

videoId: '1'

},

snippet: {

thumbnails: {

default: {

url: 'avatar-url'

}

},

localized: {

title: 'NASA'

}

},

statistics: {

subscriberCount: 100,

videoCount: 100

}

}

],

nextPageToken: 'E7GFKSDGKLF22f'

}

}

};

jest.*mock*('axios', () => ({

get: () => *Promise*.resolve(mockData.httpResponse)

}));

*describe*('YouTubeWidget.vue', () => {

const vueInstance = *createLocalVue*();

const wrapper = *mount*(YouTubeWidget, {

vueInstance,

propsData: {

apiKey: mockData.apiKey,

channelId: mockData.channelId,

resultsPerRequest: mockData.resultsPerRequest,

subscribersCountText: mockData.subscribersCountText,

subscribeBtnText: mockData.subscribeBtnText,

loadMoreBtnText: mockData.loadMoreBtnText

}

});

*it*('initialized correctly', () => {

*expect*(wrapper.isVueInstance()).*toBe*(true);

*expect*(wrapper.is(YouTubeWidget)).*toBe*(true);

});

*it*('renders props when passed', () => {

*expect*(wrapper.props().apiKey).*toBe*(mockData.apiKey);

*expect*(wrapper.props().channelId).*toBe*(mockData.channelId);

*expect*(wrapper.props().resultsPerRequest).*toBe*(mockData.resultsPerRequest);

*expect*(wrapper.props().subscribersCountText).*toBe*(mockData.subscribersCountText);

*expect*(wrapper.props().subscribeBtnText).*toBe*(mockData.subscribeBtnText);

*expect*(wrapper.props().loadMoreBtnText).*toBe*(mockData.loadMoreBtnText);

*expect*(wrapper.find('.youtube-widget\_\_subscribe-btn').attributes('href')).*toEqual*('http://www.youtube.com/channel/' + mockData.channelId + '?sub\_confirmation=1');

*expect*(wrapper.find('.youtube-widget\_\_subscribe-btn').text()).*toEqual*(mockData.subscribeBtnText);

*expect*(wrapper.find('.youtube-widget\_\_load-more-btn').text()).*toEqual*(mockData.loadMoreBtnText);

});

*it*("fetches channel avatar src", async () => {

wrapper.vm.$nextTick(() => {

*//ожидаем, что после выполнения запроса к серверу, в методе дата переменная “channelAvatar” будет идентична тому, что мы передали*

*expect*(wrapper.vm.channelAvatar).*toEqual*(mockData.httpResponse.data["items"][0].snippet.thumbnails.default.url);

*//ожидаем, что после выполнения запроса к серверу, атрибут “src” элемент с селектором “.youtube-widget\_\_logo'” будет идентичен тому, что мы передали*  *expect*(wrapper.find('.youtube-widget\_\_logo').attributes('src')).*toEqual*(mockData.httpResponse.data["items"][0].snippet.thumbnails.default.url);

})

});

*it*("fetches channel title", async () => {

wrapper.vm.$nextTick(() => {

*//ожидаем, что после выполнения запроса к серверу, в методе дата переменная “channelTitle” будет идентична тому, что мы передали*

*expect*(wrapper.vm.channelTitle).*toEqual*(mockData.httpResponse.data["items"][0].snippet.localized.title);

*//ожидаем, что после выполнения запроса к серверу, текст элемента с селектором “.youtube-widget\_\_title'” будет идентичен тому, что мы передали*  *expect*(wrapper.find('.youtube-widget\_\_title').text()).*toEqual*(mockData.httpResponse.data["items"][0].snippet.localized.title);

})

});

*it*("fetches channel subscribers count", async () => {

wrapper.vm.$nextTick(() => {

*//ожидаем, что после выполнения запроса к серверу, в методе дата переменная “subscribersCount” будет идентична тому, что мы передали*

*expect*(wrapper.vm.subscribersCount).*toEqual*(mockData.httpResponse.data["items"][0].statistics.subscriberCount);

*//ожидаем, что после выполнения запроса к серверу, текст элемента с селектором “.youtube-widget\_\_subscribers-count'” будет идентичен тому, что мы передали*  *expect*(wrapper.find('.youtube-widget\_\_subscribers-count').text()).*toEqual*(*Number*(mockData.httpResponse.data["items"][0].statistics.subscriberCount) + ' ' + mockData.subscribersCountText);

})

});

*it*("fetches 1 channel video when loaded", async () => {

wrapper.vm.$nextTick(() => {

*//ожидаем, что после выполнения запроса к серверу, количество видеозаписей в переменной “videos” метода “data” будет равняться 1*

*expect*(wrapper.vm.videos.length).*toBe*(1);

*//ожидаем, что после выполнения запроса к серверу, массив видеозаписей в переменной “videos” будет идентичен тому массиву, что мы передали*

*expect*(wrapper.vm.videos).*toEqual*(mockData.httpResponse.data.items);

*//ожидаем, что после выполнения запроса к серверу, количество элементом с селектором “'iframe'” будет равняться 1*

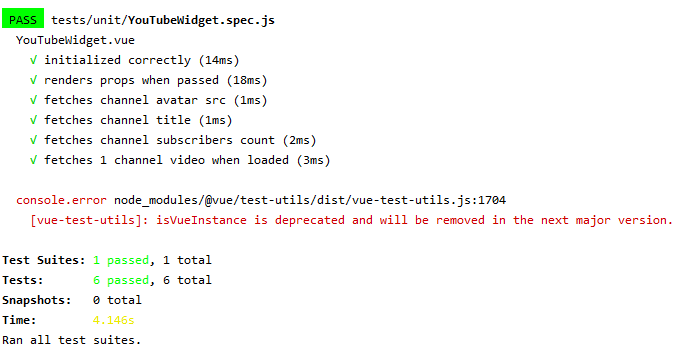
*expect*(wrapper.findAll('iframe').length).*toBe*(1);

})

});

});

Запускаем тест:



Тесты проходят!

### **<h3>Тестирование пользовательских действий</h3>**

Нам осталось протестировать пользовательские действия.

В нашем случае это будет клик по кнопке “Загрузить еще”, что спровоцирует дополнительную загрузку видеозаписей.

Давайте совсем немного обновим компонент: добавим событие клика и соответствующий метод, а в поле “methods” создадим соответствующую логику:

*//YouTubeWidget.vue*

<template>

<div class="youtube-widget">

<div class="youtube-widget\_\_header">

<div class="youtube-widget\_\_info">

<img :src="channelAvatar" alt="" class="youtube-widget\_\_logo">

<div class="youtube-widget\_\_data">

<h2 class="youtube-widget\_\_title">{{ channelTitle }}</h2>

<div class="youtube-widget\_\_subscribers-count">{{ subscribersCount }} {{ subscribersCountText }}</div>

</div>

</div>

<a :href="'http://www.youtube.com/channel/' + channelId + '?sub\_confirmation=1'" class="youtube-widget\_\_subscribe-btn">{{ subscribeBtnText }}</a>

</div>

<div class="youtube-widget\_\_inner">

<iframe v-for="*video* in videos" :key="*video*.id.videoId" :src="'https://www.youtube.com/embed/' + *video*.id.videoId" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>

</div>

<button v-if="totalVideosCount > videos.length" class="youtube-widget\_\_load-more-btn" @click="loadMore" :disabled="isSending">{{ loadMoreBtnText }}</button>

</div>

</template>

<script>

import { *getChannelTitleAndAvatar*, *getSubscribersCount*, *getVideos*, *loadMoreVideos* } from "../api/YouTubeWidget";

export default {

props: {

apiKey: {

type: *String*,

required: true

},

channelId: {

type: *String*,

required: true

},

resultsPerRequest: {

type: *Number*,

required: false,

default: 1

},

subscribersCountText: {

type: *String*,

required: false,

default: 'subscribers'

},

subscribeBtnText: {

type: *String*,

required: false,

default: 'Subscribe'

},

loadMoreBtnText: {

type: *String*,

required: false,

default: 'Load more'

}

},

data() {

return {

subscribersCount: null,

channelAvatar: '',

channelTitle: '',

videos: [],

totalVideosCount: null,

nextPageToken: null,

isSending: false

}

},

methods: {

async getChannelTitleAndAvatar() {

this.isSending = true;

try {

const response = await *getChannelTitleAndAvatar*(this.channelId, this.apiKey);

this.channelAvatar = response["items"][0].snippet.thumbnails.default.url;

this.channelTitle = response["items"][0].snippet.localized.title;

}

catch(error) {

*console*.log(error);

}

finally {

this.isSending = false;

}

},

async getSubscribersCount() {

this.isSending = true;

try {

const response = await *getSubscribersCount*(this.channelId, this.apiKey);

this.subscribersCount = response["items"][0].statistics.subscriberCount;

this.totalVideosCount = response["items"][0].statistics.videoCount;

}

catch(error) {

*console*.log(error);

}

finally {

this.isSending = false;

}

},

async getVideos() {

this.isSending = true;

try {

const response = await *getVideos*(this.channelId, this.apiKey, this.resultsPerRequest);

this.videos = response.items;

this.nextPageToken = response.nextPageToken;

}

catch(error) {

*console*.log(error);

}

finally {

this.isSending = false;

}

},

async loadMore() {

this.isSending = true;

try {

const response = await *loadMoreVideos*(this.channelId, this.apiKey, this.resultsPerRequest, this.nextPageToken);

this.videos = [...this.videos, ...response.items];

this.nextPageToken = response.nextPageToken;

}

catch(error) {

*console*.log(error);

}

finally {

this.isSending = false;

}

},

},

mounted() {

this.getChannelTitleAndAvatar();

this.getSubscribersCount();

this.getVideos();

}

}

</script>

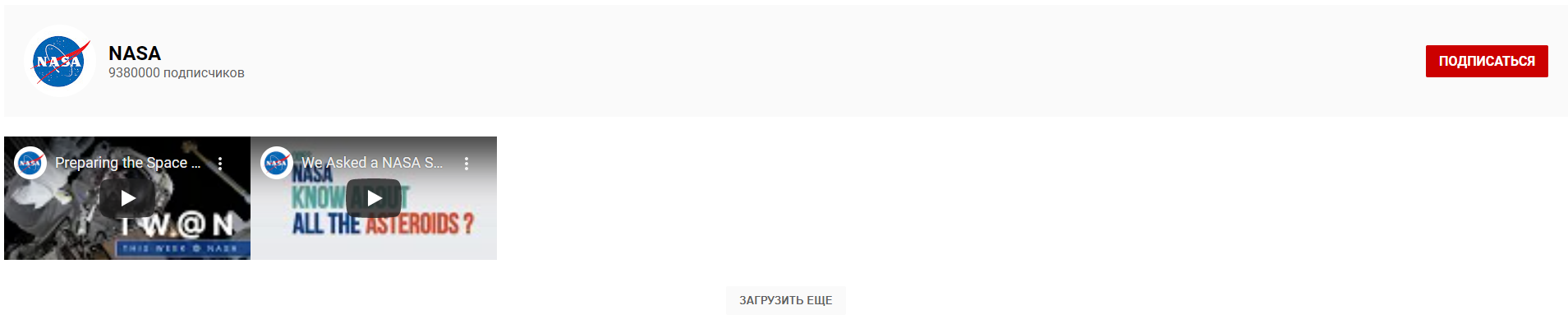
<style lang="scss">

@import "../main";

</style>

Мы добавили условное выражение в кнопке, которое говорит, что кнопка должна быть отрендерена только в том случае, если общее количество видеозаписей канала больше количества, которое уже было нами загружено. Кроме того, мы добавили атрибут “disabled” для блокировки ее нажатия во время запроса. “Вешаем” на кнопку событие “клика”, при котором вызываем метод “loadMore”, реализованный в “methods”.

Нажимаем на “Загрузить еще”:



Отлично, компонент работает! Давайте тестировать!

*//YouTubeWidget.spec.js*

import { *createLocalVue*, *mount* } from '@vue/test-utils';

import YouTubeWidget from '../../src/components/YouTubeWidget.vue';

const mockData = {

apiKey: 'ВАШ\_API\_КЛЮЧ',

channelId: 'UCLA\_DiR1FfKNvjuUpBHmylQ',

resultsPerRequest: 1,

subscribersCountToFixed: 1,

subscribersCountText: "подписчиков",

subscribeBtnText: "Подписаться",

loadMoreBtnText: "Загрузить еще",

httpResponse: {

data: {

items: [

{

id: {

videoId: '1'

},

snippet: {

thumbnails: {

default: {

url: 'avatar-url'

}

},

localized: {

title: 'NASA'

}

},

statistics: {

subscriberCount: 100,

videoCount: 100

}

}

],

nextPageToken: 'E7GFKSDGKLF22f'

}

}

};

jest.*mock*('axios', () => ({

get: () => *Promise*.resolve(mockData.httpResponse)

}));

*describe*('YouTubeWidget.vue', () => {

const vueInstance = *createLocalVue*();

const wrapper = *mount*(YouTubeWidget, {

vueInstance,

propsData: {

apiKey: mockData.apiKey,

channelId: mockData.channelId,

resultsPerRequest: mockData.resultsPerRequest,

subscribersCountText: mockData.subscribersCountText,

subscribeBtnText: mockData.subscribeBtnText,

loadMoreBtnText: mockData.loadMoreBtnText

}

});

*it*('initialized correctly', () => {

*expect*(wrapper.isVueInstance()).*toBe*(true);

*expect*(wrapper.is(YouTubeWidget)).*toBe*(true);

});

*it*('renders props when passed', () => {

*expect*(wrapper.props().apiKey).*toBe*(mockData.apiKey);

*expect*(wrapper.props().channelId).*toBe*(mockData.channelId);

*expect*(wrapper.props().resultsPerRequest).*toBe*(mockData.resultsPerRequest);

*expect*(wrapper.props().subscribersCountText).*toBe*(mockData.subscribersCountText);

*expect*(wrapper.props().subscribeBtnText).*toBe*(mockData.subscribeBtnText);

*expect*(wrapper.props().loadMoreBtnText).*toBe*(mockData.loadMoreBtnText);

*expect*(wrapper.find('.youtube-widget\_\_subscribe-btn').attributes('href')).*toEqual*('http://www.youtube.com/channel/' + mockData.channelId + '?sub\_confirmation=1');

*expect*(wrapper.find('.youtube-widget\_\_subscribe-btn').text()).*toEqual*(mockData.subscribeBtnText);

*expect*(wrapper.find('.youtube-widget\_\_load-more-btn').text()).*toEqual*(mockData.loadMoreBtnText);

});

*it*("fetches channel avatar src", async () => {

wrapper.vm.$nextTick(() => {

*expect*(wrapper.vm.channelAvatar).*toEqual*(mockData.httpResponse.data["items"][0].snippet.thumbnails.default.url);

*expect*(wrapper.find('.youtube-widget\_\_logo').attributes('src')).*toEqual*(mockData.httpResponse.data["items"][0].snippet.thumbnails.default.url);

})

});

*it*("fetches channel title", async () => {

wrapper.vm.$nextTick(() => {

*expect*(wrapper.vm.channelTitle).*toEqual*(mockData.httpResponse.data["items"][0].snippet.localized.title);

*expect*(wrapper.find('.youtube-widget\_\_title').text()).*toEqual*(mockData.httpResponse.data["items"][0].snippet.localized.title);

})

});

*it*("fetches channel subscribers count", async () => {

wrapper.vm.$nextTick(() => {

*expect*(wrapper.vm.subscribersCount).*toEqual*(mockData.httpResponse.data["items"][0].statistics.subscriberCount);

*expect*(wrapper.find('.youtube-widget\_\_subscribers-count').text()).*toEqual*(*Number*(mockData.httpResponse.data["items"][0].statistics.subscriberCount) + ' ' + mockData.subscribersCountText);

})

});

*it*('calls "loadMore" method after click on "Load More" button and fetches one more video', async () => {

*//находим по селектору кнопку*

const loadMoreBtn = wrapper.find(".youtube-widget\_\_load-more-btn");

*//имитируем событие клика пользователем*

await loadMoreBtn.trigger("click");

wrapper.vm.$nextTick(() => {

*//ожидаем, что после выполнения запроса к серверу, количество видеозаписей в переменной “videos” метода “data” будет равняться 2*

*expect*(wrapper.vm.videos.length).*toBe*(2);

*//ожидаем, что после выполнения запроса к серверу, количество элементом с селектором “'iframe'” будет равняться 2*

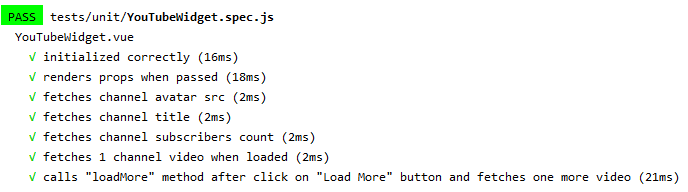
*expect*(wrapper.findAll('iframe').length).*toBe*(2);

})

});

});

Запускаем тест:



Тесты проходят!

### **<h3>Тестирование при помощи снимков</h3>**

Тестирование с использованием снимков это очень полезный инструмент в ситуациях, где вы хотите быть уверены, что ваш пользовательский интерфейс не изменяется неожиданным образом. Мы не может исключить человеческий фактор, бывают случайности, когда мы вдруг обнаружим, что структура нашего компонента не совпадает ожиданиям. Изменениями в данном случае может быть не только html-разметка, но и любые атрибуты элементов: “class”, “href”, “src” и другие.

Типичный тест снимка отображает элемент пользовательского интерфейса, совершает снимок экрана и затем сравнивает его со связанным изображением хранимым наряду с тестом. Тест провалится если два этих изображения не совпадают: либо изменение непредвиденно, либо снимок экрана нуждается в обновлении для совпадения с новой версией элемента интерфейса.

Давайте рассмотрим процесс тестирования снимками с Vue Test Utils и Jest.

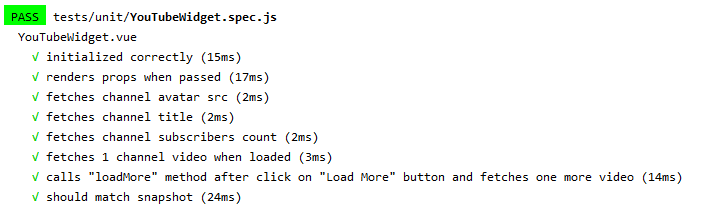
Добавим и запустим следующий тест:

*//YouTubeWidget.spec.js*

*it*('should match snapshot', () => {

*expect*(wrapper.html()).toMatchSnapshot();

})



Только что был создан снимок компонента “YouTubeWidget.vue”.

Обратите внимание на папку “tests/unit”. В ней сгенерировалась папка “\_\_snapshots\_\_” с файлом “YouTubeWidget.spec.js.snap”, который и является тем самым снимком.

Содержимое “YouTubeWidget.spec.js.snap” - строчный формат html-разметки компонента “YouTubeWidget.vue” и теперь при каждом прохождении тестов мы будем проверять - совпадает ли текущая структура html-разметки компонента с той, что мы сохранили.

В том случае, если структура различна, Vue Test Utils и Jest укажут нам на конкретное место изменений.

Если структура была изменена намеренно, мы просто удаляем старый снимок и создаем новый запуском тестов, после чего теперь идентичность структуры будет контролироваться сравнением с новым созданным снимком.

### **<h2>Вывод</h2>**

В данной статье мы рассмотрели, как на можно легко тестировать Vue.js компоненты с помощью библиотеки Vue Test Utils и фреймворка Jest. Благодаря таким инструментам тестировать фронтенд стало значительно легче и интереснее, и на самом деле они намного больше, чем я показал, поэтому рекомендую подробнее ознакомиться с их документацией <https://vue-test-utils.vuejs.org/ru/> <https://jestjs.io/uk/>

Надеюсь, данная статья стала для вас полезной!