Методическое пособие:

Знакомство с фреймворком Vue.js

Оглавление

[**1. Введение**](#_msszybe6khga) **4**

[Цель курса](#_fllkl791hghn) 4

[Для кого предназначено данное методическое пособие](#_m84iq5atzx3d) 4

[Необходимые начальные знания](#_8pofxy8nubd) 4

[**2. Экземпляр Vue**](#_1387lqm70bo2) **5**

[Цель урока](#_msrg9wsaglsl) 5

[Начальный вариант кода](#_uougg1vpb7il) 5

[Задача](#_5f3yw0e25ios) 6

[Экземпляр Vue](#_u287udbr3d24) 7

[Подключение экземпляра Vue к элементу веб-страницы](#_teyo3gccxd18) 7

[Размещение данных в экземпляре Vue](#_txyaeka9igng) 8

[Использование JavaScript-выражений в HTML-коде](#_40vk05ilxbiv) 8

[Знакомство с реактивностью](#_15cng8nrsuu4) 9

[Практическая работа №1](#_yg0ts6fdgewr) 9

[Итоги](#_9w00k13t91f) 9

[**3. Привязка атрибутов**](#_e219hqj02tku) **10**

[Цель урока](#_h1z0h451yyyf) 10

[Начальный вариант кода](#_s9yn9fwqdpe8) 10

[Задача](#_ggj14hjo0atj) 11

[Важный термин: привязка данных](#_urwr9tji448y) 12

[Решение задачи](#_djojroqg9kvu) 12

[Дополнительные варианты использования v-bind](#_21gobbdhriim) 14

[Практическая работа №2](#_uc8z4bb8sgzd) 15

[Итоги](#_156lsvuu4qxp) 15

[**4. Условный рендеринг**](#_i8fthh1m9uk8) **15**

[Цель урока](#_u9lg532hi1i2) 15

[Начальный вариант кода](#_oxncqdu8501z) 16

[Задача](#_nbathp4fn7no) 16

[Решение задачи](#_xvulbqlsrdh) 17

[Директива v-else-if](#_j0jxrpvupx6) 18

[Директива v-show](#_kaxxvu1q4g79) 19

[Практическая работа №3](#_ve61b7voaf4) 19

[Итоги](#_x1g4xdgb6prc) 19

[**5. Рендеринг списков**](#_ixika5csbg7g) **20**

[Цель урока](#_m3csqq7j8jvq) 20

[Начальный вариант кода](#_vvy9s7pfc58m) 20

[Задача](#_e5e7u4cenquq) 21

[Решение задачи](#_jpb57vgnjiwy) 21

[Перебор массива объектов](#_kc7hg7r05a04) 23

[Практическая работа №4](#_5mzmvosgppjz) 25

[Итоги](#_3f8lslwnx4w8) 25

[**6. Обработка событий**](#_6awbpqj1elyd) **25**

[Цель урока](#_uqufdrguxv20) 25

[Начальный вариант кода](#_8jw87xd0sw4x) 25

[Задача](#_992kjt9zii6i) 26

[Решение](#_ji6xbuer39qv) 27

[Задача](#_uifz5ve10gzc) 30

[Решение](#_jwksm8m1fswv) 30

[Практическая работа №5](#_jjy62cbbx2nr) 31

[Итоги](#_r9vg6ejgaa1h) 31

[**7. Привязка классов и стилей**](#_t77mtp83rexk) **31**

[Цель урока](#_h90iyigekie0) 31

[Начальный вариант кода](#_9jwtbnh0r3z4) 32

[Задача](#_42cb12sgy0dp) 33

[Решение](#_ukaglcdtz9jp) 33

[Задача](#_m4qyixvu04l9) 35

[Решение](#_l5cp87sb2qvz) 36

[Дополнительные сведения](#_rpe55v4qoxgz) 37

[Практическая работа №6](#_u36unxhnxqen) 38

[Итоги](#_vibgo3nagyce) 38

[**8. Вычисляемые свойства**](#_7qxfv5mnn6ic) **38**

[Цель урока](#_4cpu88i7vimz) 38

[Начальный вариант кода](#_clr3ozlzsav9) 38

[Задача](#_h4v6m7p945k) 40

[Решение задачи](#_4krtv6hz4dpf) 40

[Более сложный пример](#_umdk7o9ya7ub) 41

[Дополнительные сведения о вычисляемых свойствах](#_lxciwjqie5eo) 45

[Практическая работа №7](#_89i9fuhk4dfs) 45

[Итоги](#_dn6tisalftfr) 46

[**9. Компоненты**](#_5pv0f7ojyzkw) **46**

[Цель урока](#_6solyzdxpq3a) 46

[Начальный вариант кода](#_lwyq0tbb7dz3) 46

[Задача](#_hhwg79og4jz8) 48

[Решение задачи](#_65023sytav2r) 48

[Задача](#_w3tfvr96vum) 51

[Решение задачи](#_fzayo4rde7id) 52

[Практическая работа №8](#_kw3ct62xl1yt) 55

[Итоги](#_wlunzeiypmvc) 55

[**10. Пользовательские события**](#_kp0rbxjcy4op) **56**

[Цель урока](#_bm1kfu4v3ktw) 56

[Начальный вариант кода](#_s5onl07knft3) 56

[Задача](#_3ze2uwzaqkib) 59

[Решение](#_tgta8bf8sshq) 59

[Практическая работа №9](#_s726kmtui21c) 63

[Итоги](#_lhh855mpls5t) 63

[**11. Формы**](#_i4vgvd59f1i0) **64**

[Цель урока](#_e3ywu669v1xq) 64

[Начальный вариант кода](#_h73yjgs3xxnk) 64

[Задача](#_g032gwyiaja) 67

[Решение задачи](#_tpfocds6mn6g) 67

[Директива v-model](#_90bfl9a6thq1) 68

[Вывод отзывов о товаре](#_4d6c5mt2sywv) 71

[Проверка форм](#_7yzuh9txwz57) 73

[Пользовательская проверка форм](#_hcylukcp2hua) 73

[Использование модификатора .number](#_qvcanssokgqe) 76

[Практическая работа №10](#_boj6c5ud932) 76

[Итоги](#_1pjoiyb21qp7) 77

[**12. Вкладки, глобальная шина событий**](#_h6mnkolppv41) **77**

[Цель урока](#_qj2qo17l2eyu) 77

[Начальный вариант кода](#_nu43fbqj3vik) 77

[Задача](#_6xp8gm7xagsu) 82

[Решение задачи](#_4aa8fly6chs6) 82

[Создание компонента, реализующего систему вкладок](#_tqchy3d8di8b) 82

[Привязка класса к активной вкладке](#_5pyuynwo6vx3) 84

[Работа над шаблоном компонента](#_l1czg89e42d9) 85

[Вывод элементов страницы по условию](#_e025bfst2pz7) 87

[Решение проблемы с отправкой отзывов](#_1s2zuoyeteno) 89

[Рефакторинг кода проекта](#_p81xowf3414t) 89

[Причины использования стрелочной функции](#_18haad8r0e3y) 91

[Завершение работы над проектом](#_g4i4eqzaoadk) 91

[Шина событий — это не лучшее решение для обеспечения связи компонентов](#_n5scm3tzgh7y) 92

[Практическая работа №11](#_lgpkufsm6576) 92

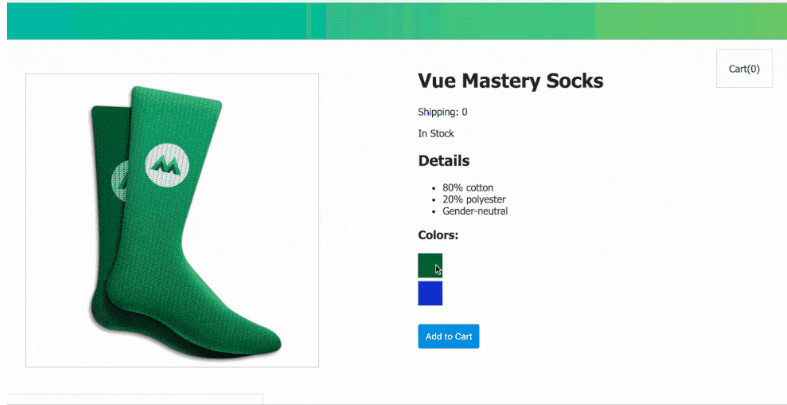
[Итоги](#_ixvjwcxcfu57) 92

# 

# 1. Введение

## Цель курса

В этом курсе мы займемся разработкой страницы, представляющей собой карточку товара (Рисунок 1.1).



**Рисунок. 1.1.** Страница, разработкой которой мы будем заниматься

## Для кого предназначено данное методическое пособие

Данное пособие предназначено для продвинутого знакомства с клиентской веб-разработкой. Представленный материал подойдет для самостоятельного изучения и использования как основы для организации учебных практик и практических занятий в учебных заведениях.

## Необходимые начальные знания

Предполагается, что читатель знаком с основами программирования на языке JavaScript, основами HTML и CSS.

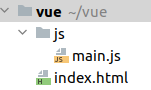
# 2. Экземпляр Vue

## Цель урока

В этом уроке мы разберёмся с тем, как использовать Vue для вывода данных на веб-странице.

## Начальный вариант кода

Первоначальная структура проекта будет выглядеть следующим образом (Рисунок 2.1).



**Рисунок. 2.1.** Первоначальная структура проекта

Мы начнём работу с очень простого HTML- и JavaScript-кода, расположенного в двух файлах (Листинг 2.1 и 2.2).

**Листинг 2.1** Файл index.html

<!doctype html>

<html lang**="ru"**>

<head>

<meta charset**="UTF-8"**>

<meta name**="viewport"**

content**="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0"**>

<meta http-equiv**="X-UA-Compatible"** content**="ie=edge"**>

<script defer src**="js/main.js"**></script>

<title>**Product App**</title>

</head>

<body>

<div id**="app"**>

<h1>**Product Name**</h1>

</div>

</body>

</html>

**Листинг 2.2** Файл main.js

let *product* = "Socks";

## Задача

Нам нужен механизм, который позволяет взять значение, имеющееся в JavaScript-коде, например то, что записано сейчас в переменную product, и поместить его в код веб-страницы, в тег h1.

Решить эту задачу нам поможет фреймворк Vue.js. Первым шагом нашей работы с Vue будет подключение фреймворка к странице. Для этого скачаем базовую версию фреймворка по ссылке (*https://ru.vuejs.org/js/vue.js*) и сохраним его в директории js (Рисунок 2.2).



**Рисунок. 2.2.** Добавляем фреймворк в проект

Затем внесем изменения в файл index.html, добавив в него, прямо над кодом подключения файла main.js (Листинг 2.3).

**Листинг 2.3** Файл index.html. Подключение фреймворка

...

<script defer src**="js/vue.js"**></script>

<script defer src**="js/main.js"**></script>

...

Далее, в main.js, вводим следующий код, убрав из него объявление переменной product (Листинг 2.4).

**Листинг 2.4** Файл main.js. Создание экземпляра фреймворка

let *app* = new Vue({

el: '#app',

data: {

product: "Socks"

}

})

Теперь нужно связать DOM с данными экземпляра Vue. Делается это с использованием особой HTML-конструкции, с помощью синтаксиса Mustache, при применении которого используются двойные фигурные скобки (Листинг 2.5).

**Листинг 2.5** Файл index.html. Вывод данных на страницу

...

<div id**="app"**>

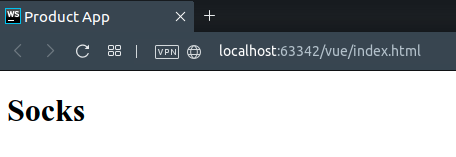
<h1>**{{ product }}**</h1>

</div>

...

JavaScript-выражение в фигурных скобках будет заменено на значение свойства product объекта data.

Вот как будет выглядеть проект в том случае, если всё работает так, как нужно (Рисунок 2.3).



**Рисунок. 2.3.** Данные перенесены из JavaScript на HTML-страницу

Как видите, нам удалось перенести данные из JavaScript-кода на HTML-страницу. А теперь давайте разберёмся в том, что мы только что сделали.

## Экземпляр Vue

Вот схема кода, с помощью которого создают экземпляр Vue (Листинг 2.6).

**Листинг 2.6** Схема создания экземпляра Vue

let *app* = new Vue({options})

Экземпляр Vue — это корневая сущность приложения. Его создают, передавая конструктору Vue объект с опциями. Этот объект содержит различные свойства и методы, которые дают экземпляру Vue возможность хранить данные и выполнять какие-то действия.

## Подключение экземпляра Vue к элементу веб-страницы

Обратите внимание на следующее свойство объекта с опциями, использованного при создании экземпляра Vue (Листинг 2.7).

**Листинг 2.7** Свойство для подключения экземпляра Vue к элементу страницы

el: '#app'

С помощью этого свойства мы подключаем экземпляр Vue к элементу нашей страницы. Благодаря этому мы создаем связь между экземпляром Vue и соответствующей частью DOM. Другими словами, мы активируем Vue в элементе <div> с идентификатором app, записывая '#app' в свойство el объекта с опциями, который был использован при создании экземпляра Vue.

## Размещение данных в экземпляре Vue

В экземпляре Vue имеется место для хранения данных. Эти данные описывают с помощью свойства data объекта с опциями (Листинг 2.8).

**Листинг 2.8** Место для хранения данных в Vue

data: {

product: "Socks"

}

К данным, хранящимся в экземпляре Vue, можно обратиться из элемента веб-страницы, к которому подключён экземпляр Vue.

## Использование JavaScript-выражений в HTML-коде

Если нам нужно, чтобы значение свойства product вывелось бы там, где выводится текст заголовка первого уровня, имя этого свойства можно поместить в двойные фигурные скобки в соответствующем теге (Листинг 2.9).

**Листинг 2.9** Вывод данных на страницу

<h1>**{{ product }}**</h1>

Фактически, речь идёт о том, что в двойных фигурных скобках находится JavaScript-выражение, результаты вычисления которого фреймворк подставляет в тег <h1> в качестве текста.

Выражения позволяют использовать значения, хранящиеся в экземпляре Vue, а так же JavaScript-конструкции, применение которых позволяет создавать какие-то новые значения.

Когда Vue видит выражение {{ product }}, он понимает, что мы ссылаемся на данные, связанные с экземпляром Vue, используя ключ product. Фреймворк заменяет имя ключа на соответствующее ему значение. В данном случае это — Socks.

Как уже было сказано, в двойных фигурных скобках можно использовать различные JavaScript-конструкции. Вот несколько примеров (Листинг 2.10).

**Листинг 2.10** Примеры выражений

{{ product + '?' }}

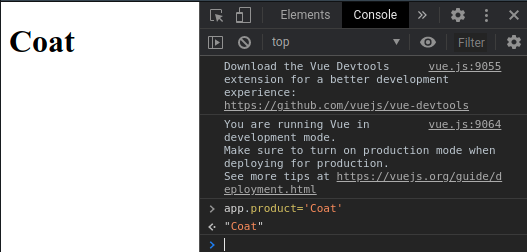
{{ firstName + ' ' + lastName }}

{{ message.split('').reverse().join('') }}

## Знакомство с реактивностью

Причина, по которой Vue сразу же после загрузки страницы выводит в теге <h1> значение, соответствующее свойству product, заключается в том, что Vue — это реактивный фреймворк. Другими словами, данные экземпляра Vue связаны со всеми местами веб-страницы, в которых есть ссылки на эти данные. В результате Vue может не только вывести данные в некоем месте страницы, но и обновить соответствующий HTML-код в том случае, если данные, на которые он ссылается, будут изменены.

Для того чтобы это доказать, давайте откроем консоль инструментов разработчика браузера и изменим значение, записанное в свойство product объекта app. Когда мы это сделаем, например, введя в консоли app.product = 'Coat', изменится и текст, выводимый на странице (Рисунок 2.4).



**Рисунок. 2.4.** Изменение значения свойства product приводит к изменению текста, выводимого на веб-странице

## Практическая работа №1

Добавьте к уже имеющимся в экземпляре Vue данным ключ description, содержащий текст A pair of warm, fuzzy socks. Затем выведите значение этого ключа в элементе <p>, который должен находиться ниже элемента <h1>.

## Итоги

* Мы узнали о том, как начать разработку Vue-приложения, создав экземпляр Vue, и о том, как выводить данные на веб-страницу.
* Экземпляр Vue является корнем каждого Vue-приложения.
* Экземпляр Vue подключается к выбранному при его создании элементу DOM.
* Данные, хранящиеся в экземпляре Vue, можно выводить на страницу, используя синтаксис Mustache, в котором используются двойные фигурные скобки, {{ }}, содержащие JavaScript-выражения.
* Vue — это реактивный фреймворк.

# 3. Привязка атрибутов

## Цель урока

Здесь мы разберёмся с тем, как, используя привязку атрибутов, вывести изображение, и задать текст атрибута alt. Соответствующие данные мы возьмём из экземпляра Vue.

## Начальный вариант кода

Начнём работу с такого HTML-кода, находящегося в файле index.html, в теге <body>(Листинг 3.1).

**Листинг 3.1** Начальный вариант кода

...

<div id**="app"**>

<div class**="product"**>

<div class**="product-image"**>

<img alt**="#"** src**=""**/>

</div>

<div class**="product-info"**>

<h1>**{{ product }}**</h1>

</div>

</div>

</div>

...

Обратите внимание на тег <div> с классом product-image. Именно в нём содержится элемент <img>, к которому мы хотим динамически привязать данные, необходимые для вывода изображения.

Элемент <div> с классом product-info используется для вывода названия товара.

Вот JavaScript-код, содержащийся в файле main.js (Листинг 3.2).

**Листинг 3.2** Начальный вариант кода

let *app* = new Vue({

el: '#app',

data: {

product: "Socks",

image: "./assets/vmSocks-green-onWhite.jpg"

}

})

Обратите внимание на то, что в объекте data теперь имеется новое свойство image, содержащее путь к изображению.

По ссылке можно скачать CSS-код и изображение, используемые в этом уроке

[*https://drive.google.com/drive/folders/1hsy1znW3Mp7k92BDUjvuyUUBPI5LEWRa?usp=sharing*](https://drive.google.com/drive/folders/1hsy1znW3Mp7k92BDUjvuyUUBPI5LEWRa?usp=sharing)

Изменим структуру проекта в соответствии с Рисунком 3.1.



**Рисунок. 3.1.** Обновленная структура проекта

Для подключения стиля к index.html нужно добавить в тег <head> следующее (Листинг 3.3).

**Листинг 3.3** Подключение файла стилей

...

<link rel**="stylesheet"** href**="css/main.css"**>

...

## Задача

Нам нужно, чтобы на странице вывелось бы изображение. При этом мы хотим динамически управлять этим изображением. То есть, нам нужна возможность, позволяющая менять путь к изображению, хранящийся в экземпляре Vue, и тут же видеть результаты этих изменений на странице. Так как именно атрибут src элемента <img> отвечает за то, какое изображение выведет элемент, нам нужно привязать некие данные к этому атрибуту. Это и позволит нам динамически, основываясь на данных, хранящихся в экземпляре Vue, менять изображение.

## Важный термин: привязка данных

Когда мы говорим о привязке данных во Vue, смысл этого заключается в том, что место в шаблоне, в котором используются или выводятся данные, напрямую «подключено», или «связано» с источником данных, то есть — с соответствующим объектом, хранящимся в экземпляре Vue.

Другими словами, сущность-источник данных, связана с сущностью, в которой эти данные используются, с приёмником данных. В нашем случае источник данных — это экземпляр Vue, а приёмник — это атрибут src элемента <img>.

## Решение задачи

Для того чтобы привязать значение свойства image из объекта с данными к свойству src тега <img>, мы воспользуемся директивой Vue v-bind. Перепишем код тега <img> из файла index.html (Листинг 3.4).

**Листинг 3.4** Привязывание значения к свойству тега

...

<img alt**="#"** v-bind:src**="image"**/>

...

Когда Vue, обрабатывая страницу, видит такую конструкцию, фреймворк заменяет её на следующий HTML-код (Листинг 3.5).

**Листинг 3.5** Генерируемый HTML-код

<img alt**="#"** src**="./assets/vmSocks-green-onWhite.jpg"**>

Если всё сделано правильно, то на странице будет выведено изображение (Рисунок 3.2).



**Рисунок. 3.2.** Изображение зеленых носков выведено на странице

А если поменять значение свойства image объекта data, то соответствующим образом изменится и значение атрибута src, что приведёт к выводу на странице нового изображения.

Предположим, нам надо заменить изображение зеленых носков на изображение синих. Для этого, учитывая то, что путь к файлу с новым изображением выглядит как ./assets/vmSocks-blue-onWhite.jpg (файл изображения можно найти по тому же адресу, что и предыдущий), достаточно привести код описания свойства image в объекте data к такому виду (Листинг 3.6).

**Листинг 3.6** Изменение изображения

...

image: "./assets/vmSocks-blue-onWhite.jpg"

...

Это приведёт к тому, что на странице появится изображение синих носков (Рисунок 3.3).



**Рисунок. 3.3.** Изображение синих носков выведено на странице

## Дополнительные варианты использования v-bind

Директиву v-bind можно использовать не только с атрибутом src. Она может помочь нам и в динамической настройке атрибута изображения alt.

Добавим в объект с опциями data новое свойство altText (Листинг 3.7)

**Листинг 3.7** Добавление нового свойства altText

...

data: {

product: "Socks",

image: "./assets/vmSocks-blue-onWhite.jpg",

altText: "A pair of socks"

}

...

Привяжем соответствующие данные к атрибуту alt, приведя код элемента <img> к такому виду (Листинг 3.8).

**Листинг 3.8** Привязка нового свойства altText

...

<img v-bind:src**="image"** v-bind:alt**="altText"** />

...

Здесь, как и в случае с атрибутом src, используется конструкция, состоящая из v-bind, двоеточия и имени атрибута (alt).

Теперь, если в данных экземпляра Vue изменятся свойства image или altText, в соответствующие атрибуты элемента <img> попадут обновленные данные. При этом связь атрибутов элемента и данных, хранящихся в экземпляре Vue, не нарушится.

Этот прием постоянно используется при разработке Vue-приложений. Из-за этого существует сокращённый вариант записи конструкции v-bind:имяатрибута. Он выглядит как :имяатрибута. Если использовать этот приём при написании кода тега <img>, то получится следующее (Листинг 3.9).

**Листинг 3.9** Сокращенный вариант привязки свойств

...

<img :src**="image"** :alt**="altText"** />

...

Это просто и удобно. Данный прием улучшает чистоту кода.

## Практическая работа №2

Добавьте на страницу ссылку (элемент <a>) с текстом More products like this. В объекте data создайте свойство link, содержащее ссылку https://www.amazon.com/s/ref=nb\_sb\_noss?url=search-alias%3Daps&field-keywords=socks. Свяжите, используя директиву v-bind, свойство link с атрибутом href элемента <a>.

## Итоги

* Данные, хранящиеся в экземпляре Vue, можно привязывать к HTML-атрибутам.
* Для привязки данных к атрибутам используется директива v-bind. Сокращённая запись этой директивы выглядит как двоеточие (:).
* Имя атрибута, которое идет за двоеточием, указывает на атрибут, к которому осуществляется привязка данных.
* В качестве значения атрибута, указываемого в кавычках, используется имя ключа, по которому можно найти данные, подключаемые к атрибуту.

# 4. Условный рендеринг

## Цель урока

Нам нужно, чтобы в карточке товара выводилась бы надпись, сообщающая посетителю о том, есть товар на складе, или нет. Если товар на складе есть, должна выводиться надпись In Stock. Если его на складе нет — надпись Out of Stock. Решение о выводе той или иной надписи должно приниматься на основе данных, хранящихся в приложении.

## Начальный вариант кода

Вот код, с которого мы начнем работу. Он, как обычно, находится в файле index.html, в теге <body> (Листинг 4.1).

**Листинг 4.1** Начальный вариант кода

...

<div id**="app"**>

<div class**="product"**>

<div class**="product-image"**>

<img :src**="image"** :alt**="altText"**/>

</div>

<div class**="product-info"**>

<h1>**{{ product }}**</h1>

</div>

</div>

</div>

...

В файле main.js, при настройке экземпляра Vue, будет применяться следующий объект с данными (Листинг 4.2).

**Листинг 4.2** Начальный вариант js кода

let *app* = new Vue({

el: '#app',

data: {

product: "Socks",

image: "./assets/vmSocks-green-onWhite.jpg",

altText: "A pair of socks",

inStock: true

}

})

Обратите внимание на то, что в объект data добавлено новое свойство. Это — свойство inStock, хранящее логическое значение true.

## Задача

При разработке веб-приложений часто бывает нужно, чтобы элемент отображался бы на странице в зависимости от выполнения некоего условия. Например, если запасы товара закончились, в карточке товара нужно об этом сообщить.

Соответствующие сообщения планируется оформить в качестве элементов <p>. Это значит, что где-то в index.html будут следующие элементы (Листинг 4.3).

**Листинг 4.3** Шаблон вывода сообщения

<p>**In Stock**</p>

<p>**Out of Stock**</p>

Наша задача заключается в том, чтобы вывести один из них в том случае, если товар на складе есть, а другой — в ситуации, когда товара на складе нет.

## Решение задачи

В Vue решение этой задачи выглядит просто и понятно.

Как вы уже знаете, данные, указывающие на наличие или отсутствие товара на складе, описаны в main.js, в объекте data (Листинг 4.4).

**Листинг 4.4** Данные, указывающие на наличие или отсутствие товара на складе

...

inStock: true

...

Для того чтобы указать системе на то, какой именно элемент <p> нужно рендерить, мы можем воспользоваться директивами v-if и v-else. Это значит, что в index.html попадёт следующее (Листинг 4.5).

**Листинг 4.5** Условный рендеринг

...

<p v-if**="inStock"**>**In Stock**</p>

<p v-else>**Out of Stock**</p>

...

Если в inStock содержится истинное значение, выведется первый элемент <p>. В противном случае будет выведен второй элемент. В нашем случае в inStock записано значение true, поэтому выведется In Stock (Рисунок 4.1).



**Рисунок. 4.1.** На складе есть запасы товара

Замечательно! Только что мы воспользовались механизмом условного рендеринга для вывода сведений о товаре. Задачу мы решили. Но не будем останавливаться на достигнутом и продолжим исследование условного рендеринга.

## Директива v-else-if

Наш механизм условного рендеринга, основанный на директивах v-if и v-else, можно расширить, добавив в него ещё один уровень логики. Сделать это можно с помощью директивы v-else-if. Для того чтобы это продемонстрировать, давайте немного усложним наш пример.

Предположим, что в объекте data, в main.js, имеются сведения о количестве товара. Они хранятся в свойстве inventory (Листинг 4.6).

**Листинг 4.6** Сведения о количестве товара

...

inventory: 100

...

Анализируя это свойство с помощью JavaScript-выражений, заключённых в кавычки, мы можем сообщать посетителям страницы более точные сведения о товаре (Листинг 4.7).

**Листинг 4.7** Условный рендеринг в зависимости от выражения

...

<p v-if**="inventory > 10"**>**In stock**</p>

<p v-else-if**="inventory <= 10 && inventory > 0"**>**Almost sold out!**</p>

<p v-else>**Out of stock**</p>

...

В данной ситуации на страницу выведется первый элемент <p>, так как соответствующее ему выражение оказывается истинным.

## Директива v-show

Если некий элемент страницы нужно часто скрывать и отображать, это значит, что для реализации этого механизма имеет смысл взглянуть на директиву v-show. Элемент с такой директивой всегда будет присутствовать в DOM, но видимым он будет только в том случае, если условие, переданное директиве, окажется истинным. Фактически, речь идёт о том, что, благодаря использованию этой директивы, к элементу, по условию, будет применяться CSS-свойство display: none.

Этот метод отличается более высокой производительностью, чем управление элементами с использованием v-if и v-else.

Вот как выглядит применение этой директивы (Листинг 4.8).

**Листинг 4.8** Использование v-show

...

<p v-show**="inStock"**>**In Stock**</p>

...

Тот вариант решения нашей задачи, в котором использовались директивы v-if и v-else, нас устраивает. Поэтому мы остановимся на нём и не будем ничего менять.

## Практическая работа №3

Добавьте в объект с данными свойство onSale. Оно должно использоваться для управления рендерингом элемента <span>, выводящего текст On Sale и сообщающего посетителям о распродаже.

## Итоги

* Существуют директивы Vue, позволяющие выводить элементы по условию:
  + v-if
  + v-else-if
  + v-else
  + v-show
* При работе с директивами можно пользоваться JavaScript-выражениями, передаваемыми им в кавычках.
* Если выражение, передаваемое директиве в кавычках, является истинным, элемент выводится.
* Директива v-show влияет только на видимость элемента, она не вставляет элементы в DOM и не удаляет элементы из DOM.

# 5. Рендеринг списков

## Цель урока

Нам нужно вывести в карточке товара дополнительные сведения о нём. Эти сведения должны быть представлены в виде списка, содержащего следующее:

* 80% cotton
* 20% polyester
* Gender-neutral

## Начальный вариант кода

Начнём работу с такого HTML-кода (файл index.html, Листинг 5.1).

**Листинг 5.1** Начальный вариант кода

...

<div id**="app"**>

<div class**="product"**>

<div class**="product-image"**>

<img :src**="image"** :alt**="altText"**/>

</div>

<div class**="product-info"**>

<h1>**{{ product }}**</h1>

<p v-if**="inStock"**>**In stock**</p>

<p v-else>**Out of Stock**</p>

</div>

</div>

</div>

...

Вот как будет выглядеть объект data, используемый при создании экземпляра Vue в main.js (Листинг 5.2)

**Листинг 5.2** Начальный вариант кода js

let *app* = new Vue({

el: '#app',

data: {

product: "Socks",

image: "./assets/vmSocks-green-onWhite.jpg",

altText: "A pair of socks",

inStock: true,

details: ['80% cotton', '20% polyester', 'Gender-neutral']

}

})

Здесь появилось новое свойство — массив details.

## Задача

Необходимо вывести на странице содержимое массива details. Для этого требуется найти ответы на вопросы о том, как перебрать массив, и о том, как визуализировать его данные.

## Решение задачи

Тут нам поможет ещё одна директива Vue — v-for. Она позволяет перебирать массивы и выводить содержащиеся в них данные.

Добавим в index.html следующий код (Листинг 5.3)

**Листинг 5.3** Рендеринг списка

...

<ul>

<li v-for**="detail in details"**>**{{ detail }}**</li>

</ul>

...

Благодаря этому на странице появится список дополнительных сведений о товаре (Рисунок 5.1).



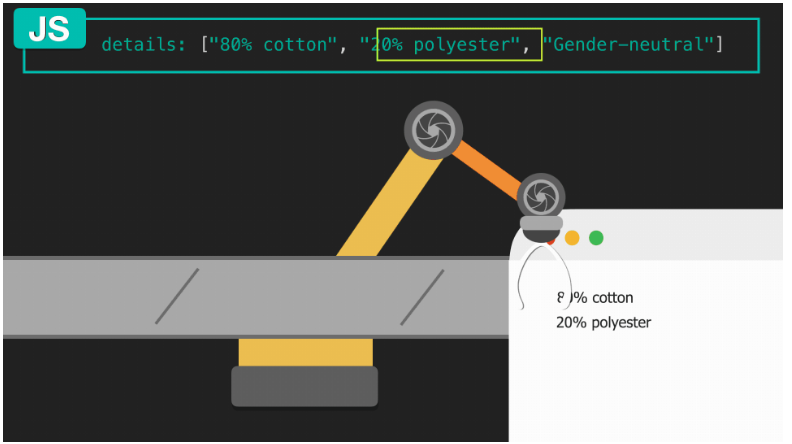
**Рисунок. 5.1.** Список на странице

Синтаксическая конструкция, используемая в кавычках вместе с директивой v-for, покажется знакомой тем, кто пользовался JavaScript-циклами for of или for in. Поговорим о том, как работает директива v-for.

Здесь мы используем существительное в единственном числе (detail) в качестве псевдонима для строковых значений, извлекаемых из массива. Затем мы пишем in и указываем имя коллекции, которую перебираем (details). В двойных фигурных скобках указывается то, какие именно данные мы хотим выводить ({{ detail }}).

Так как конструкция v-for находится внутри элемента <li>, Vue выведет новый элемент <li> для каждого элемента массива details. Если бы директива v-for использовалась внутри элемента <div>, тогда для каждого элемента массива выводился бы элемент <div>, визуализирующий значение этого элемента массива.

Директиву v-for можно представить себе в виде конвейера, на котором имеется манипулятор. Он берёт элементы коллекции, по одному за раз, и собирает список (Рисунок 5.2).



**Рисунок. 5.2.** Директива v-for похожа на конвейер

Рассмотрим ещё один пример применения v-for, более сложный. Здесь мы будем выводить в элементе <div> данные, хранящиеся в массиве объектов.

## Перебор массива объектов

Карточка товара, разработкой которой мы занимаемся, нуждается в возможности выводить сведения о разных вариантах одного и того же товара. Эти сведения содержатся в массиве объектов variants, который хранится в объекте с данными data. Как перебрать этот массив объектов для вывода данных?

Вот массив, о котором идёт речь (Листинг 5.4).

**Листинг 5.4** Массив вариантов товара

...

variants: [

{

variantId: 2234,

variantColor: 'green'

},

{

variantId: 2235,

variantColor: 'blue'

}

]

...

В объектах, которые содержатся в данном массиве, имеется название цвета и идентификатор варианта товара.

Выведем эти данные на странице (Листинг 5.5).

**Листинг 5.5** Массив вариантов товара

...

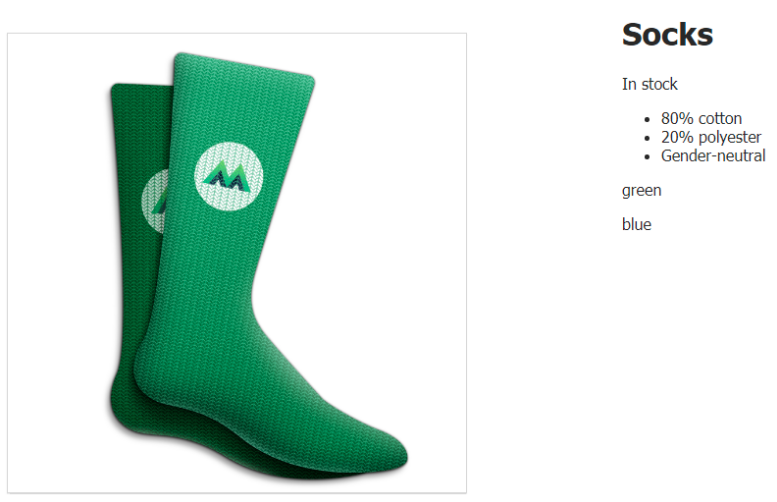
<div v-for**="variant in variants"**>

<p>**{{ variant.variantColor }}**</p>

</div>

...

Результат представлен на Рисунке 5.3.



**Рисунок. 5.3.** Список вариантов товара

Здесь нам нужно вывести на страницу лишь название цвета, соответствующее разным вариантам товара. Поэтому мы, обращаясь к элементам массива, используем точечную нотацию. Если бы мы, в фигурных скобках, написали {{ variant }}, то на страницу вывелся бы весь объект.

Обратите внимание на то, что при рендеринге подобных элементов рекомендуется использовать специальный атрибут key. Это позволяет Vue отслеживать идентичность элементов. Добавим такой атрибут в наш код, используя в качестве его значения уникальное свойство variantId объектов, содержащих сведения о вариантах товара (Листинг 5.6).

**Листинг 5.6** Добавление специального атрибута key

...

<div v-for**="variant in variants"** :key**="variant.variantId"**>

<p>**{{ variant.variantColor }}**</p>

</div>

...

## Практическая работа №4

Добавьте в объект с данными массив sizes, содержащий сведения о размерах носков, и, используя директиву v-for, выведите данные из этого массива на странице в виде списка.

Массив sizes может выглядеть так (Листинг 5.7).

**Листинг 5.7** Массив размеров

sizes: ['S', 'M', 'L', 'XL', 'XXL', 'XXXL']

## Итоги

* Директива v-for позволяет перебирать массивы для вывода содержащихся в них данных.
* В конструкции v-for для доступа к элементам массива используется псевдоним. Здесь же указывается и имя самого массива. Например, это может выглядеть так: v-for=«item in items».
* При переборе массива объектов можно использовать точечную нотацию для доступа к свойствам объектов.
* При использовании v-for рекомендуется назначать каждому выводимому элементу уникальный ключ.

# 6. Обработка событий

## Цель урока

Первая цель урока заключается в том, чтобы в карточке товара появилась бы кнопка, нажатия на которую увеличивают количество товара в корзине.

Вторая цель заключается в том, чтобы при наведении мыши на названия цветов вариантов товара менялось бы изображение товара.

## Начальный вариант кода

В файле проекта index.html будет присутствовать следующий код (Листинг 6.1).

**Листинг 6.1** Начальный вариант кода

...

<div class**="product-info"**>

<h1>**{{ product }}**</h1>

<p v-if**="inStock"**>**In stock**</p>

<p v-else>**Out of Stock**</p>

<ul>

<li v-for**="detail in details"**>**{{ detail }}**</li>

</ul>

<div v-for**="variant in variants"** :key**="variant.variantId"**>

<p>**{{ variant.variantColor }}**</p>

</div>

</div>

...

Вот содержимое main.js (Листинг 6.2).

**Листинг 6.2** Начальный вариант кода js

let *app* = new Vue({

el: '#app',

data: {

product: "Socks",

image: "./assets/vmSocks-green-onWhite.jpg",

altText: "A pair of socks",

inStock: true,

details: ['80% cotton', '20% polyester', 'Gender-neutral'],

variants: [

{

variantId: 2234,

variantColor: 'green'

},

{

variantId: 2235,

variantColor: 'blue'

}

],

}

})

## Задача

Нам нужна кнопка, которой будет назначен прослушиватель события, реагирующий на щелчок по ней. По щелчку должен запускаться метод, который и выполняет увеличение количества товара в корзине.

## Решение

Для начала — добавим, в main.js, в объект data, новое свойство, которое будет символизировать количество товара в корзине (Листинг 6.3)

**Листинг 6.3** Свойство для хранения количества товара в корзине

...

cart: 0

...

Теперь, в index.html, добавим элемент <div>, описывающий корзину. В этом элементе будет использован тег <p>, с помощью которого на страницу будет выводиться число, хранящееся в свойстве cart (Листинг 6.4).

**Листинг 6.4** Вывод количества товара на странице

...

<div class**="cart"**>

<p>**Cart({{ cart }})**</p>

</div>

...

Ещё мы создадим в коде index.html кнопку, которая позволяет добавлять товар в корзину (Листинг 6.5).

**Листинг 6.5** Кнопка для добавления товара в корзину

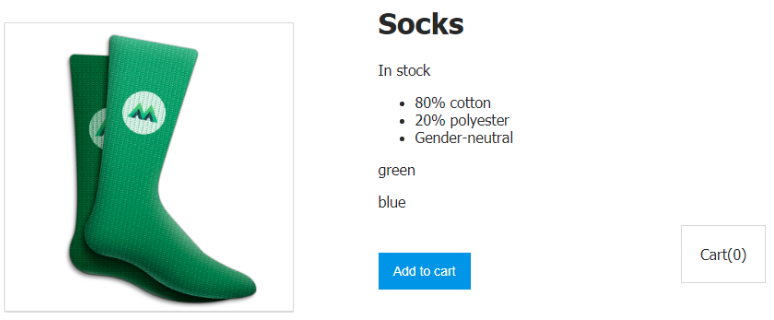
...

<button v-on:click**="cart += 1"**>**Add to cart**</button>

...

Здесь обратите внимание на то, что для инкрементирования значения, хранящегося в cart, мы используем директиву v-on.

Если теперь нажать на кнопку — количество товара в корзине увеличится на 1 (Рисунок 6.1).



**Рисунок. 6.1.** Страница с корзиной и с кнопкой для добавления товара в корзину

Как всё это работает?

Давайте разберёмся в представленной здесь конструкции. Использование директивы v-on сообщает Vue о том, что мы хотим прослушивать события, происходящие с кнопкой. Потом идет двоеточие, после которого указывается то, какое конкретно событие нас интересует. В данном случае это — событие click. В кавычках записано выражение, которое добавляет 1 к значению, хранящемуся в cart. Это происходит при каждом щелчке по кнопке.

Это — простой, но не вполне реалистичный пример. Вместо того, чтобы указывать в кавычках выражение cart += 1, давайте сделаем так, чтобы щелчок по кнопке вызывал бы метод, который будет увеличивать значение, хранящееся в cart. Вот как это выглядит (Листинг 6.6).

**Листинг 6.6** Вызов метода по щелчку

...

<button v-on:click**="addToCart"**>**Add to cart**</button>

...

Как видите, здесь addToCart — это имя метода, который будет вызван при возникновении события click. Но сам метод мы пока не объявили, поэтому давайте сделаем это прямо сейчас, оснастив им наш экземпляр Vue.

Тут используется механизм, очень похожий на тот, который мы уже применяем для хранения данных. А именно, речь идёт о том, что у объекта с опциями, используемого при создании экземпляра Vue, может быть необязательное свойство, носящее имя methods, в котором содержится объект с методами. В нашем случае это будет всего один метод — addToCart (Листинг 6.7).

**Листинг 6.7** Добавление метода

...

methods: {

addToCart() {

this.cart += 1

}

}

...

Теперь, когда мы щелкаем по кнопке, вызывается метод addToCart, который и увеличивает значение cart, выводящееся в теге <p>.

Продолжим разбор того, что здесь происходит.

Кнопка прослушивает события click благодаря директиве v-on, которая вызывает метод addToCart. Этот метод находится в свойстве methods экземпляра Vue. В теле функции содержится инструкция, добавляющая 1 к значению this.cart. Так как this хранит ссылку на то место, где хранятся данные экземпляра Vue, в котором мы находимся, функция добавляет 1 к значению cart. А this.cart — это то же самое, что и свойство cart, объявленное в свойстве data объекта с опциями.

Если бы мы просто написали бы в теле функции что-то вроде cart += 1, то мы столкнулись бы с сообщением об ошибке cart is not defined. Именно поэтому мы используем конструкцию this.cart и обращаемся к cart из экземпляра Vue, используя this.

Возможно, вы сейчас задаётесь вопросом о том, что сейчас мы просто увеличиваем количество товаров в корзине, но самого товара в корзину не добавляем. Может, мы что-то делаем не так? Это — правильный вопрос. Мы реализуем соответствующий функционал позже, в одном из следующих уроков.

Итак, теперь, когда мы изучили основы обработки событий во Vue, взглянем на более сложный пример.

Для начала — давайте расширим объекты массива variants из объекта data, добавив туда свойство variantImage, хранящее путь к изображению нужного варианта товара. Приведем соответствующий раздел файла main.js к такому виду (Листинг 6.8).

**Листинг 6.8** Изменение массива вариантов товара

...

variants: [

{

variantId: 2234,

variantColor: 'green',

variantImage: "./assets/vmSocks-green-onWhite.jpg",

},

{

variantId: 2235,

variantColor: 'blue',

variantImage: "./assets/vmSocks-blue-onWhite.jpg",

}

],

...

Теперь каждому варианту товара, зелёным и синим носкам, назначено собственное изображение.

## Задача

Нужно, чтобы, по наведению мыши на название цвета варианта носков, в поле, где выводится изображение товара, вывелось бы изображение variantImage для соответствующего цвета.

## Решение

Тут нам снова пригодится директива v-on. Но в этот раз мы воспользуемся сокращенным вариантом её записи, который выглядит как @. А прослушивать будем событие mouseover.

Вот соответствующий код в index.html (Листинг 6.9)

**Листинг 6.9** Изменение вывода списка цветов

...

<div v-for**="variant in variants"** :key**="variant.variantId"**>

<p @mouseover**="updateProduct(variant.variantImage)"**>

**{{ variant.variantColor }}**

</p>

</div>

...

Обратите внимание на то, что мы передаем методу updateProduct, в виде аргумента, variant.variantImage.

Создадим этот метод в main.js (Листинг 6.10).

**Листинг 6.10** Изменение изображения по наведению

...

updateProduct(variantImage) {

this.image = variantImage

}

...

Этот метод очень похож на тот, который мы недавно создавали для увеличения значения cart.

Но тут мы обновляем значение, хранящееся в image. А именно, в image записывается то, что хранится в variantImage того варианта товара, на который наведен указатель мыши. Соответствующее значение передается функции updateProduct из самого обработчика события, находящегося в index.html.

Другими словами, теперь метод updateProduct готов к вызову с параметром variantImage.

Когда вызывается этот метод, variant.variantImage передается ему в виде variantImage и используется для обновления значения, хранящегося в this.image. Мы, по аналогии с ранее рассмотренной конструкцией this.cart, можем сказать, что this.image — это то же самое, что image. В результате значение, хранящееся в image, теперь динамически обновляется в соответствии с данными варианта товара, на который наведен указатель мыши.

## Практическая работа №5

Создайте кнопку и соответствующий метод, которые позволят уменьшать значение, хранящееся в cart.

## Итоги

* Для организации реакции элемента на события используется директива v-on.
* Сокращённый вариант директивы v-on выглядит как @.
* При использовании v-on можно указать тип прослушиваемого события:
  + click
  + mouseover
  + любое событие DOM
* Директива v-on может вызывать методы.
* Метод, вызываемый с помощью v-on, может принимать аргументы.
* Ключевое слово this содержит ссылку на то место, где хранятся данные текущего экземпляра Vue. Его использование позволяет работать с данными экземпляра, а также с методами, объявленными в экземпляре.

# 7. Привязка классов и стилей

## Цель урока

Первой целью данного урока будет использование цвета, соответствующего вариантам товаров, для настройки свойства background-color элементов <div>, выводящих сведения об этих вариантах. Так как вариантам товара соответствуют цвета green и blue, нам нужно, чтобы один элемент <div> имел бы зеленый фоновый цвет, а второй — синий.

Вторая цель заключается в том, чтобы, используя привязку классов, отключать, по некоему условию, ненужные элементы управления.

## Начальный вариант кода

Вот как выглядит сейчас код, находящийся в index.html (Листинг 7.1).

**Листинг 7.1** Начальный вариант кода

...

<div id**="app"**>

<div class**="product"**>

<div class**="product-image"**>

<img :src**="image"** :alt**="altText"**/>

</div>

<div class**="product-info"**>

<h1>**{{ product }}**</h1>

<p v-if**="inStock"**>**In stock**</p>

<p v-else>**Out of Stock**</p>

<ul>

<li v-for**="detail in details"**>**{{ detail }}**</li>

</ul>

<div v-for**="variant in variants"** :key**="variant.variantId"**>

<p @mouseover**="updateProduct(variant.variantImage)"**>

**{{ variant.variantColor }}**

</p>

</div>

<div class**="cart"**>

<p>**Cart({{ cart }})**</p>

</div>

<button v-on:click**="addToCart"**>**Add to cart**</button>

</div>

</div>

</div>

...

Вот что сейчас находится в main.js (Листинг 7.2).

**Листинг 7.2** Начальный вариант кода js

let *app* = new Vue({

el: '#app',

data: {

product: "Socks",

image: "./assets/vmSocks-green-onWhite.jpg",

altText: "A pair of socks",

inStock: true,

details: ['80% cotton', '20% polyester', 'Gender-neutral'],

variants: [

{

variantId: 2234,

variantColor: 'green',

variantImage: "./assets/vmSocks-green-onWhite.jpg",

},

{

variantId: 2235,

variantColor: 'blue',

variantImage: "./assets/vmSocks-blue-onWhite.jpg",

}

],

cart: 0

},

methods: {

addToCart() {

this.cart += 1

},

updateProduct(variantImage) {

this.image = variantImage

}

}

})

## Задача

В предыдущем уроке мы создали обработчик событий, который меняет изображение товара, основываясь на том, на какой элемент <p> был наведен указатель мыши. Вместо того чтобы выводить название цвета в элементе <p>, мы хотели бы использовать этот цвет для настройки свойства background-color соответствующего элемента <div>. При таком подходе, вместо того, чтобы наводить мышь на тексты, мы сможем наводить ее на цветные квадраты, что приведет к выводу на странице изображения товара, цвет которого соответствует цвету, показанному в квадрате.

## Решение

Для начала — давайте добавим к элементу <div> класс color-box, который задаёт его ширину, высоту и внешний верхний отступ. Так как мы, даже сделав это, продолжаем выводить в элементах <div> слова green и blue, мы можем взять названия цветов, хранящихся в объектах, описывающих варианты товара, и использовать эти названия при привязке стиля к атрибуту style. Вот как это выглядит (Листинг 7.3).

**Листинг 7.3** Добавление класса

...

<div

class**="color-box"**

v-for**="variant in variants"**

:key**="variant.variantId"**

:style**="{ backgroundColor:variant.variantColor }"**

>

<p @mouseover**="updateProduct(variant.variantImage)"**>

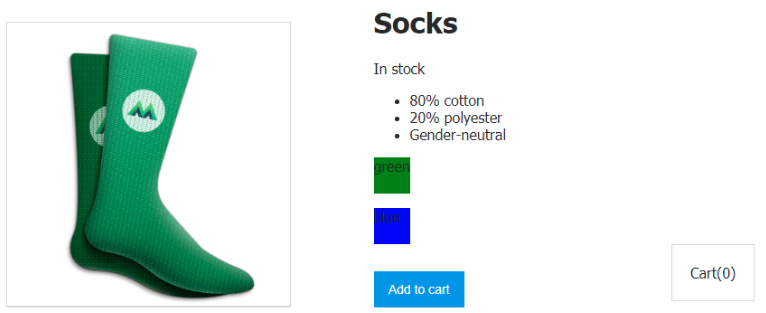
**{{ variant.variantColor }}**

</p>

</div>

...

Обратите внимание на вторую и пятую строки этого кода. Здесь мы добавляем к элементу <div> класс color-box и привязываем к нему встроенный стиль. Встроенный стиль здесь используется для динамической настройки свойства background-color элементов <div>. Цвет для фона элементов берется из variant.variantColor (Рисунок 7.1).



**Рисунок. 7.1.** Стилизованные элементы <div> и выводимые на них надписи

Теперь, когда элемент <div> стилизован с использованием variantColor, нам больше не нужно выводить в нём название цвета. Поэтому мы можем избавиться от тега <p> и переместить конструкцию @mouseover=«updateProduct(variant.variantImage)» в сам элемент <div>.

Вот как будет выглядеть код после внесения в него вышеописанных изменений (Листинг 7.4).

**Листинг 7.4** Изменение вывода списка цветов

...

<div

class**="color-box"**

v-for**="variant in variants"**

:key**="variant.variantId"**

:style**="{ backgroundColor:variant.variantColor }"**

@mouseover**="updateProduct(variant.variantImage)"**

>

</div>

...

Теперь при наведении мыши на синий квадрат на странице выводится изображение синих носков. А при наведении мыши на зелёный квадрат — изображение зеленых носков (Рисунок 7.2). Красота!



**Рисунок. 7.2.** Стилизованные элементы <div> без текста

Разобравшись с привязкой стилей, поговорим о привязке классов.

## Задача

Сейчас в наших данных есть следующее (Листинг 7.5)

**Листинг 7.5** Свойство в данных

...

inStock: true,

...

Когда свойство inStock принимает значение false, нам нужно запретить посетителям сайта щёлкать по кнопке Add to Cart, так как на складе нет товара, а значит, его нельзя добавить в корзину. К нашей удаче, существует специальный HTML-атрибут, носящий имя disabled, с помощью которого можно отключить кнопку.

Если вспомнить материал второго урока, то окажется, что мы можем воспользоваться техникой привязки атрибутов для добавления к элементу атрибута disabled тогда, когда inStock равняется false, или, скорее, в случае, когда это значение не является истинным (!inStock). Перепишем код кнопки (Листинг 7.6).

**Листинг 7.6** Отключение кнопки при отсутствии товаров

...

<button

v-on:click**="addToCart"**

:disabled**="!inStock"**

>

**Add to cart**

</button>

...

Теперь, в том случае, если в inStock записано false, кнопка работать не будет. Но её внешний вид не изменится. Другими словами, кнопка всё ещё будет выглядеть так, будто на неё можно нажать, несмотря на то, что на самом деле нажимать на неё бессмысленно.

## Решение

Тут мы поступим, действуя по той же схеме, по которой действовали, привязывая inStock к атрибуту disabled. А именно, будем привязывать класс disabledButton к нашей кнопке в случаях, когда inStock хранит false. При таком подходе, если по кнопке будет бессмысленно щёлкать, то и выглядеть она будет соответственно (Листинг 7.7).

**Листинг 7.7** Отключение кнопки при отсутствии товаров с изменением отображения

...

<button

v-on:click**="addToCart"**

:disabled**="!inStock"**

:class**="{ disabledButton: !inStock }"**

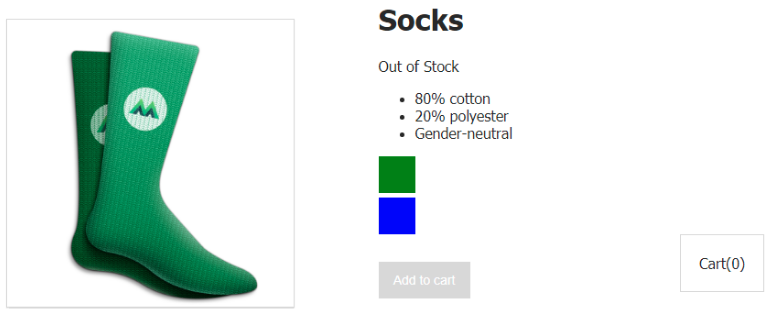
>

**Add to cart**

</button>

...

Как видите, теперь кнопка становится серой в том случае, если inStock равняется false (Рисунок 7.3).



**Рисунок. 7.3.** Отключенная кнопка выглядит так, как нужно

Давайте разберёмся в том, что здесь происходит.

Взгляните на эту строку (Листинг 7.8)

**Листинг 7.8** Привязка класса

...

:class**="{ disabledButton: !inStock }"**

**...**

Здесь мы используем сокращённый вариант записи директивы v-bind (:) для организации привязки данных к атрибуту class кнопки. В фигурных скобках мы определяем присутствие класса disabledButton на основании истинности свойства inStock.

Другими словами, когда товара на складе нет (!inStock), к кнопке добавляется класс disabledButton. Так как этот класс задаёт серый фоновый цвет кнопки, кнопка становится серой.

Замечательно! Только что мы скомбинировали наши новые знания, касающиеся привязки классов, со знаниями о привязке атрибутов, и смогли отключить кнопку и делать её серой в том случае, если inStock равняется false.

## Дополнительные сведения

К элементу можно привязывать объект классов или массив классов (Листинг 7.9).

**Листинг 7.9** Привязка объекта классов и массива классов

<div :class**="classObject"**></div>

<div :class**="[activeClass, errorClass]"**></div>

## Практическая работа №6

Когда в inStock записано значение false, нужно привязать к тегу <p>, выводящему текст Out of Stock, класс, который добавляет к элементу стиль text-decoration: line-through, перечёркивая текст.

## Итоги

* Данные можно привязывать к атрибуту элементов style.
* Данные можно привязывать к атрибуту элементов class.
* При организации привязки классов можно пользоваться выражениями, от вычисления которых зависит то, будет ли соответствующий класс привязан к элементу.

# 8. Вычисляемые свойства

## Цель урока

Нашей основной целью является вывод данных, описываемых свойствами объекта с данными brand и product, в виде единой строки.

## Начальный вариант кода

Вот код, находящийся в index.html, в теге <body>, с которого мы начнем работу (Листинг 8.1).

**Листинг 8.1** Начальный вариант кода

...

<div id**="app"**>

<div class**="product"**>

<div class**="product-image"**>

<img :src**="image"** :alt**="altText"**/>

</div>

<div class**="product-info"**>

<h1>**{{ product }}**</h1>

<p v-if**="inStock"**>**In stock**</p>

<p v-else>**Out of Stock**</p>

<ul>

<li v-for**="detail in details"**>**{{ detail }}**</li>

</ul>

<div

class**="color-box"**

v-for**="variant in variants"**

:key**="variant.variantId"**

:style**="{ backgroundColor:variant.variantColor }"**

@mouseover**="updateProduct(variant.variantImage)"**

>

</div>

<div class**="cart"**>

<p>**Cart({{ cart }})**</p>

</div>

<button

v-on:click**="addToCart"**

:disabled**="!inStock"**

:class**="{ disabledButton: !inStock }"**

>

**Add to cart**

</button>

</div>

</div>

</div>

...

Вот код main.js (Листинг 8.2)

**Листинг 8.2** Начальный вариант кода js

let *app* = new Vue({

el: '#app',

data: {

product: "Socks",

brand: 'Vue Mastery',

image: "./assets/vmSocks-green-onWhite.jpg",

altText: "A pair of socks",

inStock: false,

details: ['80% cotton', '20% polyester', 'Gender-neutral'],

variants: [

{

variantId: 2234,

variantColor: 'green',

variantImage: "./assets/vmSocks-green-onWhite.jpg",

},

{

variantId: 2235,

variantColor: 'blue',

variantImage: "./assets/vmSocks-blue-onWhite.jpg",

}

],

cart: 0

},

methods: {

addToCart() {

this.cart += 1

},

updateProduct(variantImage) {

this.image = variantImage

}

}

})

Обратите внимание на то, что в объект с данными добавлено новое свойство с именем brand.

## Задача

Нам надо, чтобы то, что хранится в brand и в product, было бы скомбинировано в одну строку. Другими словами, нам нужно вывести в теге <h1> текст Vue Mastery Socks, а не просто Socks. Для решения этой задачи нужно задаться вопросом о том, как можно конкатенировать два строковых значения, хранящихся в экземпляре Vue.

## Решение задачи

Мы для решения этой задачи воспользуемся вычисляемыми свойствами. Так как эти свойства не хранят значения, а вычисляют их, давайте добавим в объект с опциями, используемый при создании экземпляра Vue, свойство computed и создадим вычисляемое свойство с именем title (Листинг 8.3).

**Листинг 8.3** Добавление вычисляемого свойства

...

computed: {

title() {

return this.brand + ' ' + this.product;

}

}

...

Полагаем, тут всё устроено очень просто и понятно. Когда вызывается метод title(), он выполняет конкатенацию строк brand и product, после чего возвращает полученную в результате новую строку.

Теперь нам осталось лишь вывести title в теге <h1> нашей страницы.

Сейчас этот тег выглядит так (Листинг 8.4).

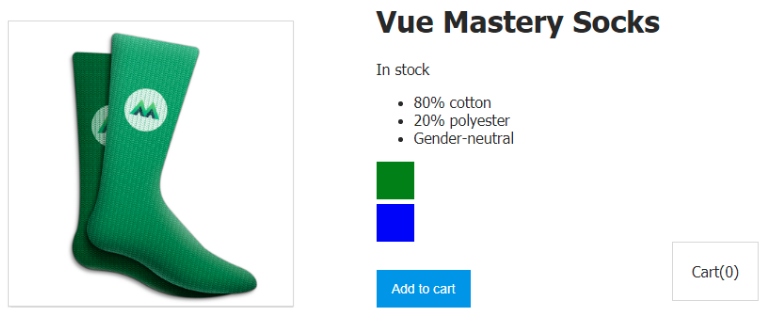
**Листинг 8.4** Вывод вычисляемого свойства

...

<h1>**{{ title }}**</h1>

...

Взглянем на страницу и проверим работоспособность того, что мы только что сделали (Рисунок 8.1).



**Рисунок. 8.1.** Заголовок страницы изменился

Как видно, в заголовке выводится Vue Mastery Socks, а это значит, что мы всё сделали правильно.

Мы взяли два значения из данных экземпляра Vue и создали на их основе новое значение. Если значение brand когда-нибудь будет обновлено, например — в это свойство окажется записанной строка Vue Craftery, то вносить какие-то изменения в код вычисляемого свойства не потребуется. Это свойство будет продолжать возвращать корректную строку, которая теперь будет выглядеть как Vue Craftery Socks. В вычисляемом свойстве title всё ещё будет использоваться свойство brand, так же, как и раньше, но теперь в brand будет записано новое значение.

Это был очень простой пример, но пример, вполне применимый на практике. Давайте теперь рассмотрим более сложный вариант использования вычисляемых свойств.

## Более сложный пример

Сейчас мы обновляем изображение, выводимое на странице, используя метод updateProduct. Мы передаем ему variantImage, а затем записываем в свойство image то, что попало в метод после наведения мыши на соответствующий цветной квадрат. Соответствующий код выглядит так (Листинг 8.5).

**Листинг 8.5** Метод updateProduct

...

updateProduct(variantImage) {

this.image = variantImage

}

...

Этот механизм работает нормально, но если нам понадобится, основываясь на том, на какой именно цветной квадрат наведена мышь, менять не только изображение, но и что-то ещё, это будет означать необходимость рефакторинга данного кода. Давайте этим и займёмся.

А именно, вместо того, чтобы хранить в данных свойство image, заменим его на свойство selectedVariant. Инициализируем его значением 0 (Листинг 8.6).

**Листинг 8.6** Свойство image заменим на selectedVariant

...

selectedVariant: 0,

...

Почему 0? Дело в том, что мы планируем устанавливать это свойство на основе индекса (index) элемента, над которым находится указатель мыши. Мы можем добавить индекс в конструкцию v-for (Листинг 8.7).

**Листинг 8.7** Добавление индекса в отображения списка

<div

class**="color-box"**

v-for**="(variant, index) in variants"**

:key**="variant.variantId"**

:style**="{ backgroundColor:variant.variantColor }"**

@mouseover**="updateProduct(variant.variantImage)"**

></div>

</div>

Обратите внимание на то, что там, где раньше была конструкция v-for=«variant in variants», теперь находится код v-for=»(variant, index) in variants».

Теперь, вместо того, чтобы передавать variant.variantImage в updateProduct, передадим в этот метод index (Листинг 8.8).

**Листинг 8.8** Передача индекса в метод updateProduct

...

@mouseover**="updateProduct(index)"**

**...**

Теперь займемся кодом метода updateProduct. Здесь мы получаем индекс. И, вместо записи нового значения в this.image, запишем index в this.selectedVariant. То есть, в selectedVariant попадёт значение index, соответствующее тому квадрату, на который был наведен указатель мыши. Ещё мы, в целях отладки, поместим в этот метод команду для логирования значения index (Листинг 8.9).

**Листинг 8.9** Изменение метода updateProduct

...

updateProduct(index) {

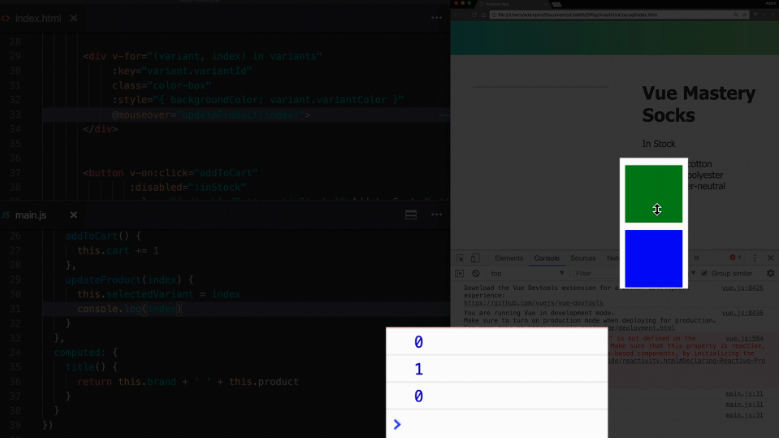
this.selectedVariant = index;

*console*.log(index);

}

...

Если сейчас обновить страницу и открыть консоль инструментов разработчика, мы можем убедиться в том, что при наведении мыши на квадраты в консоль попадают значения 0 и 1 (Рисунок 8.2).



**Рисунок. 8.2.** Проверка работоспособности созданного нами механизма

Однако изображение теперь на странице не выводится. В консоли появляется предупреждение.

Дело тут в том, что мы удалили свойство image, заменив его свойством selectedValue, но это свойство в нашем приложении все еще используется. Давайте исправим проблему, вернув image в экземпляр Vue, но на этот раз — в виде вычисляемого свойства. Соответствующий код будет выглядеть так (Листинг 8.10)

**Листинг 8.10** Добавление image в виде вычисляемого свойства

...

image() {

return this.variants[this.selectedVariant].variantImage;

}

...

Здесь мы возвращаем свойство variantImage элемента массива this.variants[this.selectedVariant]. В качестве индекса, по которому осуществляется доступ к элементу массива, используется свойство this.selectedVariant, которое равняется 0 или 1. Это, соответственно, даёт нам доступ к первому или ко второму элементу массива.

Если теперь обновить страницу, изображение выведется и будет реагировать на наведение мыши на цветные квадраты. Но теперь этот механизм реализован с использованием вычисляемого свойства.

Сейчас, когда мы подвергли рефакторингу код метода updateProduct, который теперь обновляет состояние свойства selectedVariant, мы можем поработать и с другими данными, хранящимися в объектах из массива variants, с такими, как поле variantQuantity, которое мы сейчас добавим в объект (Листинг 8.11).

**Листинг 8.11** Добавление свойства variantQuantity

...

variants: [

{

variantId: 2234,

variantColor: 'green',

variantImage: "./assets/vmSocks-green-onWhite.jpg",

variantQuantity: 10

},

{

variantId: 2235,

variantColor: 'blue',

variantImage: "./assets/vmSocks-blue-onWhite.jpg",

variantQuantity: 0

}

],

...

Давайте избавимся от обычного свойства inStock и, как и при работе со свойством image, создадим новое вычисляемое свойство с тем же именем, значение, возвращаемое которым, будет основываться на selectedVariant и variantQuantity (Листинг 8.12).

**Листинг 8.12** Добавление свойства inStock в виде вычисляемого

...

inStock(){

return this.variants[this.selectedVariant].variantQuantity

}

...

Это свойство очень похоже на вычисляемое свойство image. Но теперь мы берём из соответствующего объекта не свойство variantImage, а свойство variantQuantity.

Если теперь навести указатель мыши на квадрат, количество товара, соответствующее которому, равняется нулю, в inStock попадёт 0, а 0 является в JavaScript значением, приводимым к логическому значению false. Из-за этого на странице будет выведено сообщение Out of Stock.

Обратите внимание на то, что кнопка тоже, как и ранее, правильно реагирует на установку inStock в 0.

Почему всё продолжает правильно работать? Дело в том, что inStock всё ещё используется для привязки класса disableButton к нашей кнопке. Единственное различие нового варианта приложения и его предыдущего варианта заключается в том, что теперь inStock — это вычисляемое, а не обычное свойство.

## Дополнительные сведения о вычисляемых свойствах

Вычисляемые свойства кешируются. То есть — результаты вычисления этих свойств сохраняются в системе до тех пор, пока не изменятся данные, от которых зависят эти результаты. В результате, когда изменится variantQuantity, кеш будет очищен. А когда к inStock обратятся в следующий раз, свойство вернет новый результат, который и будет помещен в кеш.

Учитывая это, можно сказать, что если для получения некоего значения требуются ресурсоемкие вычисления, то для их выполнения выгоднее использовать вычисляемое свойство, а не метод. Метод придётся вызывать каждый раз, когда будет нужно соответствующее значение.

Кроме того, важно помнить о том, что в коде вычисляемых свойств не следует менять данные, хранящиеся в экземпляре Vue. В этом коде нужно лишь выполнять вычисления, основанные на существующих данных. Эти функции должны быть чистыми, лишенными побочных эффектов.

## Практическая работа №7

Добавьте в объект с данными, используемый при создании экземпляра Vue, новое логическое свойства onSale. Оно будет указывать на то, проводится ли распродажа. Создайте вычисляемое свойство sale, которое, на основе brand, product и onSale формирует строку, сообщающую о том, проводится ли сейчас распродажа или нет. Выведите эту строку в карточке товара.

## Итоги

* Вычисляемые свойства вычисляют значения, а не хранят их.
* Вычисляемые свойства могут использовать данные, которые хранятся в приложении, для создания на их основе новых данных.

# 9. Компоненты

## Цель урока

Основная цель данного урока — создание нашего первого компонента и исследование механизмов передачи данных в компоненты.

## Начальный вариант кода

Вот код файла index.html, находящийся в теге <body>, с которого мы начнем работу (Листинг 9.1).

**Листинг 9.1** Начальный вариант кода

...

<div id**="app"**>

<div class**="product"**>

<div class**="product-image"**>

<img :src**="image"** :alt**="altText"**/>

</div>

<div class**="product-info"**>

<h1>**{{ title }}**</h1>

<p v-if**="inStock"**>**In stock**</p>

<p v-else>**Out of Stock**</p>

<ul>

<li v-for**="detail in details"**>**{{ detail }}**</li>

</ul>

<div

class**="color-box"**

v-for**="(variant, index) in variants"**

:key**="variant.variantId"**

:style**="{ backgroundColor:variant.variantColor }"**

@mouseover**="updateProduct(index)"**

></div>

</div>

<div class**="cart"**>

<p>**Cart({{ cart }})**</p>

</div>

<button

v-on:click**="addToCart"**

:disabled**="!inStock"**

:class**="{ disabledButton: !inStock }"**

>

**Add to cart**

</button>

</div>

</div>

</div>

...

Вот код main.js (Листинг 9.2).

**Листинг 9.2** Начальный вариант кода js

let *app* = new Vue({

el: '#app',

data: {

product: "Socks",

brand: 'Vue Mastery',

selectedVariant: 0,

altText: "A pair of socks",

details: ['80% cotton', '20% polyester', 'Gender-neutral'],

variants: [

{

variantId: 2234,

variantColor: 'green',

variantImage: "./assets/vmSocks-green-onWhite.jpg",

variantQuantity: 10

},

{

variantId: 2235,

variantColor: 'blue',

variantImage: "./assets/vmSocks-blue-onWhite.jpg",

variantQuantity: 0

}

],

cart: 0

},

methods: {

addToCart() {

this.cart += 1

},

updateProduct(index) {

this.selectedVariant = index;

*console*.log(index);

}

},

computed: {

title() {

return this.brand + ' ' + this.product;

},

image() {

return this.variants[this.selectedVariant].variantImage;

},

inStock(){

return this.variants[this.selectedVariant].variantQuantity

}

}

})

## Задача

Нам не нужно, чтобы во Vue-приложении все данные, методы, вычисляемые свойства размещались бы в корневом экземпляре Vue. Со временем это приведёт к появлению кода, который будет очень тяжело поддерживать. Вместо этого нам хотелось бы разбить код на модульные части, с которыми будет проще работать, и которые сделают разработку более гибкой.

## Решение задачи

Начнём с того, что возьмем существующий код и перенесем его в новый компонент.

Вот как в файле main.js регистрируется компонент (Листинг 9.3).

**Листинг 9.3** Регистрация нового компонента

...

Vue.component('product', {})

Первый аргумент — это выбранное нами имя компонента. Второй — это объект с опциями, похожий на тот, который мы использовали при создании экземпляра Vue на прошлых занятиях.

В экземпляре Vue мы использовали свойство el для организации его привязки к элементу DOM. В случае с компонентом используется свойство template, которое определяет HTML-код компонента.

Опишем шаблон компонента в объекте с опциями (Листинг 9.4).

**Листинг 9.4** Передача опций в компонент

Vue.component('product', {

template: `

<div class="product">

// Здесь будет весь HTML-код, который раньше был в элементе с классом product

</div>

`

})

Во Vue есть несколько способов создания шаблонов. Сейчас мы пользуемся шаблонным литералом, содержимое которого заключено в обратные кавычки.

Если окажется так, что код шаблона не будет размещаться в единственном корневом элементе, в таком, как элемент <div> с классом product, это приведёт к выводу такого сообщения об ошибке:

*Component template should contain exactly one root element*

Другими словами, шаблон компонента может возвращать только один элемент.

Теперь, когда в шаблоне находится HTML-код, который раньше был в файле index.html, мы можем добавить в компонент данные, методы, вычисляемые свойства, которые раньше были в корневом экземпляре Vue (Листинг 9.5).

**Листинг 9.5** Добавление в компонент данных, методов и т.д.

Vue.component('product', {

template: `

<div class="product">

...

</div>

`,

data() {

return {

// тут будут данные

}

},

methods: {

// тут будут методы

},

computed: {

// тут будут вычисляемые свойства

}

})

...

Как видите, структура этого компонента практически полностью совпадает со структурой экземпляра Vue, с которым мы работали раньше. А вы обратили внимание на то, что data — это теперь не свойство, а метод объекта с опциями? Почему это так?

Дело в том, что компоненты часто создают, планируя использовать их многократно. Если у нас будет много компонентов product, нам нужно обеспечить то, чтобы для каждого из них создавались бы собственные экземпляры сущности data. Так как data — это теперь функция, которая возвращает объект с данными, каждый компонент гарантированно получит собственный набор данных. Если бы сущность data не была бы функцией, то каждый компонент product, везде, где использовались бы такие компоненты, содержал бы одни и те же данные. А это противоречит идее многократного использования компонентов.

Теперь, когда мы переместили код, связанный с товаром, в собственный компонент product, код описания корневого экземпляра Vue будет выглядеть так (Листинг 9.6)

**Листинг 9.6** Очищенный код корневого экземпляра Vue

...

let *app* = new Vue({

el: '#app',

})

Сейчас нам осталось лишь разместить компонент product в коде файла index.html. Это будет выглядеть так (Листинг 9.7).

**Листинг 9.7** Вызов компонента в HTML

...

<div id**="app"**>

<product></product>

</div>

...

Если теперь перезагрузить страницу приложения — она примет прежний вид (Рисунок 9.1).



**Рисунок. 9.1.** Страница приложения

А теперь, просто чтобы продемонстрировать возможности многократного использования компонентов, давайте добавим в код index.html ещё пару компонентов product. Собственно говоря, именно так организовано многократное использование компонентов. Код index.html будет выглядеть так (Листинг 9.8).

**Листинг 9.8** Вызов множества компонентов в HTML

...

<div id**="app"**>

<product></product>

<product></product>

<product></product>

</div>

...

А на странице будет выведено три копии карточки товара.

Обратите внимание на то, что в дальнейшем мы будем работать с одним компонентом product, поэтому код index.html будет выглядеть так (Листинг 9.7).

## Задача

В приложениях часто нужно, чтобы компоненты принимали бы данные, входные параметры, от родительских сущностей. В данном случае родителем компонента product является сам корневой экземпляр Vue.

Пусть в корневом экземпляре Vue имеется описание неких данных. Эти данные указывают на то, является ли пользователь обладателем премиум-аккаунта. Код описания экземпляра Vue при этом может выглядеть так (Листинг 9.9).

**Листинг 9.9** Изменение опций экземпляра Vue

...

let *app* = new Vue({

el: '#app',

data: {

premium: true

}

})

Давайте решим, что премиум-пользователям полагается бесплатная доставка.

Это означает, что нам нужно, чтобы компонент product выводил бы, в зависимости от того, что записано в свойство premium корневого экземпляра Vue, разные сведения о стоимости доставки.

Как отправить данные, хранящиеся в свойстве premium корневого экземпляра Vue, дочернему элементу, которым является компонент product?

## Решение задачи

Во Vue, для передачи данных от родительских сущностей дочерним, применяется свойство объекта с опциями props, описываемое у компонентов. Это объект с описанием входных параметров компонента, значения которых должны быть заданы на основе данных, получаемых от родительской сущности.

Начнём работу с описания того, какие именно входные параметры ожидает получить компонент product. Для этого добавим в объект с опциями, используемый при его создании, соответствующее свойство (Листинг 9.10).

**Листинг 9.10** Добавление нового свойства в компонент

Vue.component('product', {

props: {

premium: {

type: *Boolean*,

required: true

}

},

// Тут будут описание данных, методов, вычисляемых свойств

})

Обратите внимание на то, что тут используются встроенные возможности Vue по проверке параметров, передаваемых компоненту. А именно, мы указываем то, что типом входного параметра premium является Boolean, и то, что этот параметр является обязательным, устанавливая required в true.

Далее, внесем в шаблон изменение, выводящее переданные объекту параметры. Выведя значение свойства premium на странице, мы убедимся в правильности работы исследуемого нами механизма (Листинг 9.11).

**Листинг 9.11** Вывод переданных компоненту данных

...

<p>User is premium: {{ premium }}</p>

...

Пока всё идёт нормально. Компонент product знает о том, что он будет получать необходимый для его работы параметр типа Boolean. Мы подготовили место для вывода соответствующих данных.

Но мы пока ещё не передали параметр premium компоненту. Сделать это можно с помощью пользовательского атрибута, который похож на «трубопровод», ведущий к компоненту, через который ему можно передавать входные параметры, и, в частности, premium.

Доработаем код в index.html (Листинг 9.12).

**Листинг 9.12** Передача параметров компоненту

...

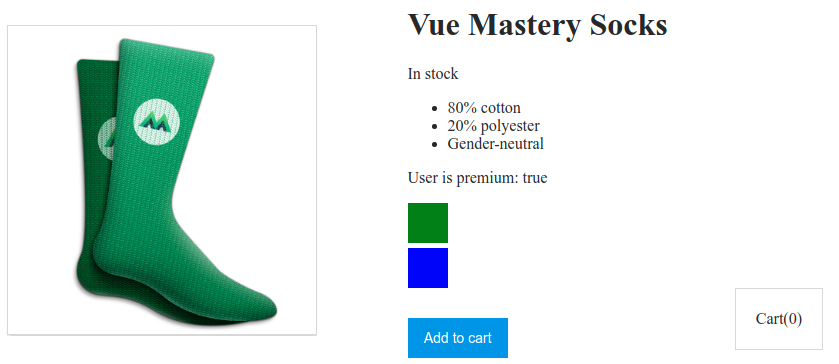
<div id**="app"**>

<product :premium**="premium"**></product>

</div>

...

Обновим страницу (Рисунок 9.2).



**Рисунок. 9.2.** Вывод данных, переданных компоненту

Теперь входные параметры передаются компоненту. Поговорим о том, что именно мы только что сделали.

Мы передаём компоненту входной параметр, или «пользовательский атрибут», называемый premium. Мы привязываем этот пользовательский атрибут, используя конструкцию, представленную двоеточием, к свойству premium, которое хранится в данных нашего экземпляра Vue.

Теперь корневой экземпляр Vue может передать premium дочернему компоненту product. Так как атрибут привязан к свойству premium из данных экземпляра Vue, текущее значение premium будет всегда передаваться компоненту product.

Вышеприведенный рисунок, а именно, надпись User is premium: true, доказывает то, что всё сделано правильно.

Сейчас, когда данные о том, обладает ли пользователь премиум-аккаунтом, попадают в компонент, давайте используем эти данные для того чтобы вывести на странице сведения о стоимости доставки. Не будем забывать о том, что если параметр premium установлен в значение true, то пользователю полагается бесплатная доставка. Создадим новое вычисляемое свойство shipping и воспользуемся в нём параметром premium (Листинг 9.13).

**Листинг 9.13** Вычисляемое свойство на основе входного параметра

...

shipping() {

if (this.premium) {

return "Free";

} else {

return 2.99

}

}

...

Если в параметре this.premium хранится true — вычисляемое свойство shipping вернёт Free. В противном случае оно вернёт 2.99.

Уберём из шаблона компонента код вывода значения параметра premium. Добавим элемент <p>Shipping: {{ shipping }}</p> для того, чтобы вывести стоимость доставки (Рисунок 9.3)



**Рисунок. 9.3.** Премиум-пользователь получает бесплатную доставку

Текст Shipping: Free появляется на странице из-за того, что компоненту передан входной параметр premium, установленный в значение true.

Замечательно! Теперь мы научились передавать данные от родительских сущностей дочерним и смогли воспользоваться этими данными в компоненте для управления стоимостью доставки товаров.

Кстати, стоит отметить, что в дочерних компонентах не следует изменять их входные параметры.

## Практическая работа №8

Создайте новый компонент product-details, который должен использовать входной параметр details и отвечать за визуализацию той части карточки товара, которая раньше формировалась с использованием следующего кода (Листинг 9.14).

**Листинг 9.14** Вывод детальной информации

...

<ul>

<li v-for="detail in details">{{ detail }}</li>

</ul>

...

## Итоги

* Компоненты — это блоки кода, представленные в виде пользовательских элементов.
* Компоненты упрощают управление приложением благодаря тому, что позволяют разделить его на части, подходящие для многократного использования. Они содержат в себе описания визуальной составляющей и функционала соответствующей части приложения.
* Данные компонента представлены методом data() объекта с опциями.
* Для передачи данных от родительских сущностей дочерним сущностям используются входные параметры (props).
* Мы можем описать требования к входным параметрам, которые принимает компонент.
* Входные параметры передаются компонентам через пользовательские атрибуты.
* Данные родительского компонента можно динамически привязать к пользовательским атрибутам.
* Инструменты разработчика Vue дают ценные сведения о компонентах.

# 10. Пользовательские события

## Цель урока

Нам надо, чтобы компонент product мог бы сообщать родительской сущности, корневому экземпляру Vue, о том, что произошло некое событие. При этом product должен отправлять, вместе с уведомлением о возникновении события, некие данные.

## Начальный вариант кода

В файле index.html учебного проекта сейчас находится такой код (Листинг 10.1).

**Листинг 10.1** Начальный вариант кода

...

<div id**="app"**>

<product :premium**="premium"**></product>

</div>

...

Вот — содержимое файла main.js (Листинг 10.2).

**Листинг 10.2** Начальный вариант кода js

Vue.component('product', {

props: {

premium: {

type: *Boolean*,

required: true

}

},

template: `

<div class="product">

<div class="product-image">

<img :src="image" :alt="altText"/>

</div>

<div class="product-info">

<h1>{{ title }}</h1>

<p v-if="inStock">In stock</p>

<p v-else>Out of Stock</p>

<ul>

<li v-for="detail in details">{{ detail }}</li>

</ul>

<p>Shipping: {{ shipping }}</p>

<div

class="color-box"

v-for="(variant, index) in variants"

:key="variant.variantId"

:style="{ backgroundColor:variant.variantColor }"

@mouseover="updateProduct(index)"

></div>

<div class="cart">

<p>Cart({{ cart }})</p>

</div>

<button

v-on:click="addToCart"

:disabled="!inStock"

:class="{ disabledButton: !inStock }"

>

Add to cart

</button>

</div>

</div>

`,

data() {

return {

product: "Socks",

brand: 'Vue Mastery',

selectedVariant: 0,

altText: "A pair of socks",

details: ['80% cotton', '20% polyester', 'Gender-neutral'],

variants: [

{

variantId: 2234,

variantColor: 'green',

variantImage: "./assets/vmSocks-green-onWhite.jpg",

variantQuantity: 10

},

{

variantId: 2235,

variantColor: 'blue',

variantImage: "./assets/vmSocks-blue-onWhite.jpg",

variantQuantity: 0

}

],

cart: 0

}

},

methods: {

addToCart() {

this.cart += 1

},

updateProduct(index) {

this.selectedVariant = index;

*console*.log(index);

}

},

computed: {

title() {

return this.brand + ' ' + this.product;

},

image() {

return this.variants[this.selectedVariant].variantImage;

},

inStock() {

return this.variants[this.selectedVariant].variantQuantity

},

shipping() {

if (this.premium) {

return "Free";

} else {

return 2.99

}

}

}

})

let *app* = new Vue({

el: '#app',

data: {

premium: true

}

})

## Задача

Теперь, когда product представлен самостоятельным компонентом, то, что код, имеющий отношение к корзине, находится в product, смысла не имеет. Если у каждого товара будет своя корзина, за состоянием которой нам нужно наблюдать, в нашем приложении заведется большой беспорядок. Вместо этого мы хотели бы, чтобы корзина существовала бы на уровне корневого экземпляра Vue. Так же нам нужно, чтобы компонент product сообщал бы корневому экземпляру Vue о добавлении товаров в корзину, то есть — о нажатиях на кнопку Add to cart.

## Решение

Переместим данные, имеющие отношение к корзине, обратно в корневой экземпляр Vue (Листинг 10.3).

**Листинг 10.3** Перенос данных корзины обратно в корневой экземпляр

...

let *app* = new Vue({

el: '#app',

data: {

premium: true,

cart: 0

}

})

Далее — переносим шаблон корзины обратно в index.html, приведя его код к такому виду (Листинг 10.4).

**Листинг 10.4** Перенос отображения корзины из компонента

...

<div id**="app"**>

<div class**="cart"**>

<p>**Cart({{ cart }})**</p>

</div>

<product :premium**="premium"**></product>

</div>

...

Теперь, если открыть страницу приложения в браузере и нажать на кнопку Add to cart, ничего, как и ожидается, не произойдёт.

А что должно происходить по нажатию на эту кнопку? Нам нужно, чтобы при нажатии на неё корневой экземпляр Vue получал бы уведомление, которое вызывало бы метод, приводящий корзину в актуальное состояние, то есть, обновляющий значение, которое хранится в cart.

Для того чтобы этого добиться, давайте сначала перепишем код метода addToCart компонента product.

Сейчас он выглядит так (Листинг 10.5).

**Листинг 10.5** Первоначальный код метода addToCart

...

addToCart() {

this.cart += 1

},

...

Приведем его к такому виду (Листинг 10.6).

**Листинг 10.6** Преобразованный код метода addToCart

...

addToCart() {

this.$emit('add-to-cart');

},

...

Что всё это значит?

А значит это вот что. Когда вызывается метод addToCart, генерируется пользовательское событие с именем add-to-cart. Другими словами — когда нажимают на кнопку Add to cart, вызывается метод, генерирующий событие, сообщающее о том, что только что была нажата кнопка (то есть — что только что произошло событие, вызванное нажатием кнопки).

Но прямо сейчас ничто в приложении не ожидает появления этого события, не прослушивает его. Давайте добавим прослушиватель событий в index.html (Листинг 10.7).

**Листинг 10.7** Добавление прослушивателя на событие

...

<product :premium**="premium"** @add-to-cart**="updateCart"**></product>

...

Тут мы пользуемся конструкцией вида @add-to-card аналогично тому, как пользуемся конструкцией :premium. Но если :premium — это «трубопровод», по которому можно передавать данные дочернему компоненту от родительского, то @add-to-cart можно сравнить с «радиоприемником» родительского компонента, который принимает от дочернего компонента сведения о том, что была нажата кнопка Add to cart. Так как «радиоприёмник» находится в теге <product>, вложенном в <div id="app">, это значит, что при поступлении сведений о нажатии на Add to cart будет вызван метод updateCart, находящийся в корневом экземпляре Vue.

Код @add-to-cart=«updateCart» в переводе на обычный язык выглядит так: «Когда услышишь, что произошло событие add-to-cart, вызови метод updateCart».

Этот метод, который теперь будет объявлен в объекте с опциями, используемом при создании экземпляра Vue, вы, точно, уже где-то видели (Листинг 10.8).

**Листинг 10.8** Добавление обработчика на событие

...

methods: {

updateCart() {

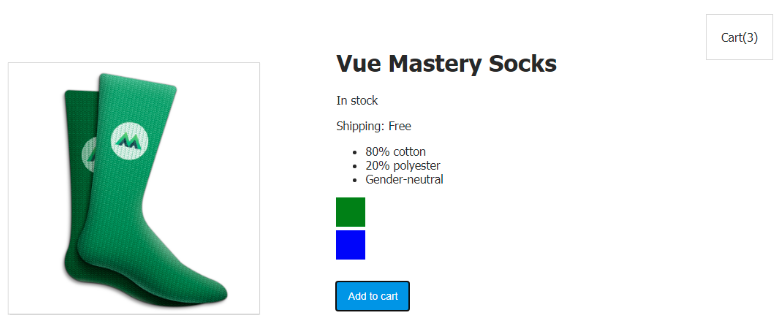
this.cart += 1;

}

}

...

Собственно говоря, это — точно такой же метод, который использовался раньше в product. Но теперь он находится в корневом экземпляре Vue и вызывается по нажатию на кнопку Add to cart (Рисунок 10.1).



**Рисунок. 10.1.** Кнопка снова работает

При нажатии на кнопку, находящуюся в компоненте product, вызывается метод addToCart, который генерирует событие. Корневой экземпляр Vue, «слушая радио», узнаёт о том, что данное событие произошло и вызывает метод updateCart, который увеличивает число, хранящееся в cart.

Мы добились своей цели, но в реальном приложении знание о том, что произошло некое событие, что некий товар добавлен в корзину, особой пользы не принесёт. В реальности нужно знать хотя бы о том, какой именно товар добавлен в корзину. А это значит, что в событии, которое генерируется в ответ на нажатие на кнопку, нужно передавать ещё и какие-то данные.

Данные, передаваемые в событии, можно описать в виде второго аргумента, передаваемого $emit в коде метода addToCart компонента product (Листинг 10.9).

**Листинг 10.9** Передача аргумента в событие

...

this.$emit('add-to-cart', this.variants[this.selectedVariant].variantId);

...

Теперь в событии передается идентификатор (variantId) товара, который пользователь хочет добавить в корзину. Это значит, что мы, вместо того чтобы просто увеличивать количество товара в корзине, можем пойти дальше и хранить в корзине более подробные сведения о добавленных в неё товарах. Для этого сначала преобразуем корзину в массив, записав пустой массив в cart (Листинг 10.10).

**Листинг 10.10** Преобразование корзины в массив

...

cart: []

...

Далее — перепишем метод updateCart. Во-первых — он теперь будет принимать id — тот самый идентификатор товара, который передается теперь в событии, во-вторых — он теперь будет помещать то, что получил, в массив (Листинг 10.11).

**Листинг 10.11** Измененный метод updateCart

...

methods: {

updateCart(id) {

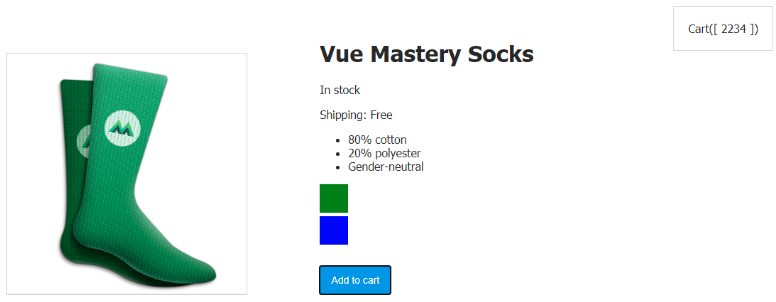
this.cart.push(id);

}

}

...

После однократного нажатия на кнопку в массив попадает идентификатор товара. Массив выводится на странице (Рисунок 10.2).

****

**Рисунок. 10.2.** Массив с идентификатором товара выводится на странице

Нам не нужно выводить на странице весь массив. Нас устроит вывод количества товаров, добавленных в корзину, то есть — в массив cart. Поэтому мы можем переписать код тега <p>, в котором выводятся сведения о количестве товаров, добавленных в корзину, так (Листинг 10.12).

**Листинг 10.12** Изменение отображение корзины

...

<p>**Cart({{ cart.length }})**</p>

...

Теперь мы просто выводим на странице длину массива, или, другими словами, количество товаров в корзине. Внешне корзина выглядит так же, как и прежде, но теперь мы, вместо простого увеличения значения числового свойства cart, сохраняем в массиве cart сведения о том, какой именно товар был добавлен в корзину.

## Практическая работа №9

Добавьте в проект кнопку, которая удаляет из массива cart товар, добавленный туда ранее. По нажатию на эту кнопку должно генерироваться событие, содержащее сведения об идентификаторе товара, который нужно убрать из корзины.

## Итоги

* Компонент может сообщать родительской сущности о том, что в нём что-то произошло, пользуясь конструкцией $emit.
* Родительский компонент может использовать обработчик события, заданный с использованием директивы v-on (или ее сокращенной версии @), для организации реакции на события, генерируемые дочерними компонентами. Если событие происходит — в родительском компоненте может быть вызван обработчик события.
* Родительский компонент может пользоваться данными, переданными в событии, сгенерированном дочерним компонентом.

# 11. Формы

## Цель урока

Мы собираемся создать форму, которая позволяет посетителям сайта отправлять отзывы на товары. При этом нужно, чтобы отзыв можно было бы отправить только в том случае, если заполнены все поля формы, которые обязательно должны быть заполнены.

## Начальный вариант кода

В файле index.html учебного проекта сейчас находится такой код (Листинг 11.1).

**Листинг 11.1** Начальный вариант кода

...

<div id**="app"**>

<div class**="cart"**>

<p>**Cart({{ cart.length }})**</p>

</div>

<product :premium**="premium"** @add-to-cart**="updateCart"**></product>

</div>

...

Вот — содержимое файла main.js (Листинг 11.2).

**Листинг 11.2** Начальный вариант кода js

Vue.component('product', {

props: {

premium: {

type: *Boolean*,

required: true

}

},

template: `

<div class="product">

<div class="product-image">

<img :src="image" :alt="altText"/>

</div>

<div class="product-info">

<h1>{{ title }}</h1>

<p v-if="inStock">In stock</p>

<p v-else>Out of Stock</p>

<ul>

<li v-for="detail in details">{{ detail }}</li>

</ul>

<p>Shipping: {{ shipping }}</p>

<div

class="color-box"

v-for="(variant, index) in variants"

:key="variant.variantId"

:style="{ backgroundColor:variant.variantColor }"

@mouseover="updateProduct(index)"

></div>

<button

v-on:click="addToCart"

:disabled="!inStock"

:class="{ disabledButton: !inStock }"

>

Add to cart

</button>

</div>

</div>

`,

data() {

return {

product: "Socks",

brand: 'Vue Mastery',

selectedVariant: 0,

altText: "A pair of socks",

details: ['80% cotton', '20% polyester', 'Gender-neutral'],

variants: [

{

variantId: 2234,

variantColor: 'green',

variantImage: "./assets/vmSocks-green-onWhite.jpg",

variantQuantity: 10

},

{

variantId: 2235,

variantColor: 'blue',

variantImage: "./assets/vmSocks-blue-onWhite.jpg",

variantQuantity: 0

}

],

}

},

methods: {

addToCart() {

this.$emit('add-to-cart', this.variants[this.selectedVariant].variantId);

},

updateProduct(index) {

this.selectedVariant = index;

*console*.log(index);

}

},

computed: {

title() {

return this.brand + ' ' + this.product;

},

image() {

return this.variants[this.selectedVariant].variantImage;

},

inStock() {

return this.variants[this.selectedVariant].variantQuantity

},

shipping() {

if (this.premium) {

return "Free";

} else {

return 2.99

}

}

}

})

let *app* = new Vue({

el: '#app',

data: {

premium: true,

cart: []

},

methods: {

updateCart(id) {

this.cart.push(id);

}

}

})

## Задача

Нам надо, чтобы посетители сайта могли бы оставлять отзывы на товары, но на нашем сайте пока нет средств, позволяющих получать данные от пользователей. Такими средствами являются формы.

## Решение задачи

Нам, для решения стоящей перед нами задачи, нужно создать форму. Начнём работу с создания нового компонента, предназначенного специально для работы с формой. Назовём этот компонент product-review. Такое имя выбрано из-за того, что компонент будет обеспечивать работу формы, направленной на сбор отзывов (review) о товарах. Компонент product-review будет вложен в компонент product.

Зарегистрируем новый компонент, приступим к формированию его шаблона и оснастим его некоторыми данными (Листинг 11.3).

**Листинг 11.3** Регистрация компонента product-review

...

Vue.component('product-review', {

template: `

<input>

`,

data() {

return {

name: null

}

}

})

...

Как видите, в шаблоне компонента есть элемент <input>, а в данных компонента есть свойство data, пока пустое.

Как привязать то, что пользователь вводит в поле, к свойству name?

На предыдущих занятиях мы говорили о привязке данных с использованием директивы v-bind, но тогда мы рассматривали лишь одностороннюю привязку. Поток данных шёл от свойства, хранящего данные, к элементу управления, визуализирующего их. А теперь нам надо, чтобы то, что пользователь вводит в поле, оказывалось бы в свойстве name, хранящемся в составе данных компонента. Другими словами, нам нужно, чтобы поток данных был бы направлен от поля ввода к свойству.

## Директива v-model

Директива v-model позволяет организовывать двустороннюю привязку данных. При такой схеме работы, если что-то новое оказывается в поле ввода, это приводит к изменению данных. И, соответственно, когда меняются данные, обновляется состояние элемента управления, эти данные использующего.

Добавим к полю ввода директиву v-model и привяжем это поле к свойству name из данных компонента (Листинг 11.4).

**Листинг 11.4** Использование директивы v-model

...

<input v-model="name">

...

Теперь добавим в шаблон компонента полный код формы (Листинг 11.5).

**Листинг 11.4** Итоговый шаблон формы

...

<form class="review-form" @submit.prevent="onSubmit">

<p>

<label for="name">Name:</label>

<input id="name" v-model="name" placeholder="name">

</p>

<p>

<label for="review">Review:</label>

<textarea id="review" v-model="review"></textarea>

</p>

<p>

<label for="rating">Rating:</label>

<select id="rating" v-model.number="rating">

<option>5</option>

<option>4</option>

<option>3</option>

<option>2</option>

<option>1</option>

</select>

</p>

<p>

<input type="submit" value="Submit">

</p>

</form>

...

Как видите, директивой v-model оснащены поля input, textarea и select. Обратите внимание на то, что при настройке поля select использован модификатор .number (подробнее о нём мы поговорим ниже). Это позволяет обеспечить преобразование соответствующих данных к типу Number, в то время как обычно они представлены в строковом виде.

Дополним набор данных компонента, добавив в него те данные, к которым привязаны вышеописанные элементы управления (Листинг 11.5)

**Листинг 11.5** Измененный набор данных компонента

...

data() {

return {

name: null,

review: null,

rating: null

}

}

...

В верхней части шаблона формы можно видеть, что при отправке формы вызывается метод onSubmit. Скоро мы создадим этот метод. Но сначала давайте поговорим о том, какую роль играет конструкция .prevent.

Это — модификатор события. Он предотвращает перезагрузку страницы после возникновения события submit. Есть и другие полезные модификаторы событий.

Теперь мы готовы к созданию метода onSubmit. Начнём с такого кода (Листинг 11.6).

**Листинг 11.6** Метод onSubmit

...

methods:{

onSubmit() {

let productReview = {

name: this.name,

review: this.review,

rating: this.rating

}

this.name = null

this.review = null

this.rating = null

}

}

...

Как видите, в этом методе, на основе данных, введенных пользователем, создается объект. Ссылка на него записывается в переменную productReview. Здесь же мы сбрасываем в null значения свойств name, review, rating. Но работа пока не окончена. Нам ещё нужно куда-то отправить productReview. Куда же отправлять этот объект?

Отзывы о товаре имеет смысл хранить там же, где хранятся данные компонента product. Учитывая то, что компонент product-review вложен в компонент product, мы можем сказать, что product-review — это дочерний компонент компонента product. Как мы уже выяснили на предыдущем занятии, для отправки данных от дочерних компонентов родительским можно использовать события, генерируемые с помощью $emit.

Доработаем метод onSubmit (Листинг 11.7).

**Листинг 11.7** Доработанный метод onSubmit

...

methods:{

onSubmit() {

let productReview = {

name: this.name,

review: this.review,

rating: this.rating

}

this.$emit('review-submitted', productReview)

this.name = null

this.review = null

this.rating = null

}

}

...

Теперь мы генерируем событие с именем review-submitted и передаём в нём только что созданный объект productReview.

Далее, нам надо организовать прослушивание этого события, разместив в шаблоне product следующее (Листинг 11.8).

**Листинг 11.8** Прослушивание события review-submitted

...

<product-review @review-submitted="addReview"></product-review>

...

Эта строка читается так: «Когда происходит событие review-submitted, нужно запустить метод addReview компонента product».

Вот код этого метода (Листинг 11.9).

**Листинг 11.9** Метод addReview

...

addReview(productReview) {

this.reviews.push(productReview)

}

...

Этот метод берет объект productReview, пришедший от метода onSubmit, а после этого помещает его в массив reviews, хранящийся в данных компонента product. Но такого массива в данных этого компонента пока нет. Поэтому добавим его туда (Листинг 11.10).

**Листинг 11.10** Массив отзывов

...

reviews: []

...

Замечательно! Теперь элементы формы привязаны к данным компонента product-review. Эти данные используются для создания объекта productReview. А этот объект передается, при отправке формы, компоненту product. В итоге объект productReview добавляется в массив reviews, который хранится в данных компонента product.

## Вывод отзывов о товаре

Теперь осталось лишь вывести на странице товара отзывы, оставленные посетителями сайта. Делать мы это будем в компоненте product, поместив соответствующий код выше кода, с помощью которого компонент product-review размещён в компоненте product (Листинг 11.11).

**Листинг 11.11** Вывод отзывов о товаре

...

<div>

<h2>Reviews</h2>

<p v-if="!reviews.length">There are no reviews yet.</p>

<ul>

<li v-for="review in reviews">

<p>{{ review.name }}</p>

<p>Rating: {{ review.rating }}</p>

<p>{{ review.review }}</p>

</li>

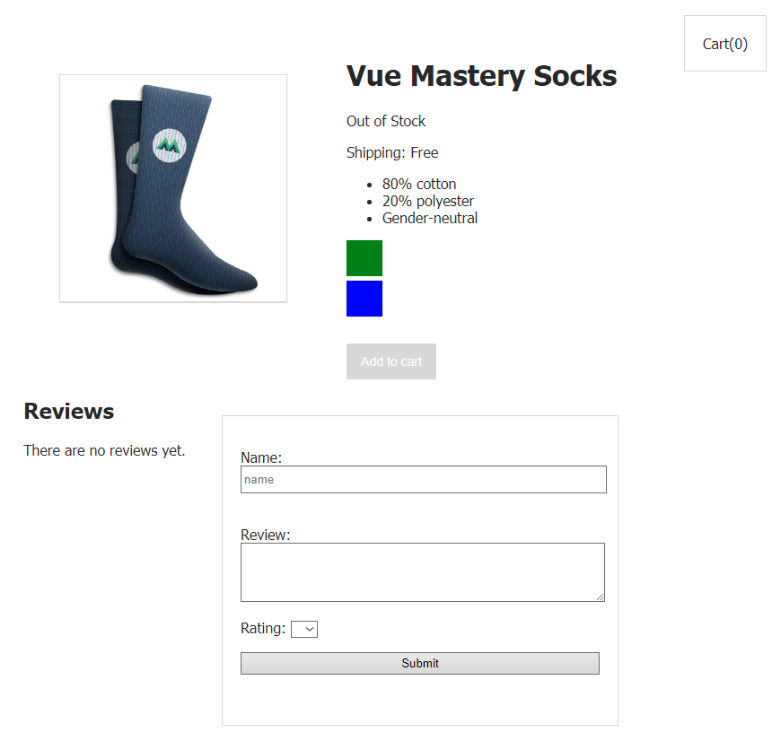
</ul>

</div>

...

Тут мы создаём список отзывов, пользуясь директивой v-for, и выводим данные, хранящиеся в объекте review, используя точечную нотацию.

В теге <p> мы проверяем, есть ли что-то в массиве reviews (проверяя его длину). Если в массиве ничего нет — мы выводим сообщение There are no reviews yet (Рисунок 11.1).



**Рисунок. 11.1.** Страница с формой для ввода отзывов

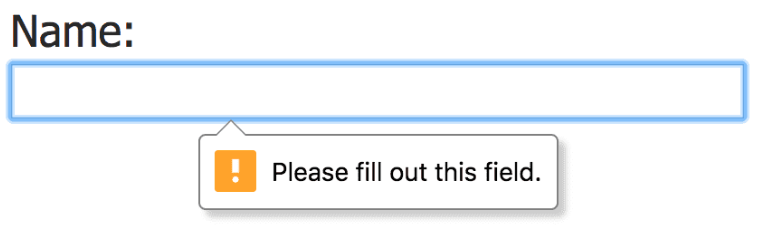
## Проверка форм

В формах часто бывают поля, которые, перед отправкой формы, должны быть обязательно заполнены. Например, нам не хотелось бы, чтобы пользователи отправляли бы отзывы, в которых поле текста отзыва было бы пустым.

К нашему счастью, HTML5 поддерживает атрибут required. Выглядит его использование так:

<input required >

Подобная конструкция приведёт к автоматическому выводу сообщения об ошибке в том случае, если пользователь попытается отправить форму, в которой required-поле окажется пустым (Рисунок 11.2).



**Рисунок. 11.2.** Сообщение об ошибке

Наличие в браузере стандартных механизмов проверки полей форм — это очень приятно, так как это может освободить нас от создания собственных механизмов проверки полей. Но то, как проводится стандартная проверка данных, может, в каком-то особом случае, нас не устроить. В такой ситуации вполне логично будет написать собственный код проверки форм.

## Пользовательская проверка форм

Поговорим о том, как создать собственную систему проверки форм.

Включим в состав данных компонента product-review массив для хранения сообщений об ошибках. Назовём его errors (Листинг 11.12).

**Листинг 11.12** Массив ошибок errors

...

data() {

return {

name: null,

review: null,

rating: null,

errors: []

}

},

...

Мы хотели бы добавлять в этот массив сведения об ошибках, возникающих в ситуациях, когда поля формы оказываются пустыми. Речь идёт о следующем коде (Листинг 11.13).

**Листинг 11.13** Валидация значений полей формы

if(!this.*name*) this.errors.push("Name required.")

if(!this.review) this.errors.push("Review required.")

if(!this.rating) this.errors.push("Rating required.")

Первая из этих строк сообщает системе о том, что если поле name не заполнено, нужно поместить в массив errors сообщение об ошибке Name required. Аналогично работают и другие строки, проверяющие поля review и rating. Если любое из них окажется пустым — в массив array попадёт сообщение об ошибке.

А куда поместить этот код?

Так как мы хотим, чтобы сообщения об ошибках записывались бы в массив только в тех случаях, когда при попытке отправки формы оказывается, что поля name, review или submit не заполнены, мы можем разместить этот код в методе onSubmit. Мы, кроме того, перепишем тот код, что в нём уже есть, добавив в него некоторые проверки (Листинг 11.14).

**Листинг 11.14** Валидация значений полей формы

...

onSubmit() {

if(this.name && this.review && this.rating) {

let productReview = {

name: this.name,

review: this.review,

rating: this.rating

}

this.$emit('review-submitted', productReview)

this.name = null

this.review = null

this.rating = null

} else {

if(!this.name) this.errors.push("Name required.")

if(!this.review) this.errors.push("Review required.")

if(!this.rating) this.errors.push("Rating required.")

}

}

...

Теперь мы проверяем поля name, review и rating. Если во всех этих полях имеются данные — мы создаем на их основе объект productReview и отправляем его родительскому компоненту. Затем значение соответствующих свойств сбрасывается.

Если же хотя бы одно из полей оказывается пустым, мы помещаем в массив errors сообщение об ошибке, зависящее от того, что именно не ввёл пользователь перед отправкой формы.

Осталось лишь вывести эти ошибки, что можно сделать с помощью следующего кода (Листинг 11.15).

**Листинг 11.15** Вывод ошибок валидации полей формы

...

<p v-if="errors.length">

<b>Please correct the following error(s):</b>

<ul>

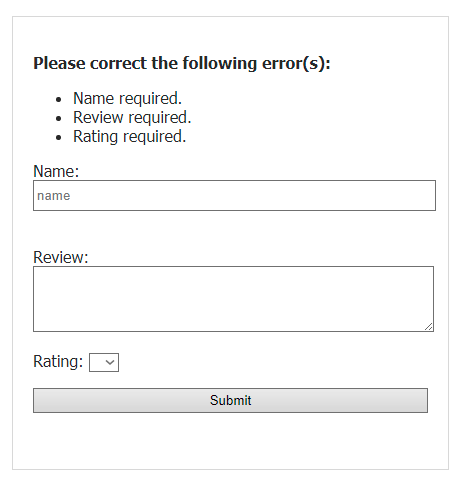
<li v-for="error in errors">{{ error }}</li>

</ul>

</p>

...

Тут используется директива v-if, с помощью которой мы проверяем массив errors на предмет наличия в нём сообщений об ошибках (фактически, анализируем длину массива). Если в массиве что-то есть, выводится элемент <p>, который, применяя v-for, выводит список ошибок из массива errors (Рисунок 11.3).



**Рисунок. 11.3.** Сообщение об ошибке

Теперь у нас имеется собственная система валидации формы.

## Использование модификатора .number

Модификатор .number, используемый в директиве v-model, способен принести немалую пользу. Но, применяя его, учитывайте то, что с ним связана одна проблема. Если соответствующее значение окажется пустым, оно будет представлено строкой, а не числом. В «Книге рецептов Vue» предлагается решение этой проблемы. Оно заключается в явном преобразовании соответствующего значения к числовому типу:

Number(this.myNumber)

## Практическая работа №10

Добавьте в форму следующий вопрос: «Would you recommend this product?». Сделайте так, чтобы пользователь мог бы ответить на него, пользуясь радиокнопками «yes» и «no». Проведите проверку ответа на этот вопрос и включите соответствующие данные в объект productReview.

## Итоги

* Для организации двусторонней привязки данных к элементам форм можно использовать директиву v-model.
* Модификатор .number сообщает Vue о том, что соответствующее значение нужно приводить к числовому типу. Но при работе с ним есть одна проблема, связанная с тем, что пустые значения остаются строками.
* Модификатор события .prevent позволяет предотвратить перезагрузку страницы после отправки формы.
* Средствами Vue можно реализовать достаточно простой механизм пользовательской валидации форм.

# 12. Вкладки, глобальная шина событий

## Цель урока

Мы хотим, чтобы на странице приложения присутствовали бы вкладки, одна из которых позволяет посетителям писать отзывы о товарах, а другая — просматривать существующие отзывы.

## Начальный вариант кода

Вот как на данном этапе работы выглядит содержимое файла index.html (Листинг 12.1).

**Листинг 12.1** Начальный вариант кода

...

<div id**="app"**>

<div class**="cart"**>

<p>**Cart({{ cart.length }})**</p>

</div>

<product :premium**="premium"** @add-to-cart**="updateCart"**></product>

</div>

...

В main.js имеется следующий код (Листинг 12.2).

**Листинг 12.2** Начальный вариант кода js

Vue.component('product-review', {

template: `

<form class="review-form" @submit.prevent="onSubmit">

<p v-if="errors.length">

<b>Please correct the following error(s):</b>

<ul>

<li v-for="error in errors">{{ error }}</li>

</ul>

</p>

<p>

<label for="name">Name:</label>

<input id="name" v-model="name" placeholder="name">

</p>

<p>

<label for="review">Review:</label>

<textarea id="review" v-model="review"></textarea>

</p>

<p>

<label for="rating">Rating:</label>

<select id="rating" v-model.number="rating">

<option>5</option>

<option>4</option>

<option>3</option>

<option>2</option>

<option>1</option>

</select>

</p>

<p>

<input type="submit" value="Submit">

</p>

</form>

`,

data() {

return {

name: null,

review: null,

rating: null,

errors: []

}

},

methods:{

onSubmit() {

if(this.name && this.review && this.rating) {

let productReview = {

name: this.name,

review: this.review,

rating: this.rating

}

this.$emit('review-submitted', productReview)

this.name = null

this.review = null

this.rating = null

} else {

if(!this.name) this.errors.push("Name required.")

if(!this.review) this.errors.push("Review required.")

if(!this.rating) this.errors.push("Rating required.")

}

}

}

})

Vue.component('product', {

props: {

premium: {

type: *Boolean*,

required: true

}

},

template: `

<div class="product">

<div class="product-image">

<img :src="image" :alt="altText"/>

</div>

<div class="product-info">

<h1>{{ title }}</h1>

<p v-if="inStock">In stock</p>

<p v-else>Out of Stock</p>

<ul>

<li v-for="detail in details">{{ detail }}</li>

</ul>

<p>Shipping: {{ shipping }}</p>

<div

class="color-box"

v-for="(variant, index) in variants"

:key="variant.variantId"

:style="{ backgroundColor:variant.variantColor }"

@mouseover="updateProduct(index)"

></div>

<button

v-on:click="addToCart"

:disabled="!inStock"

:class="{ disabledButton: !inStock }"

>

Add to cart

</button>

</div>

<div>

<h2>Reviews</h2>

<p v-if="!reviews.length">There are no reviews yet.</p>

<ul>

<li v-for="review in reviews">

<p>{{ review.name }}</p>

<p>Rating: {{ review.rating }}</p>

<p>{{ review.review }}</p>

</li>

</ul>

</div> <product-review @review-submitted="addReview"></product-review>

</div>

`,

data() {

return {

product: "Socks",

brand: 'Vue Mastery',

selectedVariant: 0,

altText: "A pair of socks",

details: ['80% cotton', '20% polyester', 'Gender-neutral'],

variants: [

{

variantId: 2234,

variantColor: 'green',

variantImage: "./assets/vmSocks-green-onWhite.jpg",

variantQuantity: 10

},

{

variantId: 2235,

variantColor: 'blue',

variantImage: "./assets/vmSocks-blue-onWhite.jpg",

variantQuantity: 0

}

],

reviews: []

}

},

methods: {

addToCart() {

this.$emit('add-to-cart', this.variants[this.selectedVariant].variantId);

},

updateProduct(index) {

this.selectedVariant = index;

*console*.log(index);

},

addReview(productReview) {

this.reviews.push(productReview)

}

},

computed: {

title() {

return this.brand + ' ' + this.product;

},

image() {

return this.variants[this.selectedVariant].variantImage;

},

inStock() {

return this.variants[this.selectedVariant].variantQuantity

},

shipping() {

if (this.premium) {

return "Free";

} else {

return 2.99

}

}

}

})

let *app* = new Vue({

el: '#app',

data: {

premium: true,

cart: []

},

methods: {

updateCart(id) {

this.cart.push(id);

}

}

})

## Задача

Сейчас отзывы и форма, которая используется для отправки отзывов, выводятся на странице рядом друг с другом. Это — вполне рабочая структура. Но ожидается, что со временем на странице будет появляться всё больше и больше отзывов. Это значит, что пользователям будет удобнее взаимодействовать со страницей, на которой, по их выбору, выводится либо форма, либо список отзывов.

## Решение задачи

Для того чтобы решить нашу задачу, мы можем добавить на страницу систему вкладок. Одна из них, с заголовком Reviews, будет выводить отзывы. Вторая, с заголовком Make a Review, будет выводить форму для отправки отзывов.

## Создание компонента, реализующего систему вкладок

Начнём работу с создания компонента product-tabs. Он будет выводиться в нижней части визуального представления компонента product. Со временем он заменит собой тот код, который сейчас используется для вывода на странице списка отзывов и формы (Листинг 12.3).

**Листинг 12.3** Регистрация нового компонента

Vue.component('product-tabs', {

template: `

<div>

<span class="tab" v-for="(tab, index) in tabs" :key="index">{{ tab }}</span>

</div>

`,

data() {

return {

tabs: ['Reviews', 'Make a Review']

}

}

})

...

Сейчас это — лишь заготовка компонента, которую мы скоро доработаем. Пока же в двух словах обсудим то, что представлено в этом коде.

В данных компоненте имеется массив tabs, содержащий строки, которые мы используем в качестве заголовков вкладок. В шаблоне компонента применяется конструкция v-for, с помощью которой для каждого элемента массива tabs создается элемент <span>, содержащий соответствующую строку. То, что формирует этот компонент на данном этапе работы над ним, будет выглядеть так, как показано ниже (Рисунок 12.1).



**Рисунок. 12.1.** Компонент product-tabs на начальном этапе работы над ним

Нам, для достижения наших целей, нужно знать о том, какая из вкладок является активной. Поэтому добавим в данные компонента свойство selectedTab. Будем динамически задавать значение этого свойства, пользуясь обработчиком событий, реагирующим на щелчки по заголовкам вкладок (Листинг 12.4).

**Листинг 12.4** Событие - щелчок по заголовку вкладки

...

@click="selectedTab = tab"

...

В свойство будут записываться строки, соответствующие заголовкам вкладок.

То есть, если пользователь щёлкнет по вкладке Reviews, то в selectedTab будет записана строка Reviews. Если будет сделан щелчок по вкладке Make a Review, то в selectedTab попадёт строка Make a Review.

Вот как теперь будет выглядеть полный код компонента (Листинг 12.5).

**Листинг 12.5** Измененный код компонента

Vue.component('product-tabs', {

template: `

<div>

<ul>

<span class="tab"

v-for="(tab, index) in tabs"

@click="selectedTab = tab"

>{{ tab }}</span>

</ul>

</div>

`,

data() {

return {

tabs: ['Reviews', 'Make a Review'],

selectedTab: 'Reviews' // устанавливается с помощью @click

}

}

})

...

## Привязка класса к активной вкладке

Пользователь, работая с интерфейсом, в котором используются вкладки, должен знать о том, какая вкладка является активной. Реализовать подобный механизм можно, воспользовавшись привязкой классов к элементам <span>, использующимся для вывода названий вкладок (Листинг 12.6).

**Листинг 12.6** Отображение активной вкладки

...

:class="{ activeTab: selectedTab === tab }"

...

Если объяснить вышеприведенную конструкцию простым языком, то оказывается, что к вкладке применяется стиль, заданный для класса activeTab, в том случае, когда selectedTab равняется tab. Так как в selectedTab записывается название вкладки, по которой только что щелкнул пользователь, стиль .activeTab будет применяться именно к активной вкладке.

Другими словами, когда пользователь щелкнет по первой вкладке, в tab будет находиться Reviews, то же самое будет записано и в selectedTab. В результате к первой вкладке будет применен стиль .activeTab.

Теперь заголовки вкладок на странице будут выглядеть так, как показано ниже (Рисунок 12.2).



**Рисунок. 12.2.** Выделенный заголовок активной вкладки

Судя по всему, на данном этапе всё работает так, как ожидается, поэтому мы можем идти дальше.

## Работа над шаблоном компонента

Теперь, когда мы можем сообщить пользователю о том, какая именно вкладка является активной, можно продолжить работу над компонентом. А именно, речь идет о доработке его шаблона, об описании того, что именно будет выводиться на странице при активации каждой из вкладок.

Подумаем о том, что надо показать пользователю в том случае, если он щелкнет по вкладке Reviews. Это, понятно, отзывы о товаре. Поэтому переместим код вывода отзывов из шаблона компонента product в шаблон компонента product-tabs, разместив этот код ниже конструкции, используемой для вывода заголовков вкладок. Вот как теперь будет выглядеть шаблон компонента product-tabs (Листинг 12.7)

**Листинг 12.7** Измененный шаблон компонента product-tabs

...

template: `

<div>

<ul>

<span class="tab"

:class="{ activeTab: selectedTab === tab }"

v-for="(tab, index) in tabs"

@click="selectedTab = tab"

>{{ tab }}</span>

</ul>

<div>

<p v-if="!reviews.length">There are no reviews yet.</p>

<ul>

<li v-for="review in reviews">

<p>{{ review.name }}</p>

<p>Rating: {{ review.rating }}</p>

<p>{{ review.review }}</p>

</li>

</ul>

</div>

</div>

`,

...

Обратите внимание на то, что мы избавились от тега <h2><font color="#3AC1EF">, так как нам больше не нужно выводить заголовок Reviews над списком отзывов. Вместо этого заголовка будет выводиться заголовок соответствующей вкладки.

Но одного только переноса кода шаблона недостаточно для того чтобы обеспечить вывод отзывов. Массив reviews, данные которого используются для вывода отзывов, хранится в составе данных компонента product. Нам нужно передать этот массив в компонент product-tabs, используя механизм входных параметров компонента. Добавим в объект с опциями, используемый при создании product-tabs, следующее (Листинг 12.8).

**Листинг 12.8** Входные параметры компонента product-tabs

...

props: {

reviews: {

type: *Array*,

required: false

}

},

...

Передадим массив reviews из компонента product в компонент product-tabs, воспользовавшись, в шаблоне product, следующей конструкцией (Листинг 12.9).

**Листинг 12.9** Передача входных параметров компоненту product-tabs

...

<product-tabs :reviews="reviews"></product-tabs>

...

А теперь поразмыслим о том, что нужно вывести на странице в том случае, если пользователь щёлкнет по заголовку вкладки Make a Review. Это, конечно, форма для отправки отзывов. Для того чтобы подготовить проект к дальнейшей работе над ним — перенесем код подключения компонента product-review из шаблона компонента product в шаблон product-tabs. Разместим следующий код ниже элемента <div>, используемого для вывода списка отзывов (Листинг 12.10).

**Листинг 12.10** Вывод формы для отправки отзыва

...

<div>

<product-review @review-submitted="addReview"></product-review>

</div>

...

Если взглянуть сейчас на страницу приложения, то окажется, что список отзывов и форма выводятся на ней ниже заголовков вкладок.

При этом щелчки по заголовкам, хотя и приводят к их выделению, никак не влияют на другие элементы страницы. Далее, если попытаться воспользоваться формой, окажется, что она перестала нормально работать. Всё это — вполне ожидаемые следствия изменений, внесенных нами в приложение. Продолжим работу и приведем наш проект в работоспособное состояние.

## Вывод элементов страницы по условию

Теперь, когда мы подготовили основные элементы шаблона компонента product-tabs, пришло время создать систему, которая позволит выводить разные элементы страницы основываясь на том, по заголовку какой именно вкладки щелкнул пользователь.

В данных компонента уже есть свойство selectedTab. Мы можем воспользоваться им в директиве v-show для организации условного рендеринга того, что относится к каждой из вкладок.

Так, к тегу <div>, содержащему код формирования списка отзывов, мы можем добавить такую конструкцию (Листинг 12.11)

**Листинг 12.11** Отображение блока в зависимости от условия

...

v-show="selectedTab === 'Make a Review'"

...

Это приведёт к тому, что форма будет выводиться только тогда, когда активна вкладка Make a Review.

Вот как теперь будет выглядеть шаблон компонента product-tabs (Листинг 12.12).

**Листинг 12.12** Шаблон компонента product-tabs

template: `

<div>

<ul>

<span class="tab"

:class="{ activeTab: selectedTab === tab }"

v-for="(tab, index) in tabs"

@click="selectedTab = tab"

>{{ tab }}</span>

</ul>

<div v-show="selectedTab === 'Reviews'">

<p v-if="!reviews.length">There are no reviews yet.</p>

<ul>

<li v-for="review in reviews">

<p>{{ review.name }}</p>

<p>Rating: {{ review.rating }}</p>

<p>{{ review.review }}</p>

</li>

</ul>

</div>

<div v-show="selectedTab === 'Make a Review'">

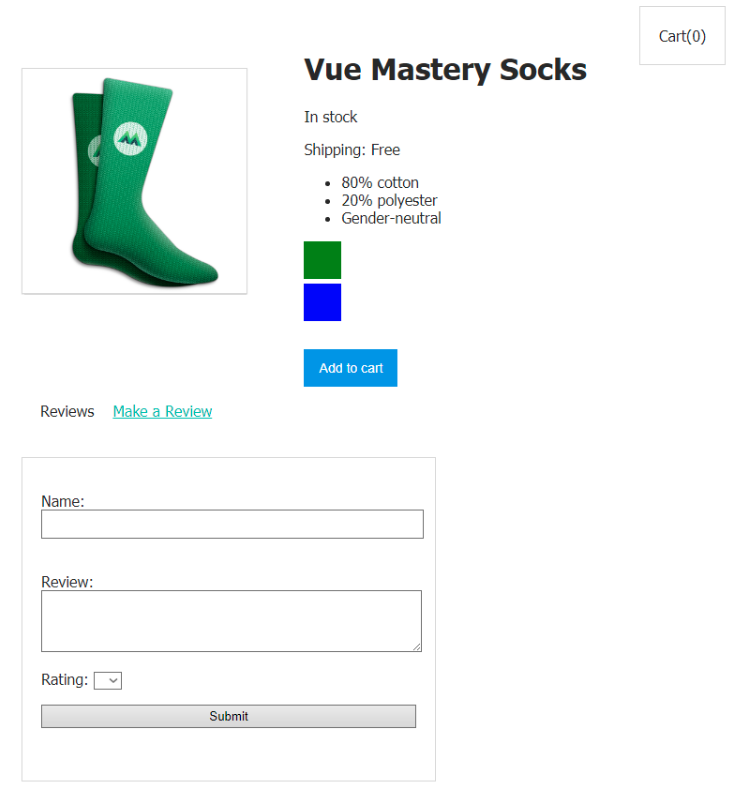
<product-review @review-submitted="addReview"></product-review>

</div>

</div>

`,

Если взглянуть на страницу и пощелкать по вкладкам — можно убедиться в том, что созданный нами механизм работает правильно (Рисунок 12.3).



**Рисунок. 12.3.** Щелчки по вкладкам приводят к скрытию одних элементов и к отображению других

Но отправка отзывов с помощью формы всё так же не работает. Исследуем проблему и исправим её.

## Решение проблемы с отправкой отзывов

Если сейчас взглянуть в консоль инструментов разработчика браузера, там можно обнаружить предупреждение.

Очевидно, система не может обнаружить метод addReview. Что с ним случилось?

Для ответа на этот вопрос вспомним о том, что addReview — это метод, который объявлен в компоненте product. Он должен вызываться в том случае, если компонент product-review (а это — дочерний компонент компонента product) генерирует событие review-submitted (Листинг 12.13).

**Листинг 12.13** Генерация события review-submitted

...

<product-review @review-submitted="addReview"></product-review>

...

Именно так всё и работало до переноса вышеприведенного фрагмента кода в компонент product-tabs. А теперь дочерним компонентом product является компонент product-tabs, а product-review — это теперь не «ребёнок», компонента product, а его «внук».

Наш код сейчас рассчитан на взаимодействие компонента product-review с родительским компонентом. Но теперь это — уже не компонент product. В результате оказывается, что нам, чтобы форма заработала бы правильно, нужно подвергнуть код проекта рефакторингу.

## Рефакторинг кода проекта

Для того чтобы обеспечить связь внучатых компонентов с их «бабушками» и «дедушками», или для того, чтобы наладить связь между компонентами одного уровня, нередко используют механизм, называемый глобальной шиной событий (global event bus).

Глобальная шина событий — это канал связи, который можно использовать для передачи информации между компонентами. И это, на самом деле, просто экземпляр Vue, который создают, не передавая ему объект с опциями. Создадим шину событий (Листинг 12.14).

**Листинг 12.14** Создание шины событий

let *eventBus* = new Vue()

Этот код попадет на верхний уровень файла main.js.

Возможно, вам будет легче освоить эту концепцию, если вы представите шину событий в виде автобуса. Его пассажирами являются данные, которые одни компоненты отправляют другим. В нашем случае речь идет о передаче одним компонентам сведений о событиях, сгенерированных другими компонентами. То есть, наш «автобус» будет ездить от компонента product-review к компоненту product, перевозя сведения о том, что форма был отправлена, и доставляя данные формы из product-review в product.

Сейчас в компоненте product-review, в методе onSubmit, есть такая строчка (Листинг 12.15).

**Листинг 12.15** Вызов события в компоненте product-review

...

this.$emit('review-submitted', productReview)

...

Заменим её на следующую, воспользовавшись eventBus вместо this (Листинг 12.16).

**Листинг 12.16** Измененный вызов события в компоненте product-review

*...*

*eventBus*.$emit('review-submitted', productReview)

...

После этого больше не нужно прослушивать событие review-submitted компонента product-review. Поэтому изменим код этого компонента в шаблоне компонента product-tabs на такой (Листинг 12.17).

**Листинг 12.17** Измененный вызов в шаблоне компонента product-tabs

...

<product-review></product-review>

...

Из компонента product теперь можно удалить метод addReview. Вместо него мы воспользуемся следующей конструкцией (Листинг 12.18).

**Листинг 12.18** Использование глобальной шины событий

eventBus.$on('review-submitted', productReview => {

this.reviews.push(productReview)

})

О том, как именно применить её в компоненте, мы поговорим ниже, а пока в двух словах опишем то, что в ней происходит. Эта конструкция указывает на то, что когда eventBus генерирует событие review-submitted, нужно взять данные, передаваемые в этом событии (то есть — productReview) и поместить их в массив reviews компонента product. Собственно говоря, это очень похоже на то, что до сих пор делалось в методе addReview, который нам больше не нужен. Обратите внимание на то, что в вышеприведенном фрагменте кода используется стрелочная функция. Этот момент достоин более подробного освещения.

## Причины использования стрелочной функции

Здесь мы используем синтаксис стрелочных функций, который появился в ES6. Дело в том, что контекст стрелочной функции привязан к родительскому контексту. То есть — когда мы, внутри этой функции, пользуемся ключевым словом this, оно равнозначно тому ключевому слову this, которое соответствует сущности, содержащей стрелочную функцию.

Этот код можно переписать и без использования стрелочных функций, но тогда понадобится организовать привязку this (Листинг 12.19).

**Листинг 12.19** Использование глобальной шины событий без стрелочных функций

*eventBus*.$on('review-submitted', function (productReview) {

this.reviews.push(productReview)

}.bind(this))

## Завершение работы над проектом

Мы почти достигли цели. Всё, что осталось сделать — найти место для фрагмента кода, обеспечивающего реакцию на событие review-submitted. Таким местом в компоненте product может стать функция mounted (Листинг 12.20).

**Листинг 12.20** Использование глобальной шины событий в функции mount

...

mounted() {

*eventBus*.$on('review-submitted', productReview => {

this.reviews.push(productReview)

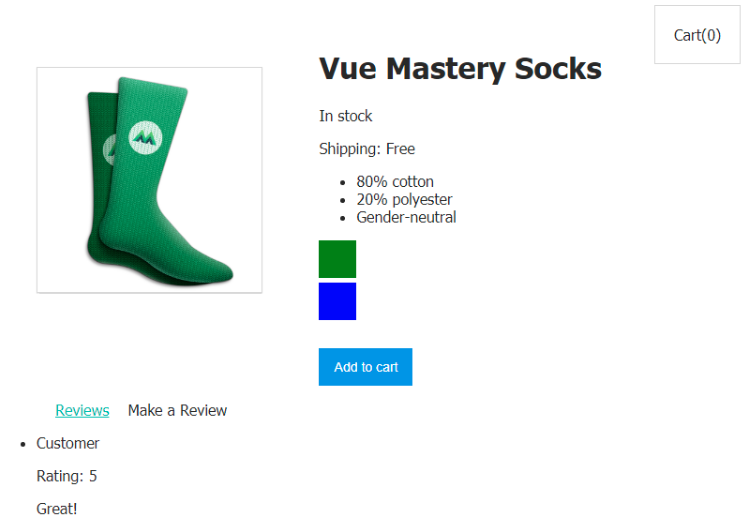
})

}

...

Что это за функция? Это — хук жизненного цикла, который вызывается один раз после того, как компонент будет смонтирован в DOM. Теперь, после того, как компонент product будет смонтирован, он будет ожидать появления событий review-submitted. После того, как такое событие будет сгенерировано, в данные компонента будет добавлено то, что передано в этом событии, то есть — productReview.

Если теперь попробовать оставить отзыв о товаре, воспользовавшись формой, окажется, что этот отзыв выводится там, где он должен быть (Рисунок 12.4).



**Рисунок. 12.4.** Форма работает так, как нужно

## Шина событий — это не лучшее решение для обеспечения связи компонентов

Хотя шина событий используется часто, и хотя вы можете встретить её в различных проектах, учитывайте то, что это — далеко не самое лучшее решение задачи обеспечения связи компонентов приложений.

По мере того, как приложение растёт, в нём может очень пригодиться система управления состоянием, основанная на Vuex. Это — паттерн и библиотека управления состоянием приложений.

## Практическая работа №11

Добавьте в проект вкладки Shipping и Details, на которых, соответственно, выводится стоимость доставки покупок и сведения о товарах.

## Итоги

* Для организации механизма вкладок можно воспользоваться средствами условного рендеринга.
* Для передачи данных между компонентами приложений часто пользуются глобальной шиной событий, представленной отдельным экземпляром Vue, создаваемым без использования объекта с опциями.
* Шина событий — это не самое лучшее решение задачи организации обмена данными между компонентами. Для решения подобной задачи лучше всего воспользоваться специализированной библиотекой. Например — Vuex.