**4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

Листинг программы представлен в Приложении А.

**4.1 Алгоритм выбора бренда**

Данный алгоритм служит для создания и заполнения селектора выбора бренда товара. В этом алгоритме особого внимания заслуживает технология AJAX, которая является способом организации клиент-серверного взаимодействия. При помощи этой технологии происходит асинхронное взаимодействие сервера с клиентом. Выбор бренда доступен на страницах создания и редактирования товара. Необходимо выбрать категорию товара. Это действие сгенерирует запрос на AJAX-запрос на клиенте и отправит его на сервер.

*document*.addEventListener("turbolinks:load", function() {  
 jQuery(function() {  
 return $('#subcategory\_select').change(function() {  
 const brandSelect = $('#brand\_select')[0];  
 const categoryId = $('#product\_category\_id')[0].value;  
 $.ajax({  
 type:"GET",  
 url:"/categories/" + categoryId + "/brands",  
 dataType:"json",  
 success: function(response) {  
 const ajaxVariable = response.brands;  
 const oldSelector = $('#category\_select')[0];  
 if (oldSelector) {  
 oldSelector.remove();  
 }  
 const selectList = ***document***.createElement("select");  
 selectList.setAttribute("id", "category\_select");  
 selectList.setAttribute("name", "product[brand\_id]");  
 brandSelect.appendChild(selectList);  
 $.each(ajaxVariable, function(index, value) {  
 $('#category\_select')  
 .append($("<option></option>")  
 .attr("value", value[0])  
 .text(value[1]));  
 });  
 }  
 });  
 });  
 });  
});

Данный запрос будет направлен по адресу /categories/:category\_id/brands и соответствовать методу index контроллера BrandsController.

def index  
 @brands = Brand.joins(brand\_categories: :category).where(categories: { id: params[:category\_id] }).pluck(:id, :brand\_name)  
 render json: { brands: @brands }  
end

Разберем данный метод подробнее. Первой строкой в переменную экземпляра @brands записываем c помощью запроса в базу данных массив, состоящий из массивов идентификаторов и названий брендов, которые содержит данная категория товара. Данный массив получается с помощью переданного через json-запрос идентификатора категории товаров id: params[:category\_id]. Если категория с данным идентификатором не содержит никаких товаров, то будет передан пустой массив, а селектор не будет содержать в себе ничего. Далее отправляем на клиент полученный массив в формате json. После получения ответа клиент отобразит в селекторе Brands полученные из данной категории бренды. В случае отсутствия в данной категории брендов, селектор Brands будет пустым.

**4.2 Алгоритм оформления заказа**

После заполнения пользователем корзины и нажатием кнопки«CHECKOUT» происходит формирование корзины. В первую очередь вызывается контроллер CheckoutController, который начинает формирование заказа.

class *CheckoutController* < ApplicationController  
  
 def new  
 @products = @basket.all  
 @order = Order.new  
 @user = current\_user || User.new  
 end  
  
 def create  
 @order = ::CheckoutsService.new(session, order\_params).make\_order  
 @basket.clear  
 redirect\_to '/'  
 end  
  
 private  
  
 def order\_params  
 params.require(:order).permit(:user\_id, :order\_price, :dest\_address, :name, :phone\_number)  
 end  
end

Рассмотрим данный класс подробнее. В первую очередь работает метод new

def new  
 @products = @basket.all  
 @order = Order.new  
 @user = current\_user || User.new  
end

Данный метод создает и инициализирует переменные экземпляра для дальнейшего их использования. Переменная @products заполняется товарами из корзины, переменная @order инициализируется как новая переменная класса Order, в переменную User записывается информация о текущем пользователе, а при его отсутствии – происходит создание нового пользователя.

Следующим отрабатывает метод create.

def create  
 @order = ::CheckoutsService.new(session, order\_params).make\_order  
 @basket.clear  
 redirect\_to '/'  
end

Данный метод заполняет переменную экземпляра @order, в которой вызывается сервис создания заказа CheckoutsService, который инициализируется с помощью данных о текущей сессии и параметрами заказа, которые представлены в приватном методе экземпляра класса order\_params.

private  
  
def order\_params  
 params.require(:order).permit(:user\_id, :order\_price, :dest\_address, :name, :phone\_number)  
end

Далее переходим к обзору сервиса CheckoutsService. Первым вызывается метод initialize при вызове сервиса CheckoutsService в контроллере CheckoutController.

def initialize(*session*, *order\_params*)  
 @session = *session* @order\_params = *order\_params* @key = if *session*['warden.user.user.key']  
 *session*['warden.user.user.key'][0][0]  
 else  
 *session*[:token]  
 end  
end

Данный метод инициализирует переменные, переданные при вызове сервиса при вызове сервиса CheckoutsService в контроллере CheckoutController. В переменную @session записываются данные о текущей сессии, переменная @order\_params получает параметры о заказе, в переменную @key записывается значение пользовательского ключа или сессионный токен.

Следующим выполняется метод make\_order.

def make\_order  
 @basket = ::BasketsService.new(@session).all  
  
 build\_form\_params  
 *order* = Order.new(@order\_params)  
 return *order*.errors unless *order*.valid?  
  
 *order*.save  
 make\_order\_product  
 ::EmailSenderWorker.perform\_async(@session['warden.user.user.key'][0][0], @order.id) if @session['warden.user.user.key'][0][0]  
end

Данный метод является основным в сервисе CheckoutService. В переменную @basket записываются данные о существующей в данной сессии корзине. Далее с помощью приватного метода экземпляра класса build\_form\_params устанавливаются параметры заказа, а именно: с помощью сервиса BasketsService полученной стоимости товаров из корзины и умножением на 1,2 с учётом НДС, а также устанавливается параметр идентификатора пользователя user\_id с помощью значения ключа сессии @key.

private  
  
def build\_form\_params  
 @order\_params[:order\_price] = ::BasketsService.new(@session).sum \* 1.2  
 @order\_params[:user\_id] = @key  
end

Далее в локальную переменную order заносятся данные созданного заказа, а затем заказ сохраняется при условии, что заказ валиден. В ином случае выводятся ошибки о создании заказа. Далее при помощи приватного метода экземпляра класса make\_order\_product составляется заказ из продуктов, содержащихся в корзине.

private  
  
def make\_order\_product(*order*)  
 @basket.map { |*product*| OrderProduct.create!(product\_id: *product*['id'], order\_id: *order*.id) }  
end

После составления заказа выполняется воркер EmailSenderWorker, который помещает задачу по отправке электронного письма в асинхронную очередь, и с чьей помощью происходит отправка письма о том, что заказ был принят.

class *EmailSenderWorker* include Sidekiq::Worker  
  
 def perform(*user\_id*, *order\_id*)  
 OrderReportMailer.with(user: User.find(*user\_id*), order: Order.find(*order\_id*)).report.deliver\_now  
 end  
end

Воркер EmailSenderWorker выполняет функцию perform, которая, передавая параметры user\_id и order\_id в OrderReportMailer, выполняет отправку электронного письма пользователю, который совершил данный заказ.

class *OrderReportMailer* < ApplicationMailer  
  
 def report  
 @user = params.fetch(:user)  
 @order = params[:order]  
  
 mail(to: @user.email, subject: 'Your order is still accepted!')  
 end  
  
end

Класс OrderReportMailer выполняет непосредственно отправку электронного письма адресату с помощью метода report. Параметры, полученные из метода perform воркера EmailSenderWorker, присваиваются непосредственно переменным экземпляра @user и @order. Далее с помощью функции mail, которую класс получил с помощью наследования от встроенного в Rails ApplicationMailer, совершает отправку электронного письма по электронному адресу пользователя, совершившего данный заказ. Результат получения письма об оформлении заказа представлен на рисунке 4.1.

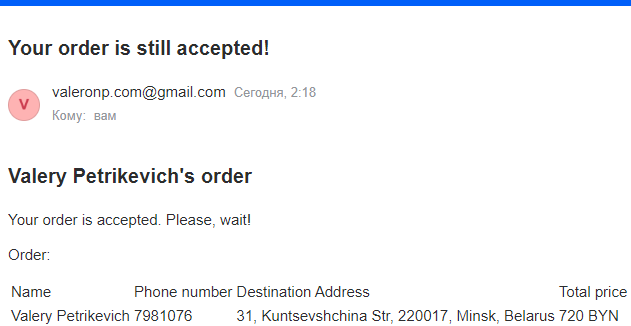


Рисунок 4.1 – Результат получения письма об оформлении заказа