**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Расчетно-графическое задание  
дисциплина: Технологии Web-программирования  
тема: «Сайт интернет-магазина кондитерских изделий»

Выполнила: ст. группы ВТ-41  
Горохов В.О.  
Проверил: Картамышев С.В.

Белгород 2020

**Оглавление**

[Цель работы 3](#_Toc59396838)

[HTML. Разработка макетов и верстка шаблонов web-приложения с помощью языков HTML и CSS 3](#_Toc59396839)

[Клиентское программирование. Vue js 5](#_Toc59396840)

[Серверное программирование. Yii2 7](#_Toc59396841)

[Разработка и проектирование базы данных 11](#_Toc59396842)

[REST API 15](#_Toc59396843)

[Работа с HTTP запросами 19](#_Toc59396844)

[Вывод 22](#_Toc59396845)

**Цель работы:** разработать проект по выбранной теме (интернет-магазин дверей и окон), разработать макеты и сверстать страницы для web-приложения, перенести вёрстку в отдельные компоненты фреймворка vuejs, настроить бекенд с помощью docker и yii2, спроектировать, создать и заполнить базу данных, реализовать REST API, связать фронтенд с бэкендом и базой данных.

**HTML. Разработка макетов и верстка шаблонов web-приложения с помощью языков HTML и CSS**

Приложение должно содержать следующие страницы:

1) Главная страница

2) Страница продукта

3) Авторизация пользователя

4) Регистрация пользователя

Каждая страница должна содержать header («шапка») и footer («нижняя часть»).

Header содержит ссылки на главную страницу (по нажатию на название и логотип сайта), страницу каталога, контактов, корзину и страницу профиля пользователя (на которой возможно авторизоваться или зарегистрироваться на сайте).

Макеты страниц создаем при помощи **HTML**-разметки. Основные HTML-теги:

<html></html> - корневой тег, который сообщает браузеру, что это HTML-документ. Все остальные элементы помещаются внутри него.

<head></head> - контейнер, в который помещаются метаданные документа, не видимые пользователям, но считываемые поисковыми роботами.

<body></body> - тег, обрамляющий видимую пользователям часть документа. Всё, что вы укажите внутри этого контейнера, отобразится на странице.

<h1></h1>......<h6></h6>. — теги заголовков, от самого большого к самому маленькому.

<b></b> — жирный текст без придания важности выделенному фрагменту.

<i></i> — выделение текста курсивом без придания важности.

<p></p> — контейнер для абзаца.

<img></img> — вставка изображения. Имеет атрибут src, который указывает на адрес нужного файла.

Настройка стилей HTML-документов осуществляется при помощи **CSS**. Подключение css в HTML-файл осуществляется в head при помощи следующей команды: <link rel="stylesheet" href="../css/homePage.css">.

Также при стилизации страниц была использована библиотека **Bootstrap**, которая предоставляет удобные компоненты с точки зрения графического содержания и автоматической оптимизации отображения под мобильные устройства с малым экраном.

Подключение библиотеки, а также используемых JS скриптов осуществляется аналогично CSS файлам – в head документа.

Основные использованные bootstrap компоненты navbar, сard, jumbotron, btn, carousel и другие. Для компонентов проводилась настройка css (например, используемых шрифтов, цветов и т.д.).

# 

# **Клиентское программирование. Vue js**

**Vue js** - веб-фреймворк, который может быть использован для создания одностраничных проектов в реактивном стиле на JavaScript. Включает в себя: vue-cli – инструментарий для разработки на vue js, vue-router – маршрутизатор для vue, а также vue-loader – загрузчик веб-пакетов. Vue-cli позволяет создать новый проект при помощи vue create newProject. Далее, выбираются необходимые параметры с точки зрения пользователя.

Одним из главных файлов проекта является index.js, который настраивает маршрутизацию проекта:

import Vue from 'vue'

import VueRouter from 'vue-router'

import Home from '../views/Home.vue'

import Enter from '../views/Enter.vue'

import Registration from '../views/Registration.vue'

import Product from '../views/Product.vue'

import Door from "@/views/Door";

// import Product from '../views/Product.vue'

Vue.use(VueRouter)

const routes = [

{

path: '/', // / - корневой каталог

name: 'Home',

component: Home

},

{

path: '/enter',

name: 'Enter',

component: Enter

},

{

path: '/registration',

name: 'Registration',

component: Registration

},

{

path: '/:url',

name: 'Product',

component: Product

},

{

path: '/:url',

name: 'Door',

component: Door

}

]

const router = new VueRouter({

mode: 'history',

base: process.env.BASE\_URL,

routes

})

export default router

Основой фреймворка являются компоненты, позволяющие использовать их повторно для различных страниц и различного контента.

Были добавлены следующие компоненты:

**Product.vue**

<template>

<div class="product mt-5">

<div class="form-group text-center">

<router-link v-bind:to="{name: 'Product', params: {url: productParam.url}}">

<img v-bind:src="productParam.itemsImage" height="350"> <br>

</router-link>

{{productParam.itemsName}} <br>

<b> {{ productParam.formattedPrice }} </b> <br>

<p> <button class="btn btn-secondary" v-on:click="addToCart">Купить</button> </p>

</div>

</div>

</template>

<script>

export default {

// import CartData from '@/components/cart/cart'

name: "Product",

props: ['productParam'], // props принемает на вх. данные из Home.vue items

methods: {

addToCart () {

alert ('Товар ' + this.productParam.itemsName + ' добавлен в корзину')

// CartData.add(this.product)

}

}

}

</script>

**ProductInfo.vue**

<template>

<div class="container divider mt-5 mb-5">

<div class="container\_local mr-5">

<img v-bind:src="productParam.itemsImage" height="350">

</div>

<div class="container\_local">

{{productParam.itemsName}} <br><br>

<b> Серия: Porta X </b><br><br>

Размеры:<br>

200\*60<br>

200\*70<br>

200\*80<br>

200\*90<br><br>

<b> {{ productParam.formattedPrice }} </b> <br><br>

<p> <button class="btn btn-secondary" v-on:click="addToCart">Купить</button> </p>

</div>

</div>

</template>

<script>

export default {

// import CartData from '@/components/cart/cart'

name: "Product",

props: ['productParam'], // props принемает на вх. данные из Home.vue items

methods: {

addToCart () {

alert ('Товар ' + this.productParam.itemsName + ' добавлен в корзину')

// CartData.add(this.product)

}

}

}

</script>

# **Серверное программирование. Yii2**

В данном разделе начали разработку бэкенда нашего проекта. Для этого был использован фреймворк yii2.

Развернули базовое приложение Yii2 App Basic используя команду composer create-project --prefer-dist yiisoft/yii2-app-basic basic.

Для настройки конфигурации и использования docker в проекте изменили файл docker-compose.yaml:

version: '3.1'

services:

php:

image: yiisoftware/yii2-php:7.3-apache

volumes:

- ~/.composer-docker/cache:/root/.composer/cache:delegated

- ./:/app:delegated

ports:

- '1199:80'

mysql:

image: mysql

command: --default-authentication-plugin=mysql\_native\_password --character-set-server=utf8 --collation-server=utf8\_general\_ci

environment:

- MYSQL\_ROOT\_PASSWORD=verysecret

- MYSQL\_DATABASE=bstu

- MYSQL\_USER=bstu

- MYSQL\_PASSWORD=bstu321!

Добавили следующий модуль и некоторые контроллеры для работы со статическими данными:

**ApiController.php**

<?php

namespace app\modules\v1\controllers;

use yii\filters\auth\CompositeAuth;

use yii\filters\ContentNegotiator;

use yii\filters\Cors;

use yii\filters\RateLimiter;

use yii\filters\VerbFilter;

use yii\rest\Controller;

use yii\web\Response;

class ApiController extends Controller {

public function behaviors () {

return [

'contentNegotiator' => [

'class' => ContentNegotiator::className(),

'formats' => [

'application/json' => Response::FORMAT\_JSON,

],

],

'verbFilter' => [

'class' => VerbFilter::className(),

'actions' => $this->verbs(),

],

'authenticator' => [

'class' => CompositeAuth::className(),

],

'rateLimiter' => [

'class' => RateLimiter::className(),

],

'corsFilter' => [

'class' => Cors::class

]

];

}

}

**CategoryController.php**

<?php

namespace app\modules\v1\controllers;

use app\modules\v1\models\Category;

use Yii;

class CategoryController extends ApiController {

public function actionAll() {

return Category::find()->all();

}

public function actionId($id) {

return Category::findOne(['id' => $id]);

}

public function actionCreate() {

$category = new Category();

$category->load(Yii::$app->request->getBodyParams(), '');

$category->save();

return $category;

}

}

**ProductController.php**

<?php

namespace app\modules\v1\controllers;

use app\modules\v1\models\Product;

use Yii;

class ProductController extends ApiController {

public function actionHome () {

$products = Product::find()->all(); // Вернет все записи

return $products;

}

public function actionCreate () {

$model = new Product();

$model->load(Yii::$app->request->getBodyParams(), '');

$model->save();

return $model;

}

public function actionInfo ($url) {

$product = Product::find()->where(['url' => $url])->all();

return $product;

}

public function actionIdcategory($category\_id) {

return Product::findAll(['idCategory' => $category\_id]);

}

public function actionId($id) {

return Product::findOne(['id' => $id]);

}

}

**Product.php**

<?php

namespace app\modules\v1\models;

use yii\behaviors\SluggableBehavior;

use Yii;

/\*\*

\* This is the model class for table "products".

\*

\* @property int $id

\* @property string $itemsName Название

\* @property string|null $itemsImage Изображение

\* @property float $itemsPrice Стоимость

\* @property string|null $itemsSeries Серия

\* @property string $createdAt Дата создания

\* @property string|null $updatedAt Дата изменения

\* @property string $url URL

\*/

class Product extends BaseModel {

public static function tableName () {

return 'products';

}

public function rules () {

return [

[['itemsName', 'itemsPrice'], 'required'],

[['itemsPrice'], 'number'],

[['createdAt', 'updatedAt'], 'safe'],

[['itemsName', 'itemsImage', 'itemsSeries', 'url'], 'string', 'max' => 128],

];

}

public function attributeLabels () {

return [

'id' => 'ID',

'itemsName' => 'Название',

'itemsImage' => 'Изображение',

'itemsPrice' => 'Стоимость',

'itemsSeries' => 'Серия',

'createdAt' => 'Дата создания',

'updatedAt' => 'Дата изменения',

'url' => 'URL'

];

}

public function toArray(array $fields = [], array $expand = [], $recursive = true) {

return [

'id' => $this->id,

'itemsName' => $this->itemsName,

'itemsImage' => $this->itemsImage,

'itemsPrice' => (double)$this->itemsPrice,

'formattedPrice' => number\_format($this->itemsPrice, 0, '.', ' ') . ' ₽',

'url' => $this->url,

'createdAt' => $this->createdAt,

// 'category' => $this->category,

// 'propertiesValues' => $this->propertiesValues

];

}

public function behaviors () {

$behaviors = parent::behaviors();

$behaviors[] = [

'class' => SluggableBehavior::class,

'attribute' => 'itemsName', // из itemsName берем и в url записываем

'slugAttribute' => 'url'

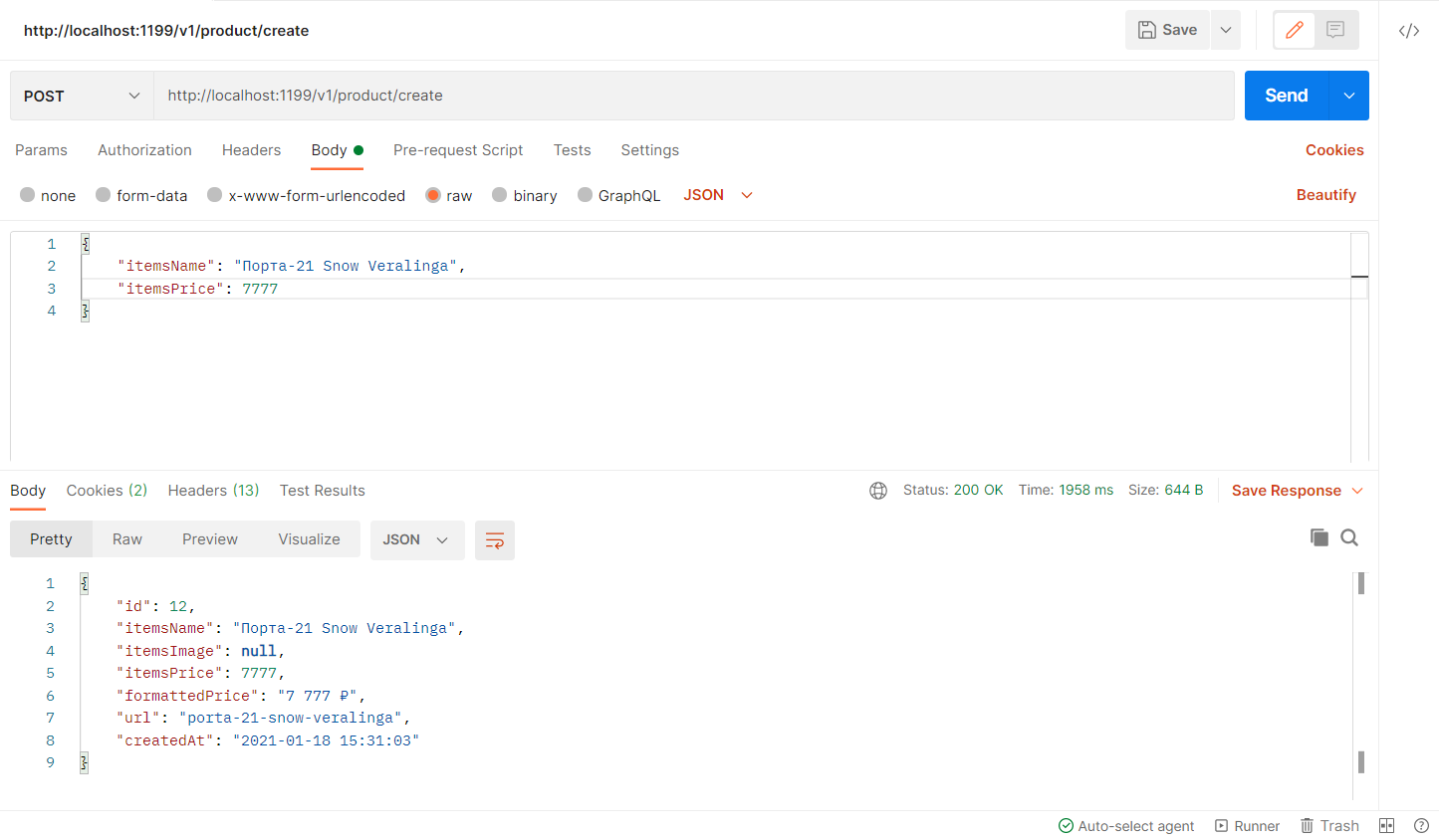
];

return $behaviors;

}

}

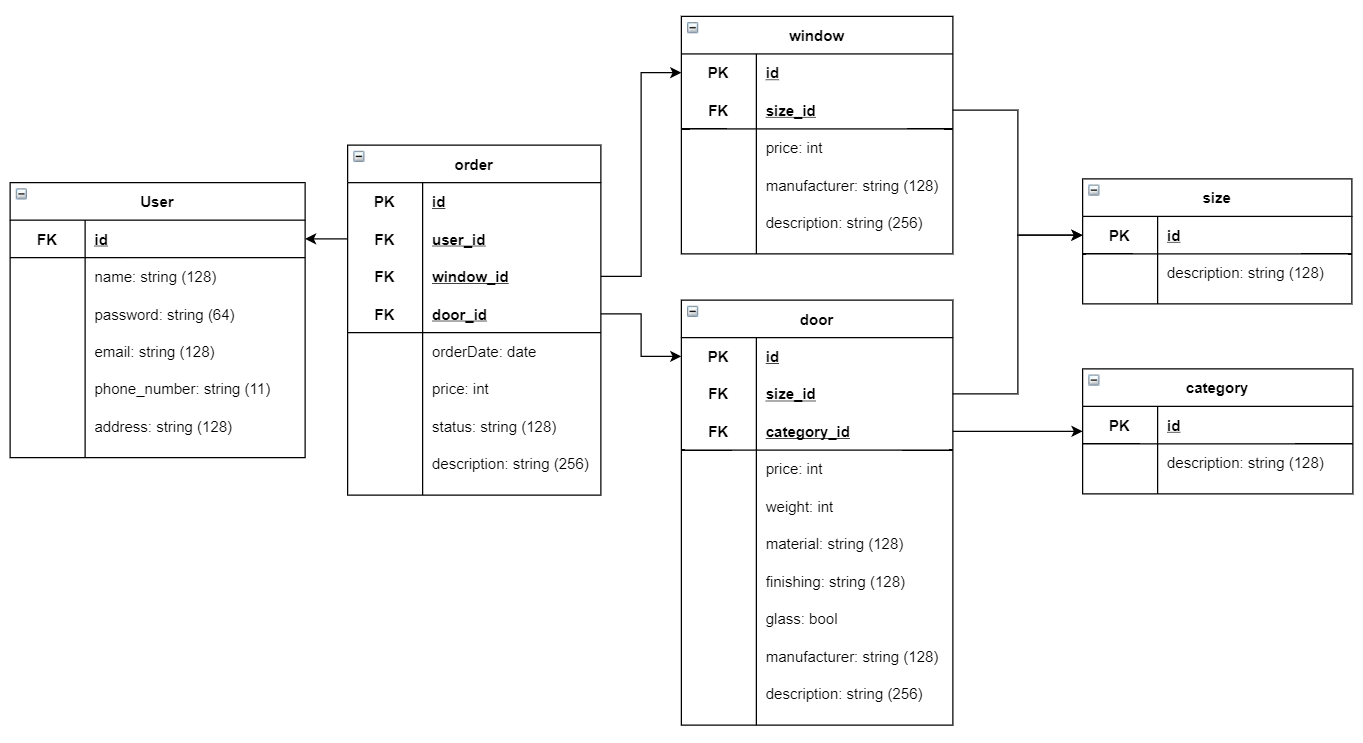
Работа запросов написанных контроллеров была проверена с помощью postman:



# **Разработка и проектирование базы данных**

В данном проекте была выбрана СУБД MySQL. Для удобного администрирования базы данных к проекту был подключен adminer.

Для сайта интернет-магазина дверей и окон была разработана следующая структура базы данных:



В файле конфигурации db.php и файле docker-compose были прописаны следующие настройки базы данных:

**db.php**

<?php

return [

'class' => 'yii\db\Connection',

'dsn' => 'mysql:host=mysql;dbname=bstu',

'username' => 'bstu',

'password' => 'bstu321!',

'charset' => 'utf8',

];

**docker-compose.yml**

mysql:

image: mysql

command: --default-authentication-plugin=mysql\_native\_password --character-set-server=utf8 --collation-server=utf8\_general\_ci

environment:

- MYSQL\_ROOT\_PASSWORD=verysecret

- MYSQL\_DATABASE=bstu

- MYSQL\_USER=bstu

- MYSQL\_PASSWORD=bstu321!

Для создания таблиц с помощью команды docker-compose run php yii migrate/create create\_product\_table были созданы миграции:

**m201215\_181557\_create\_door\_table.php**

<?php

use yii\db\Migration;

/\*\*

\* Handles the creation of table `{{%door}}`.

\*/

class m201215\_181557\_create\_door\_table extends Migration {

public function safeUp () {

$this->createTable('{{%door}}', [

'id' => $this->primaryKey(),

'size\_id' => $this->integer()->notNull()->comment('Размер'),

'category\_id' => $this->integer()->notNull()->comment('Категория'),

'price' => $this->integer()->comment('Цена'),

'weight' => $this->integer()->comment('Вес'),

'material' => $this->string(128)->comment('Материал'),

'finishing' => $this->string(128)->comment('Отделка'),

'glass' => $this->boolean()->comment('Стекло'),

'manufacturer' => $this->string(128)->comment('Производитель'),

'description' => $this->string(256)->comment('Описание'),

]);

$this->addForeignKey('fk-door-size\_id', '{{%door}}', 'size\_id', '{{%size}}', 'id', 'NO ACTION', 'NO ACTION');

$this->addForeignKey('fk-door-category\_id', '{{%door}}', 'category\_id', '{{%category}}', 'id', 'NO ACTION', 'NO ACTION');

}

public function safeDown () {

$this->dropForeignKey('fk-door-size\_id', '{{%door}}');

$this->dropForeignKey('fk-door-category\_id', '{{%door}}');

$this->dropTable('{{%door}}');

}

}

**m201215\_183800\_create\_order\_table.php**

<?php

use yii\db\Migration;

/\*\*

\* Handles the creation of table `{{%order}}`.

\*/

class m201215\_183800\_create\_order\_table extends Migration {

public function safeUp () {

$this->createTable('{{%order}}', [

'id' => $this->primaryKey(),

'door\_id' => $this->integer()->notNull()->comment('Дверь'),

'window\_id' => $this->integer()->notNull()->comment('Окно'),

'user\_id' => $this->integer()->notNull()->comment('Пользователь'),

'orderDate' => $this->dateTime()->comment('Дата заказа'),

'price' => $this->integer()->comment('Сумма заказа'),

'status' => $this->string(128)->comment('Статус'),

'description' => $this->string(256)->comment('Описание'),

]);

$this->addForeignKey('fk-order-door\_id', '{{%order}}', 'door\_id', '{{%door}}', 'id', 'NO ACTION', 'NO ACTION');

$this->addForeignKey('fk-order-window\_id', '{{%order}}', 'window\_id', '{{%window}}', 'id', 'NO ACTION', 'NO ACTION');

$this->addForeignKey('fk-order-user\_id', '{{%order}}', 'user\_id', '{{%user}}', 'id', 'NO ACTION', 'NO ACTION');

}

public function safeDown () {

$this->dropForeignKey('fk-order-door\_id', '{{%order}}');

$this->dropForeignKey('fk-order-window\_id', '{{%order}}');

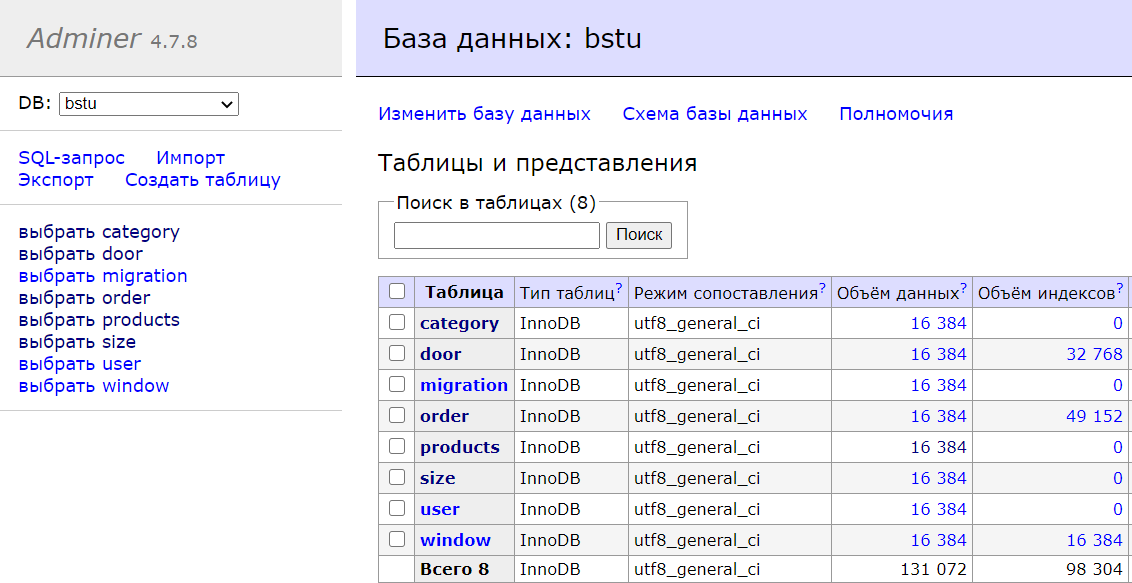
$this->dropForeignKey('fk-order-user\_id', '{{%order}}');

$this->dropTable('{{%order}}');

}

}

После применении команды docker-compose run php yii migrate/up проверили в adminer созданные таблицы:



Далее с помощью генератора моделей yii code generator были сгенерированы соответствующие модели. Для примера приведем код модели Door.php:

<?php

namespace app\modules\v1\models;

use app\models\Category;

use app\models\Size;

use app\models\Order;

use app\modules\v1\models\BaseModel;

/\*\*

\* This is the model class for table "door".

\*

\* @property int $id

\* @property int $size\_id Размер

\* @property int $category\_id Категория

\* @property int|null $price Цена

\* @property int|null $weight Вес

\* @property string|null $material Материал

\* @property string|null $finishing Отделка

\* @property int|null $glass Стекло

\* @property string|null $manufacturer Производитель

\* @property string|null $description Описание

\*

\* @property Category $category

\* @property Size $size

\* @property Order[] $orders

\*/

class Door extends BaseModel {

public static function tableName () {

return 'door';

}

public function rules () {

return [

[['size\_id', 'category\_id'], 'required'],

[['size\_id', 'category\_id', 'price', 'weight', 'glass'], 'integer'],

[['material', 'finishing', 'manufacturer'], 'string', 'max' => 128],

[['description'], 'string', 'max' => 256],

[['category\_id'], 'exist', 'skipOnError' => true, 'targetClass' => Category::className(), 'targetAttribute' => ['category\_id' => 'id']],

[['size\_id'], 'exist', 'skipOnError' => true, 'targetClass' => Size::className(), 'targetAttribute' => ['size\_id' => 'id']],

];

}

public function attributeLabels () {

return [

'id' => 'ID',

'size\_id' => 'Размер',

'category\_id' => 'Категория',

'price' => 'Цена',

'weight' => 'Вес',

'material' => 'Материал', // Картинка

'finishing' => 'Отделка', // Название

'glass' => 'Стекло',

'manufacturer' => 'Производитель',

'description' => 'Описание',

];

}

public function getCategory () {

return $this->hasOne(Category::className(), ['id' => 'category\_id']);

}

public function getSize () {

return $this->hasOne(Size::className(), ['id' => 'size\_id']);

}

public function getOrders () {

return $this->hasMany(Order::className(), ['door\_id' => 'id']);

}

}

# **REST API**

В данном пункте мы изучили основы разработки API для web-приложений и разработали REST API для своего проекта. Для взаимодействия с базой было разработано следующее REST API:

Базовый URL: localhost:1199/api/v1

**Товары**

1. Получение всех товаров:

URL: /product/all

Тип запроса: GET

Параметры: нет

1. Получение конкретного товара по его id:

URL: /product/id

Тип запроса: GET

Параметры: id – id товара, integer, обязательный

1. Получение товара по его url

URL: /product/info

Тип запроса: GET

Параметры: url – адрес товара, string, обязательный

1. Создание товара

URL: /product/create

Тип запроса: POST

Параметры: в теле запроса принимается объект товара с полями: name (обязательный, string, название товара), price (обязательный, int, цена товара)

**Двери**

1. Получение всех дверей:

URL: /door/all

Тип запроса: GET

Параметры: нет

1. Получение конкретной двери по ее id:

URL: /door/id

Тип запроса: GET

Параметры: id – id товара, integer, обязательный

1. Получение двери по ее url

URL: /door/info

Тип запроса: GET

Параметры: url – адрес двери, string, обязательный

4. Получение двери определенной категории:

URL: /door/Idcategory

Тип запроса: GET

Параметры: category\_id – id категории двери, integer, обязательный

5. Получение двери определенного размера:

URL: /door/IdSize

Тип запроса: GET

Параметры: size\_id – id категории двери, integer, обязательный

**Категории товаров**

1. Получение всех категорий:

URL: /category/all

Тип запроса: GET

Параметры: нет

1. Получение определенной категории:

URL: /category/id

Тип запроса: GET

Параметры: id – id категории, integer, обязательный

1. Создание категории:

URL: /category/create

Тип запроса: POST

Параметры: в теле запроса принимается объект создаваемой категории с полем description (обязательный, string, название категории)

**Окна**

1. Получение всех окон:

URL: /window/all

Тип запроса: GET

Параметры: нет

1. Получение определенного окна:

URL: /window/id

Тип запроса: GET

Параметры: id – id окна, integer, обязательный

1. Получение окна по его размеру:

URL: /window/size\_id

Тип запроса: GET

Параметры: size\_id – размер окна, string, обязательный

1. Создание категории:

URL: /window/create

Тип запроса: POST

Параметры: в теле запроса принимается объект создаваемого окна с полем size (обязательный, string, размер окна)

**Пользователи**

1. Получение всех пользователей:

URL: /user/all

Тип запроса: GET

Параметры: нет

1. Вход пользователя:

URL: /user/login

Тип запроса: GET

Параметры: username – имя пользователя, string, обязательный; password – пароль пользователя, string, обязательный

1. Создание пользователя:

URL: /user/register

Тип запроса: POST

Параметры: username – имя пользователя, string, обязательный; password – пароль пользователя, string, обязательный

**Заказы**

1. Получение всех заказов:

URL: /order/all

Тип запроса: GET

Параметры: нет

1. Получение конкретного заказа по его id:

URL: /order/id

Тип запроса: GET

Параметры: id – id продукта, integer, обязательный

1. Получение заказов определенного пользователя:

URL: /order/user\_id

Тип запроса: GET

Параметры: idUser – id пользователя, integer, обязательный

1. Создание заказа

URL: /order/create

Тип запроса: POST

Параметры: в теле запроса принимается объект заказа с полями: idProduct (обязательный, integer, продукт в заказе), idUser (обязательный, integer, пользователь, сделавший заказ), price (обязательный, float, сумма заказа), orderDate (необязательный, string, дата заказа), status (обязательный, string, статус выполнения).

# **Работа с HTTP запросами**

Для выполнения http-запросов из Vue.js была выбрана библиотека Axios. При помощи axios.get(url).then(promise), можно совершить AJAX get-запрос по заданному url.

При помощи axios.post(url, body-dict).then(promise), можно совершить AJAX post-запрос по заданному url. body-dict представляет собой словарь, содержащий пары ключ значения переданных параметров.

Пример взаимодействия фронтенда с REST API:

Получение товара по его url (Product.vue):

created() {

const instance = Axios.create({

baseURL: 'http://localhost:1199/v1'

});

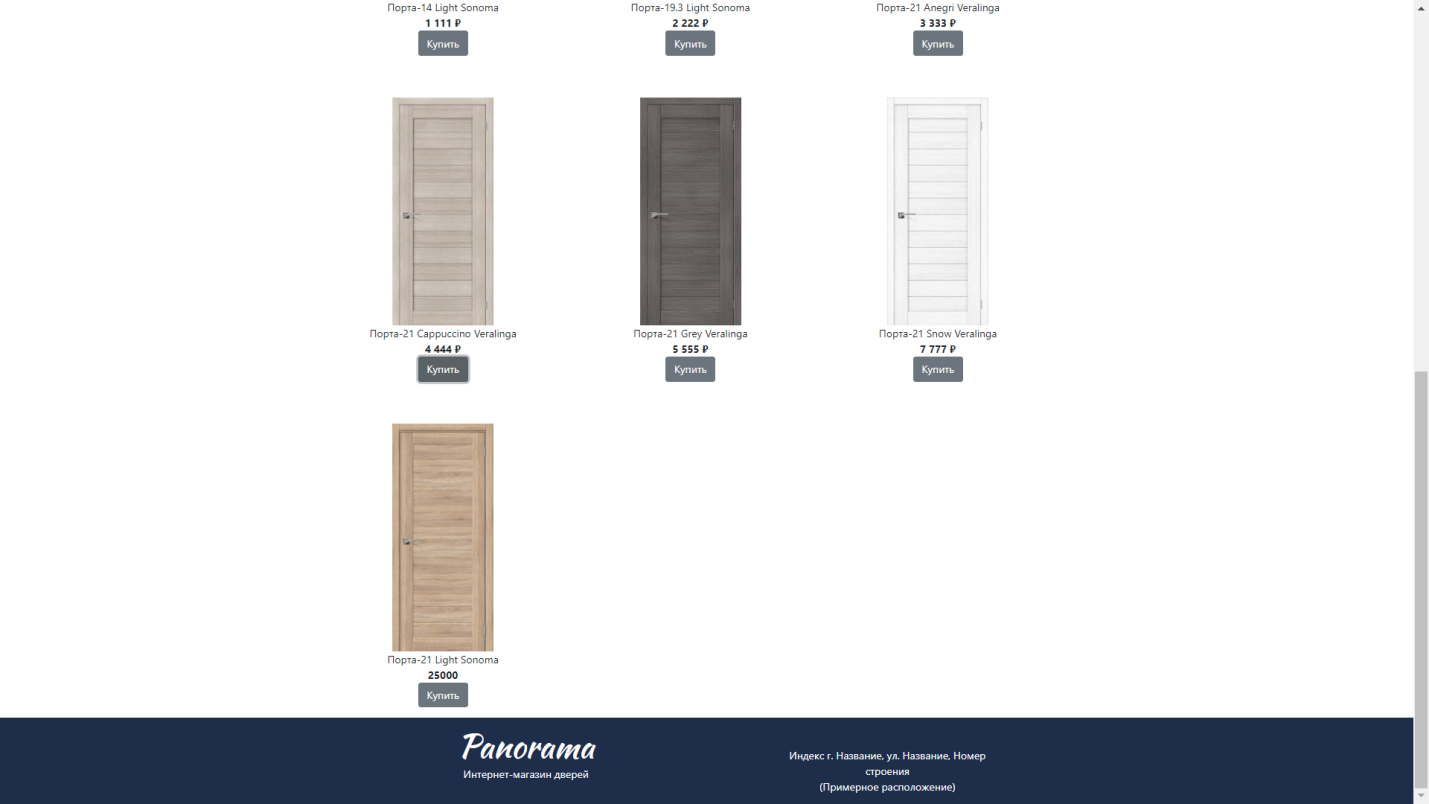
const url = this.$route.params.url

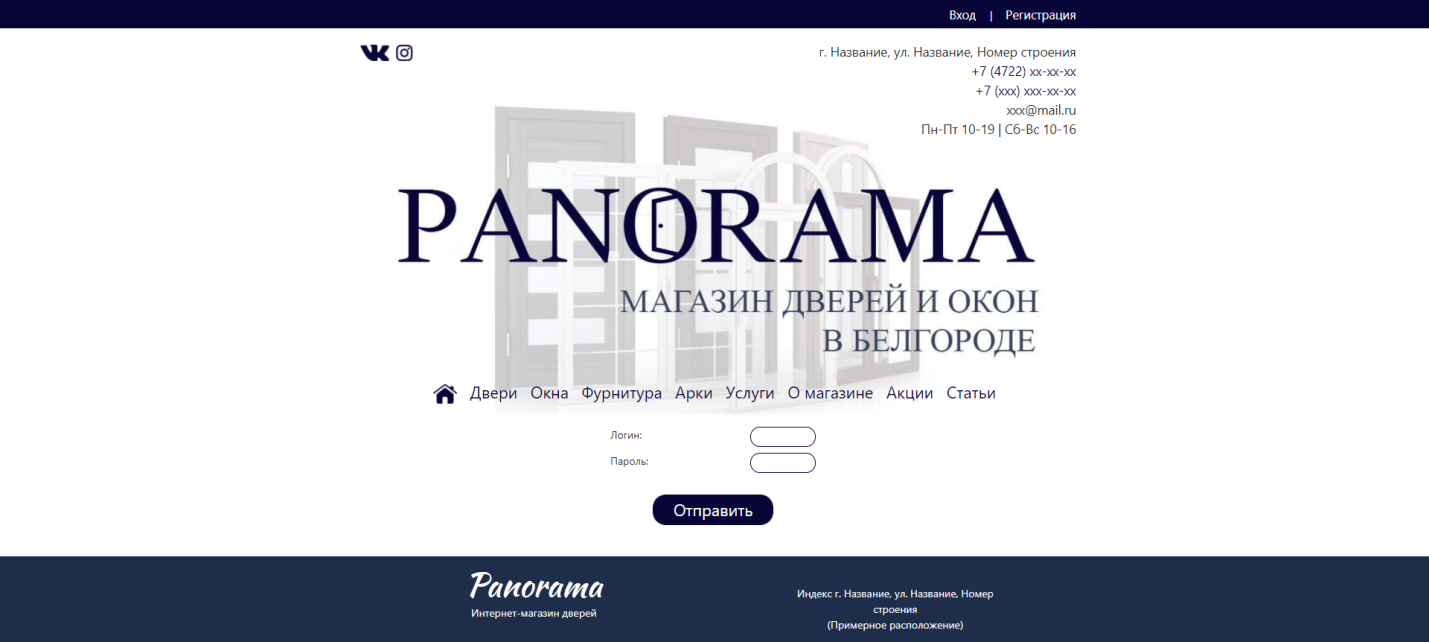
instance.get('/product/info', {params: {url: url}}) // product - контроллер home - метод

.then((response) => this.itemsProduct = response.data)

Полученный сайт:

****

****

**  **

**Вывод:** в результате выполнения работы разработали проект по выбранной теме (интернет-магазин дверей и окон), разработали макеты и сверстали страницы для web-приложения, перенесли вёрстку в отдельные компоненты фреймворка vuejs, настроили бекенд с помощью docker и yii2, спроектировали, создали и заполнили базу данных, реализовали REST API, связали фронтенд с бэкендом и базой данных.