**РУКОВОДСТВО ПРОГРАММИСТА**

**Интернет-магазин**

**«PlugLanguage»**

**Листов 18**

2022

**АННОТАЦИЯ**

В данном программном документе приведено руководство программиста по работе с интернет-магазином «PlugLanguage».

В разделе «Назначение и условия применения программы» указаны назначение и функции, выполняемые сайтом, условия, необходимые для его работы.

В разделе «Характеристика сайта» приведено описание основных

характеристик и особенностей.

1. Назначение и условия применения программы

Цель и назначение разработки информационной системы – создание сайта для реализации собственной продукции или для перепродажи купленной продукции посредством использования веб ресурсов, что позволит увеличить количество покупателей сэкономив на аренде или покупки площади для физической реализации.

Интернет-магазин в настоящее время имеет большой спрос, так как у людей не бывает свободного времени чтобы заехать в какой-то магазин или просто не хотят ходить по магазинам, они заказывают товар с доставкой на дом.

Поскольку именно через Интернет потенциальные клиенты будут искать информацию о том или ином товаре, даже перед тем, как пойти в магазин, то интернет-магазин будет им в помощь.

Основной целью сайта будет являться реализация продукции, получение прибыли

1. Характеристики и работа программы

Требования к логике форума, которые были выполнены в ходе разработки:

1. Вход на форум открыт для любого человека, имеющий доступ в интернет.
2. Регистрация, после чего авторизация, если аккаунт уже зарегистрирован, соответственно только авторизация, данный пункт обязательным только для админа, так как он даёт возможность добавлять товары, бренды и типы.
3. Создание нового товара на странице сайта.
4. Отправка информации о новых товарах в базу данных, для хранения.
5. Фильтрация списка товаров по брендам и типам.
6. Вывод товаров на сайте.

Для того, чтобы развернуть сайт требуется:

1. Скачать и установить Visual Studio Code по следующей ссылке <https://code.visualstudio.com/>
2. После того, как была установлена данная программная среда для веб-разработки, следует запустить её, нажать на кнопку «файл» и «открыть папку» как показано на рисунке 1.

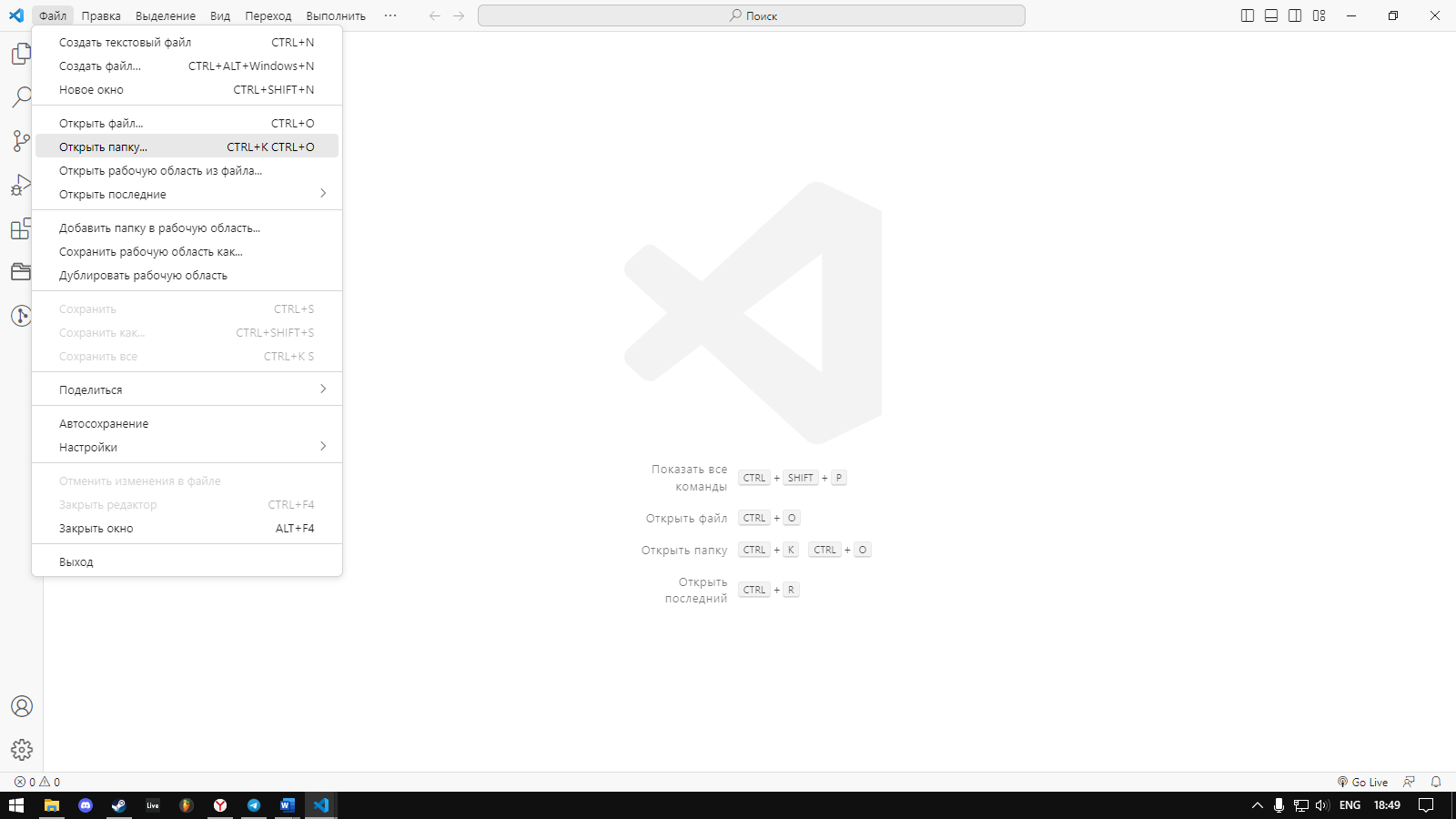


Рисунок 1 – Visual Studio Code

1. После этого, на компьютере выбрать папку, скачанную с гит хаба.
2. Далее нужно открыть два терминала, для этого нажать три кнопки в верхней панели приложения, нажать «Терминал» далее «создать терминал», представлено на рисунке 2, так откроется 1 терминал, чтобы открыть ещё один нужно нажать на плюсик как на рисунке 3.

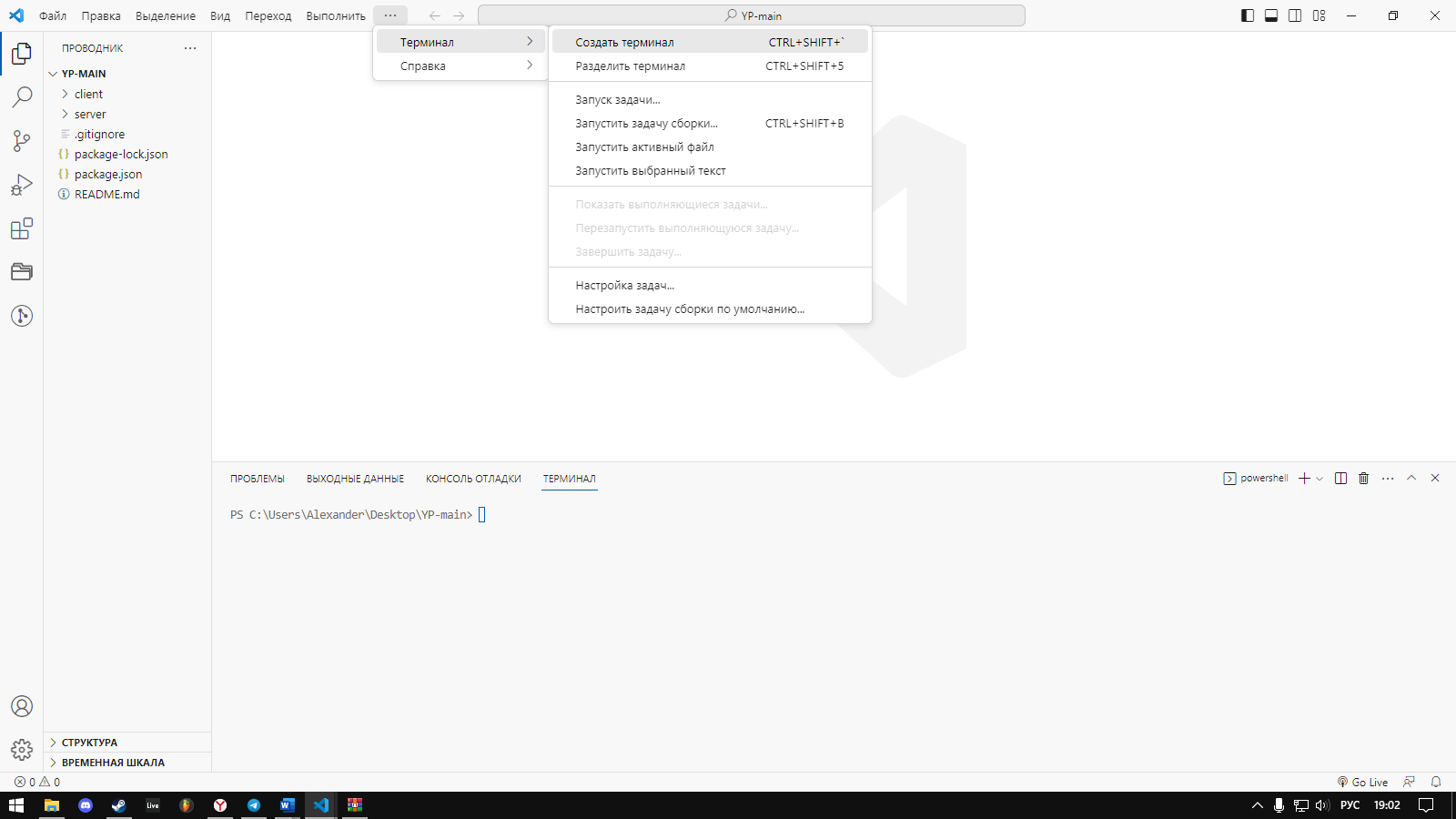


Рисунок 2 – Создание терминала

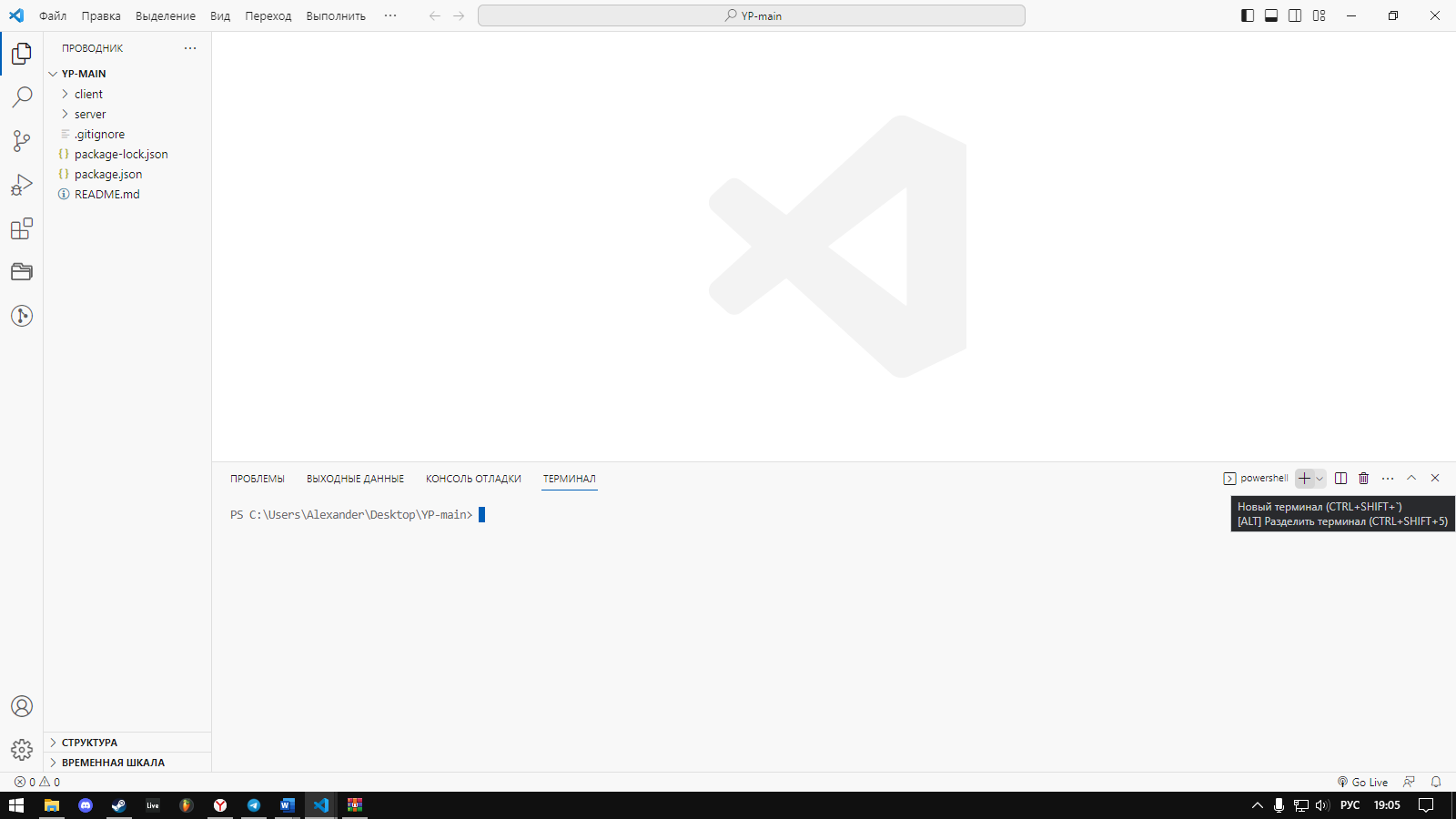


Рисунок 3 – Добавление второго терминала

1. После нажимаем на первый терминал, терминалы находятся ниже плюсика, вводим в командную строку «cd client» нажимаем enter и «npm install» нажимаем enter, представлено на рисунке 4, загружаются необходимые зависимости. Сделать то же самое на втором терминале, только первым написать «cd server» вместо «cd client».

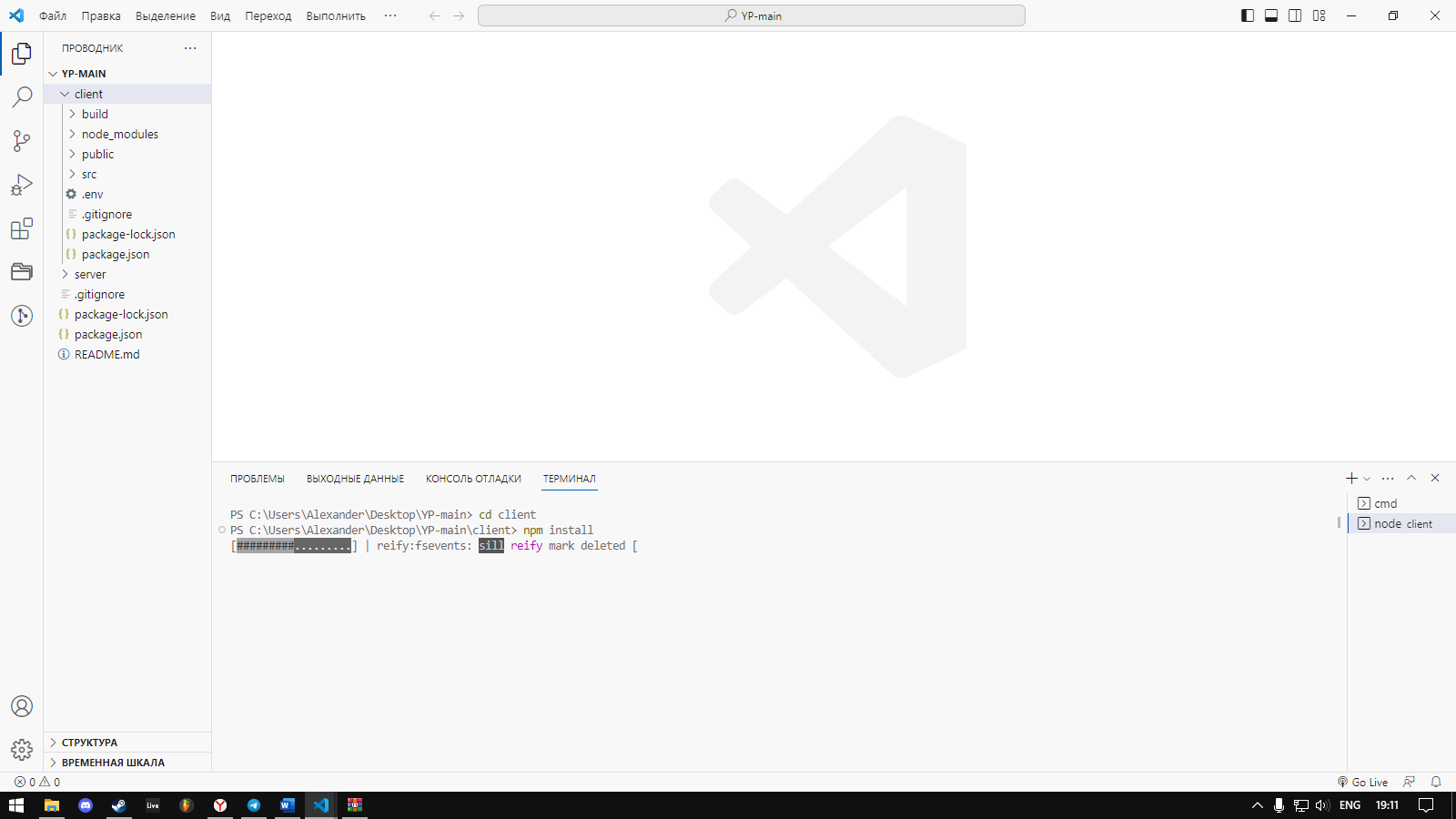


Рисунок 4 – Загрузка необходимых зависимостей

1. После того как загрузка завершилась, в терминале сервера пишем «npm start» нажимаем enter, как на рисунке 5, переходим в терминал с клиентом пишем «npm start» нажимаем enter, точно так же как на рисунке 6, после чего нас попросит подтвердить запуск на другом порте, так как на том сервер нажимаем «y», как на рисунке 7. После выполненных действий сайт запущен и откроется в браузере, либо вы перейдёте по его ссылке.

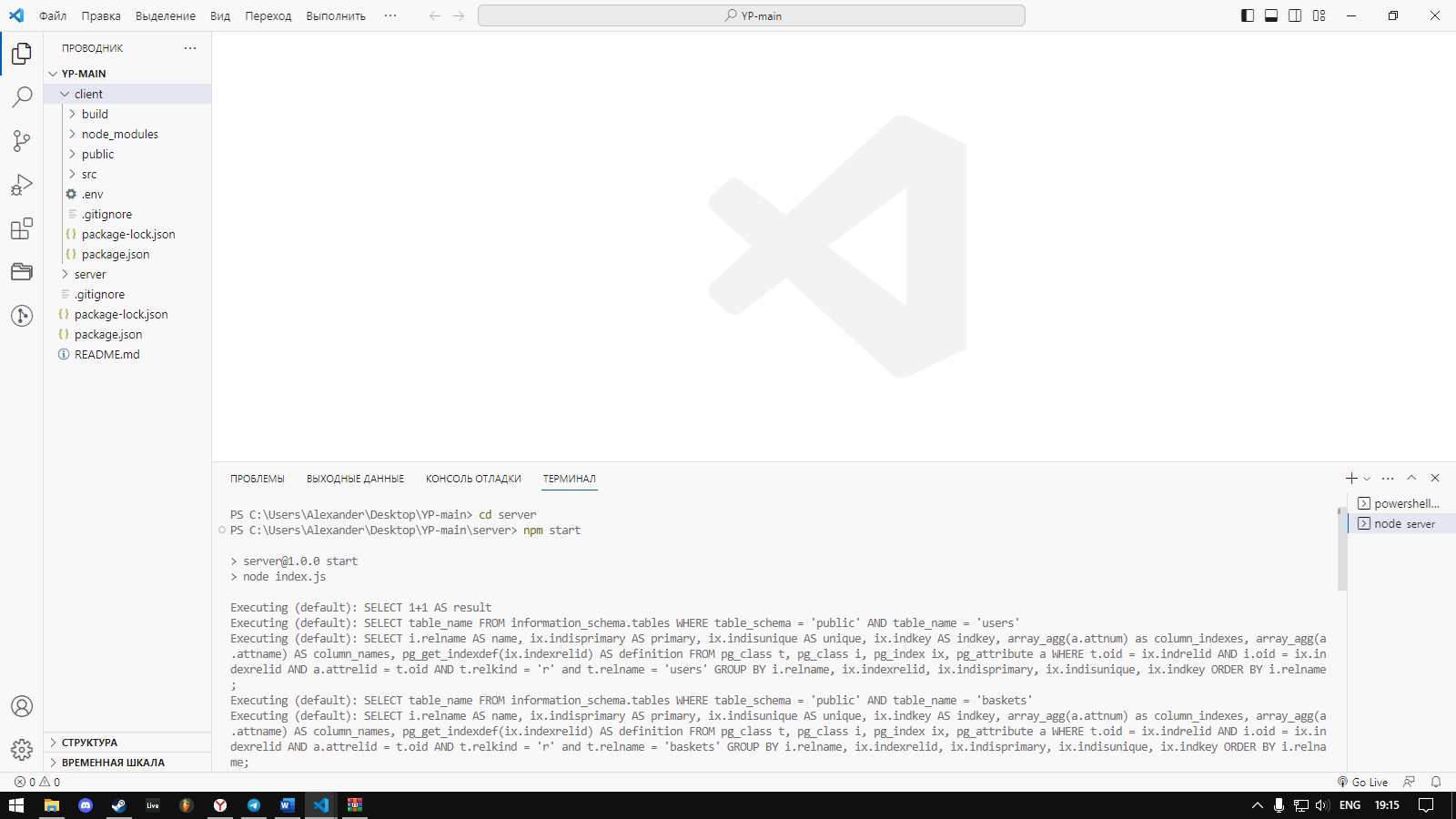


Рисунок 5 – Запуск сервера

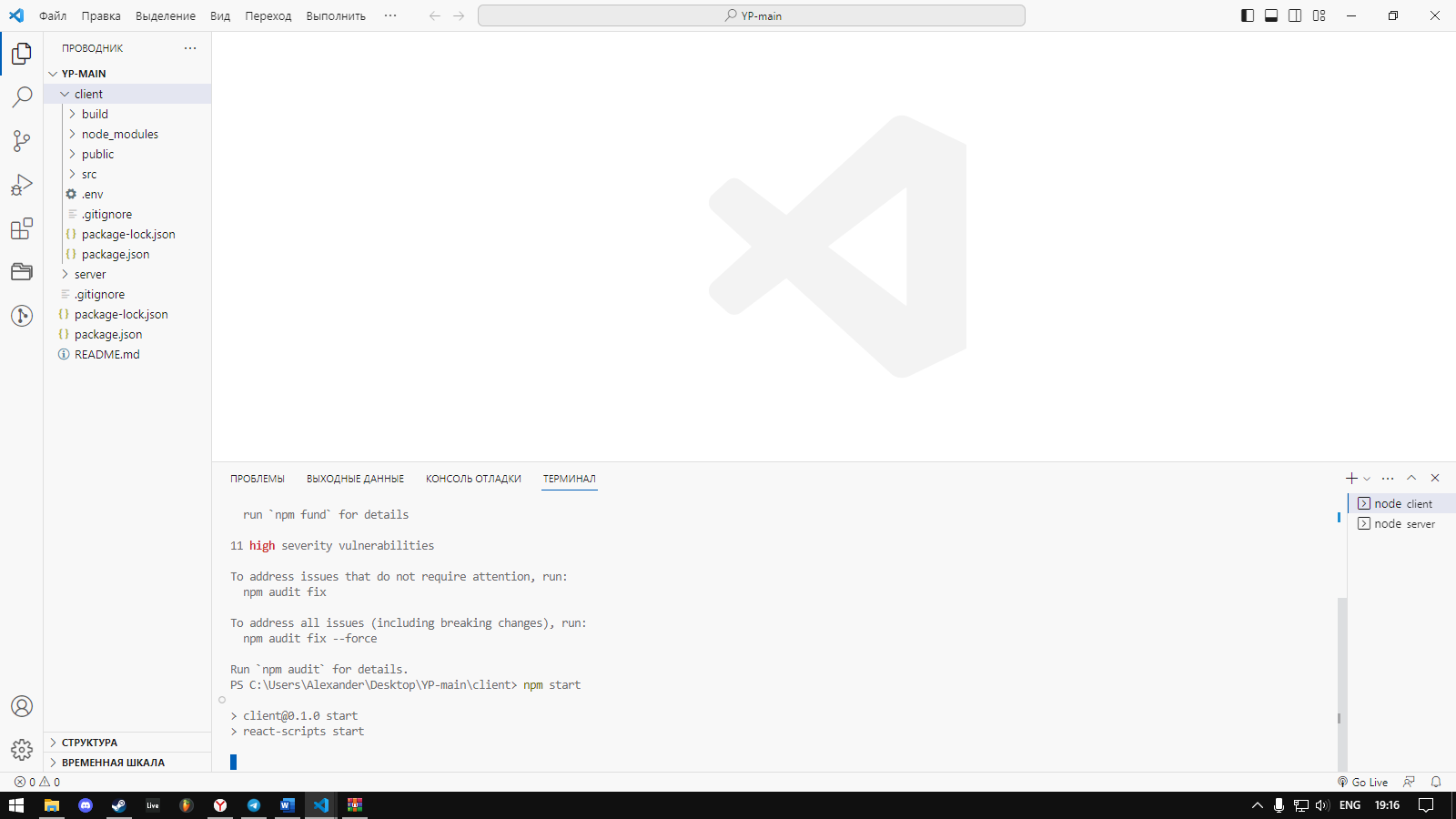


Рисунок 6 – Запуск клиента

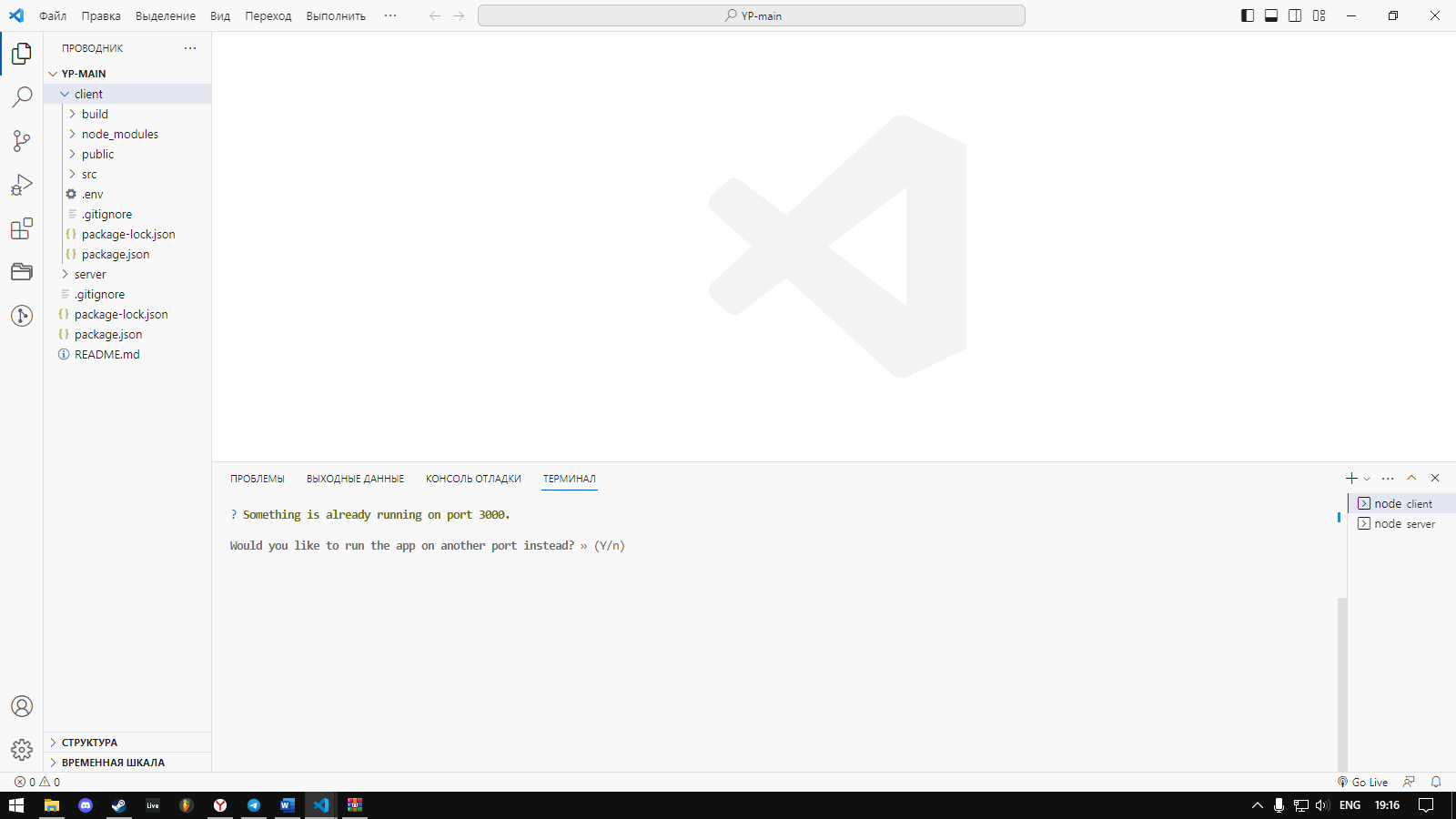


Рисунок 7 – Запуск клиента

На рисунке 8 представлена файловая архитектура сервера, а на рисунке 9 клиента, они составляют файловую архитектуру сайта.

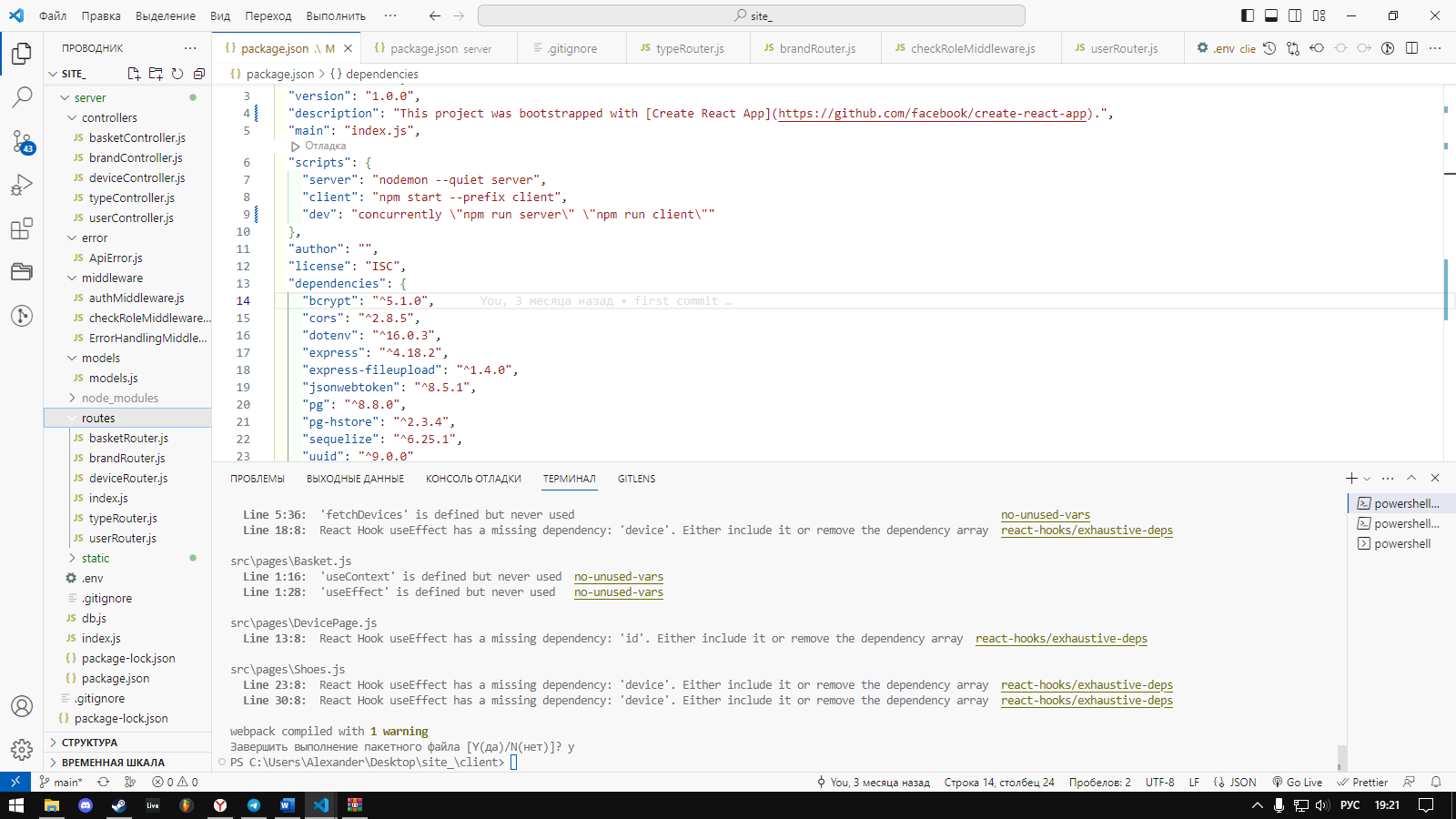


Рисунок 8 – Файловая архитектура сервера

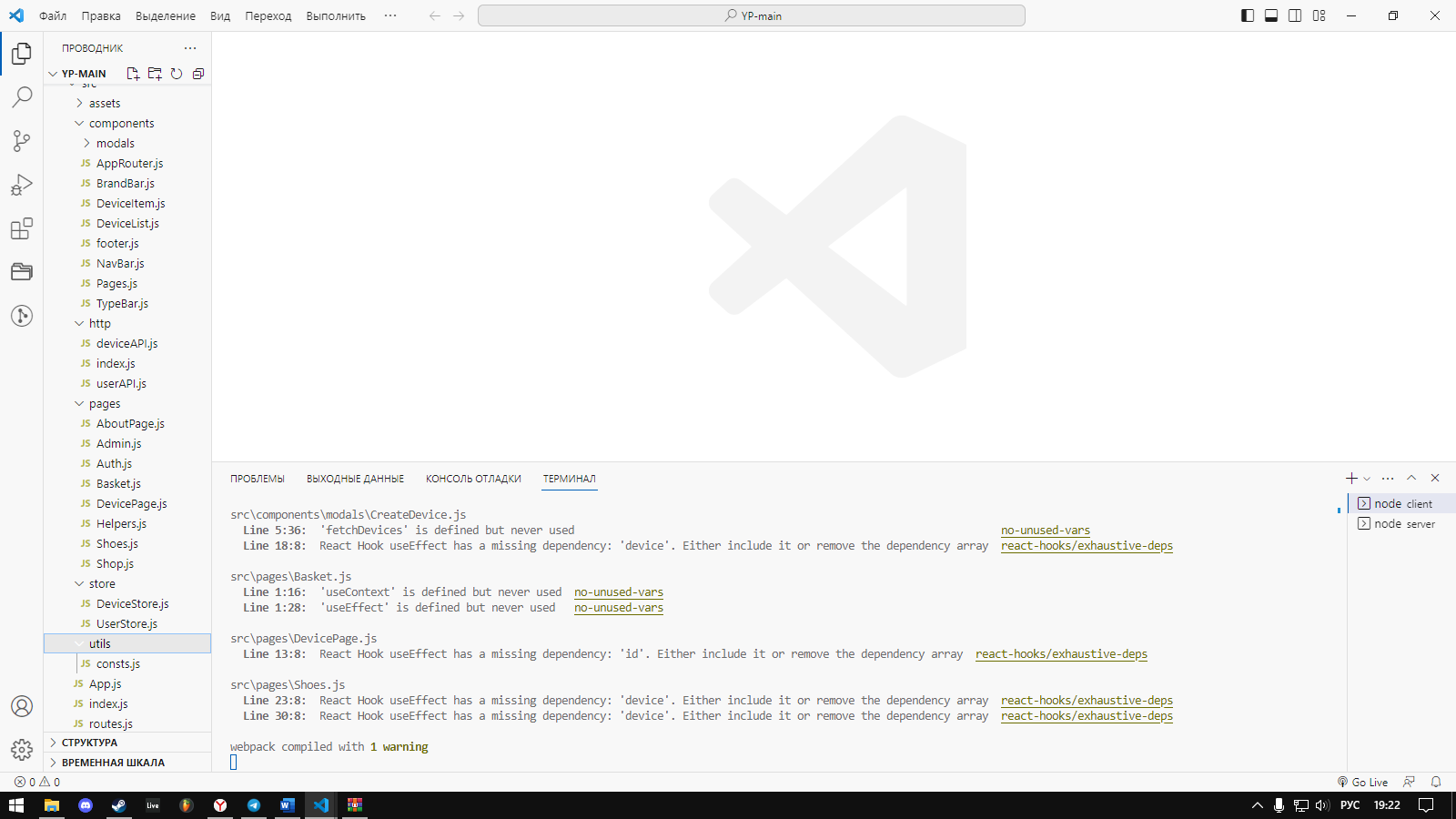


Рисунок 9 – Файловая архитектура клиента

Для работы форума необходимы все файлы, представленные на рисунках 8 и 9, их можно разделить на следующие группы:

1. Страницы сайта
2. Элементы сайта
3. Стили
4. Коды клиентской части
5. Коды серверной части
6. Картинки

При создании сайта использовались языки JS для фронт и бек разработки HTML для разметки страницы и CSS для применения единого шрифта сайта, на фронте использовался React и bootstrap, для построения визуальной структуры сайта.

Сайт состоит из восьми страниц AboutPage, Admin,Auth, Basket, DevicePage, helpers, Shop, Shoes, данные файлы используют расширение .js так как страницы написаны с помощю React’a, а это библеотека java script’a, остальные файлы являются исполняемым кодом или компонентами этих страниц, необходимым для функционирования продукта. Все стили для элементов сайта пишутся прямо в коде файла страницы, это позваоляет делать bootstrap от реакта и сам реакт. Также шаблонные части сайта и модальные окна вынесены в отдельные файлы и находятся в папке components туда входят подвал сайта, шапка сайта, бары, карточка товара, структура страницы товара, структура страницы каталога, это нужно для более читаемого и понятного кода, а также для динамического обновления информации.

При создании сайта учитывались 2 сценария: авторизован пользователь или нет. Если пользователь не авторизован, то у него не отображается какая-та информация на страницах сайта, а если авторизан соответственно отоброжается, но также у пользователей есть роли и роль может быть либо USER, либо ADMIN она также предоставляет свои права.

На рисунке 10 представлен вид сайта для неавторизованного пользователя.

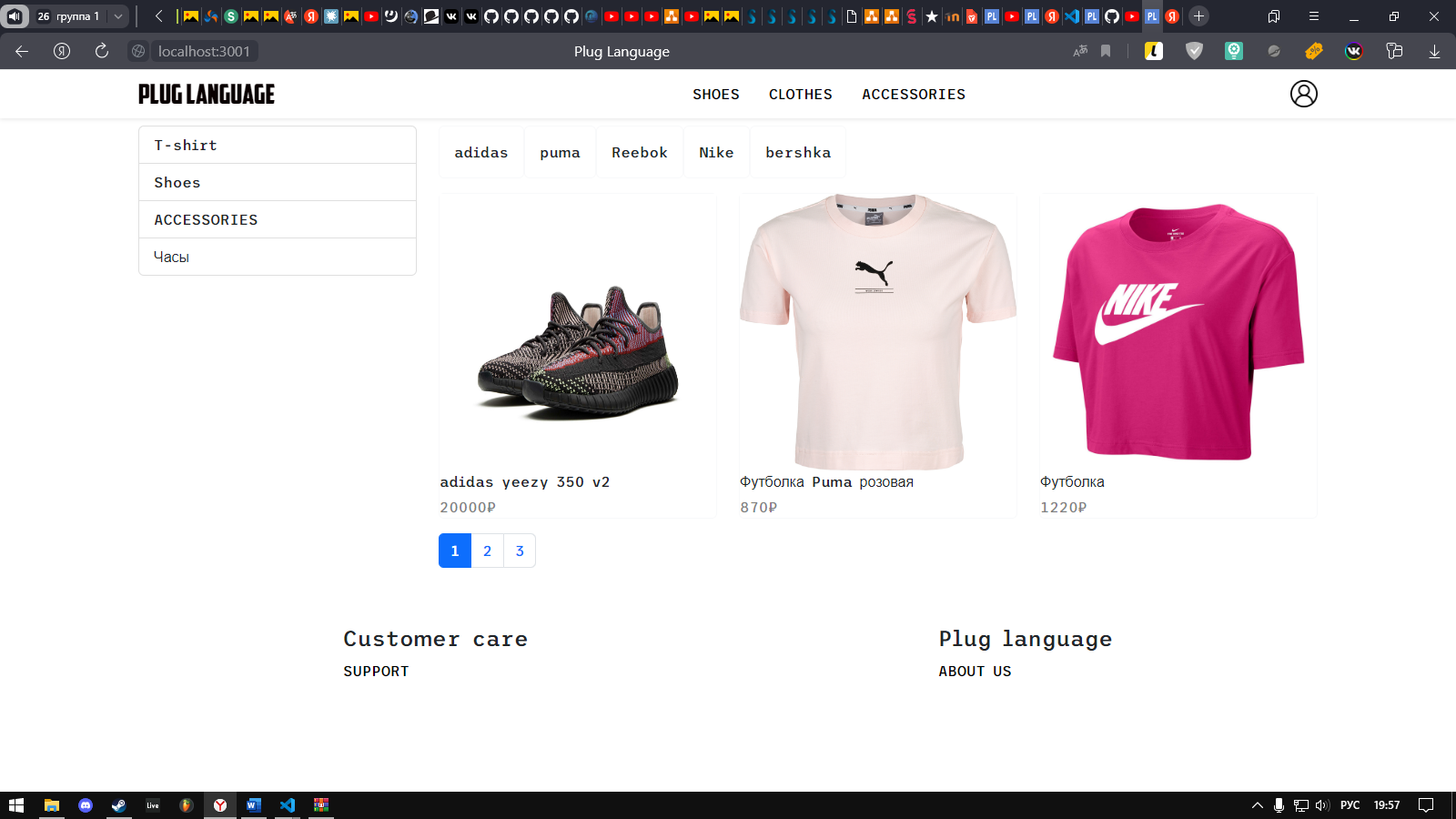


Рисунок 10 – Вид сайта для неавторизованного пользователя

На рисунке 11 представлен вид сайта для авторизированного пользователя.

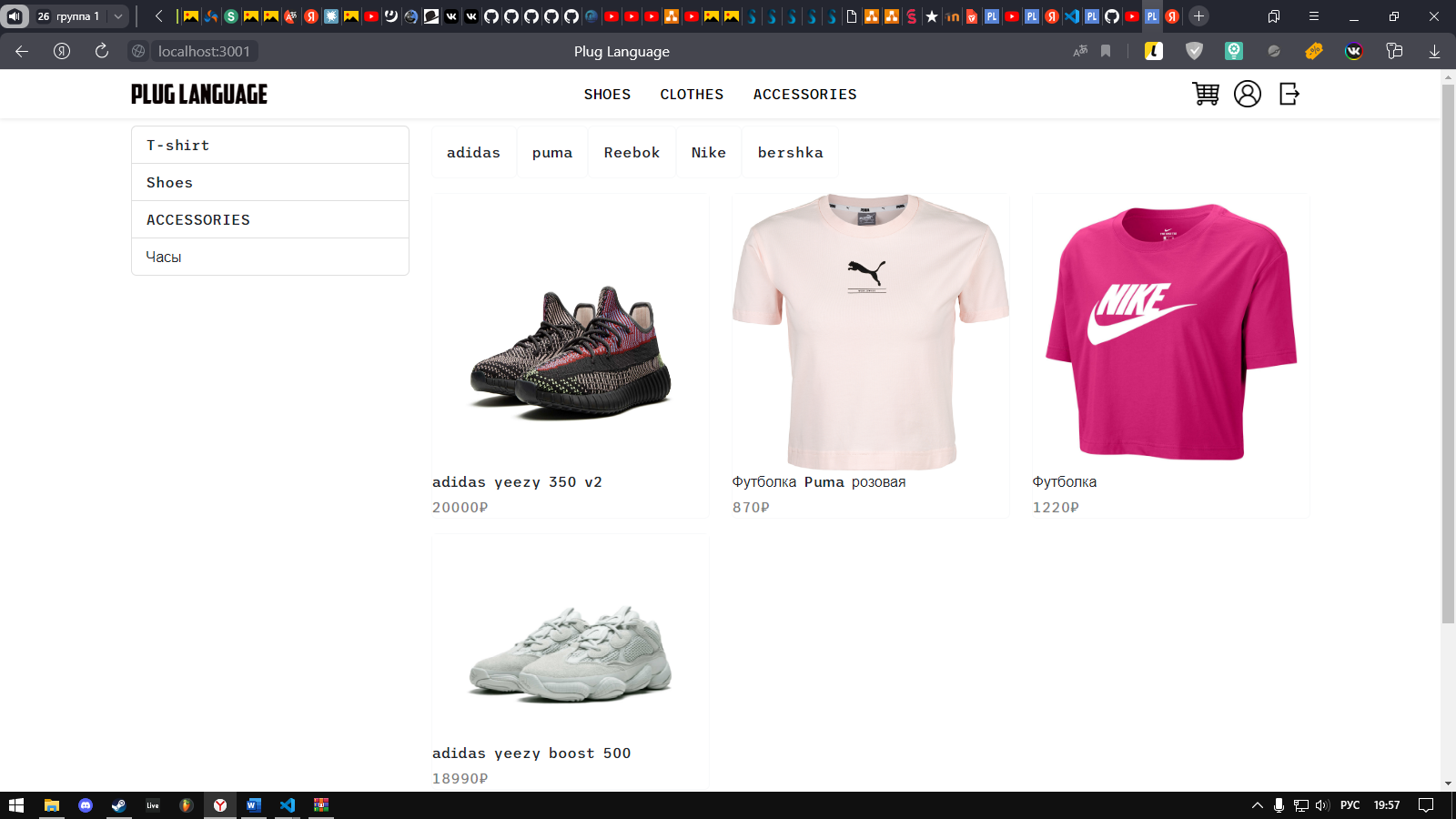


Рисунок 11 – Вид сайта для авторизованного пользователя

Также есть особенность у каждого аккаунта есть свой jwt токен, который позволяет не выходить из аккаунта пользователя, даже если он перезагрухит компьютер, токен действует 24 часа, потом надо будет снова авторизоваться.

На рисунке 12 предсталено динамическое отображение информации на сайте, в зависимости от авторизации.

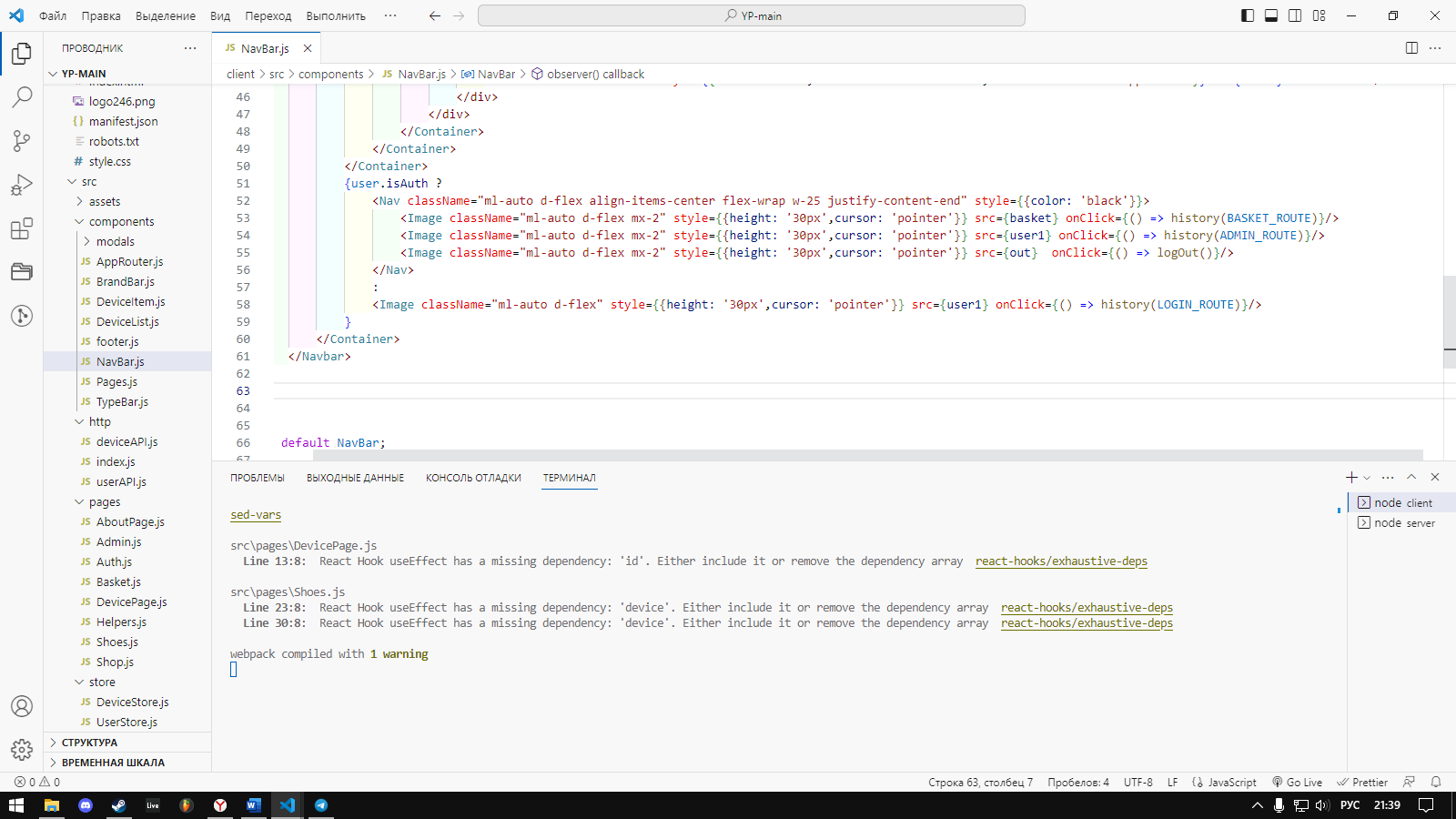


Рисунок 12 – Динамическое отображение кнопок в шапке сайта в зависимости от авторизации

На рисунке видно, что при нажатии на иконку «входа» осуществляется переход на страницу авторизации.

После авторизации становится доступен функционал добовления товаров, типов , брендов, но только для админов сайта. Для этого после авторизации необходимо нажать на иконку «Аккаунт», вследствие чего откроется страница админа, представленая на рисунке 13.

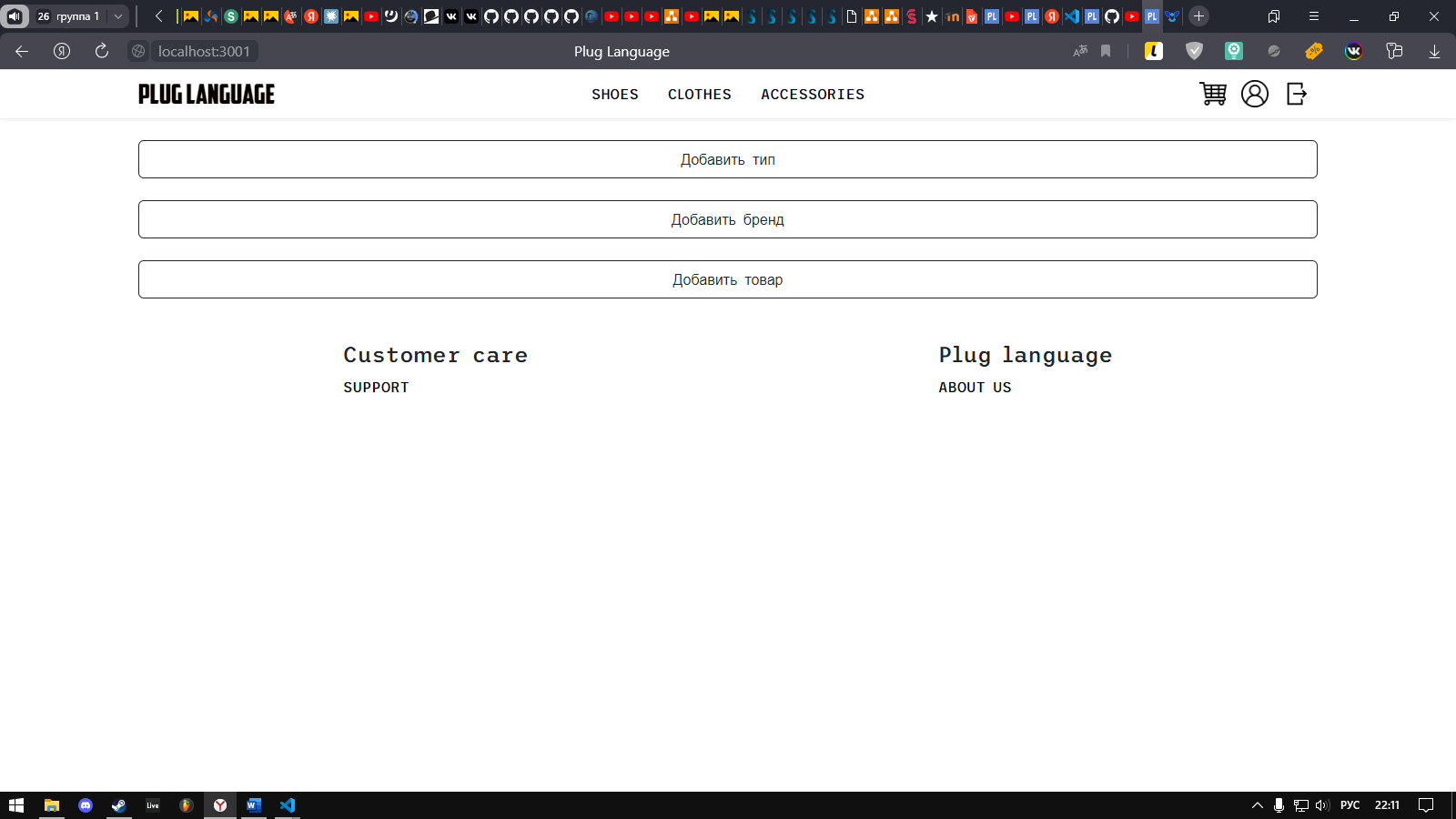


Рисунок 13 – Страница администратора

На рисунке 14 представлена страница администратора на которой администратор может добавить тип, товар, бренд, на рисунке видно что при нажатии на кнопку «Добавить (бренд, тип, товар)», появляется модальное окно соответствующее нажатой кнопке.

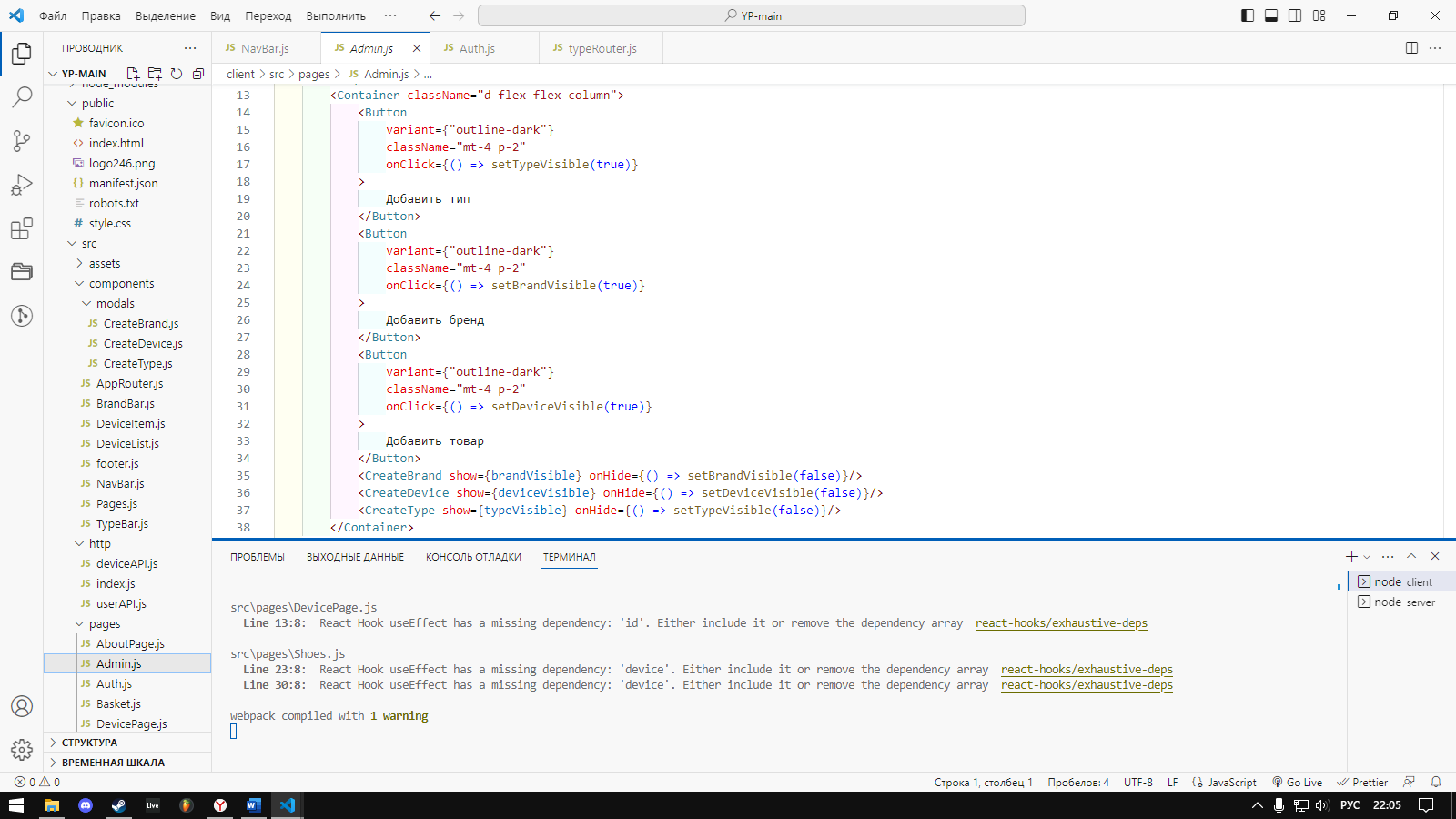


Рисунок 14 – Код на странице администратора

Чтобы добавить тип или бренд нужно нажать на кнопку «Добавить тип» или «Добавить бренд», после чего откроется модальное окно, в котором нужно ввести название типа или бренда и нажать кнопку «Добавить», вид модального окна представлен на рисунке 15.

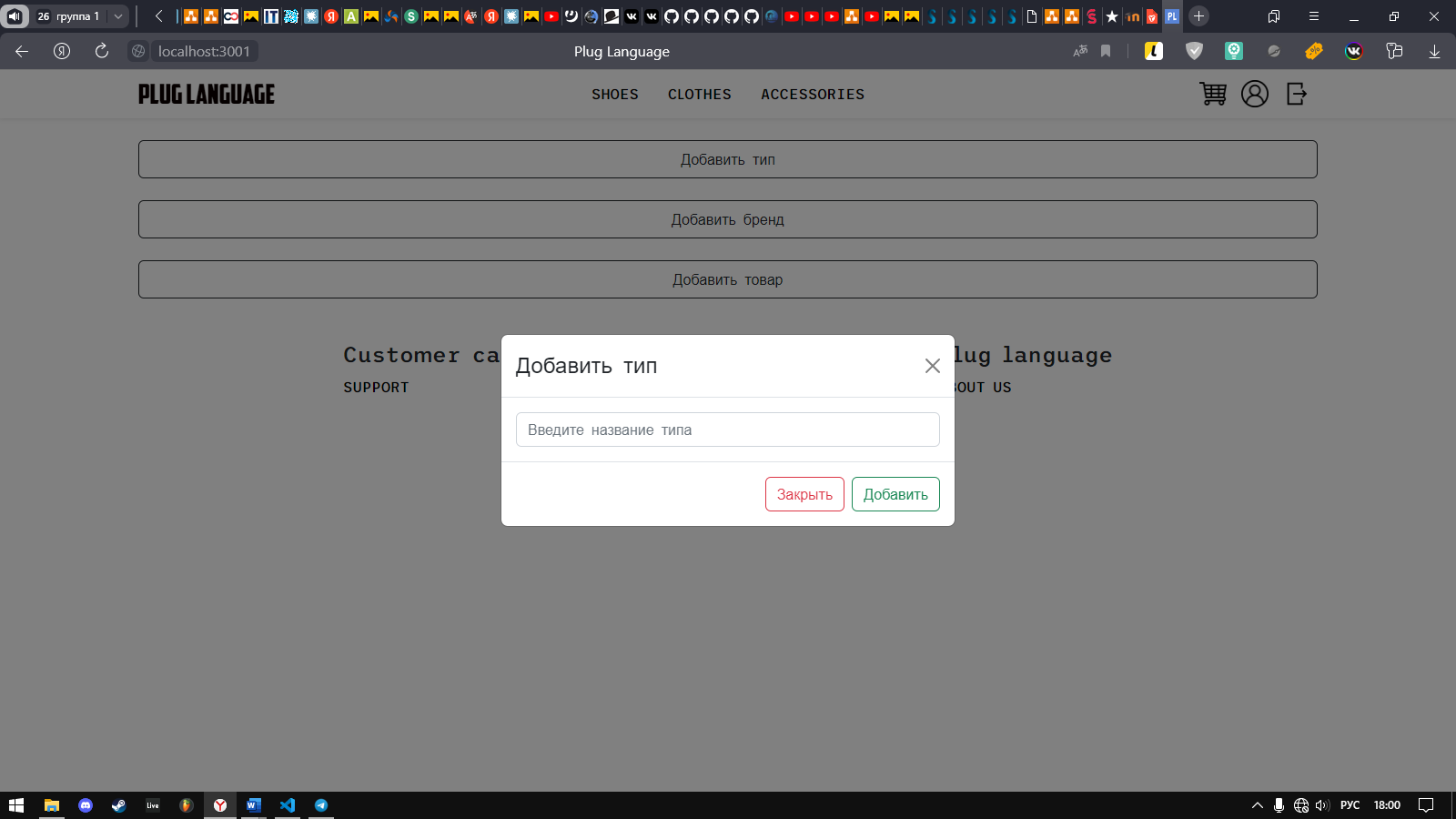


Рисунок 15 – Модальное окно добавление типа

Чтобы добавить товара нужно нажать на кнопку «Добавить товар», выбрать существующие тип и бренд, ввести название товара, его цену и добавить картинку товара, после чего нажать «Добавить». Чтобы добавить характеристики товара, нужно нажать на кнопку «Добавить новое свойство», ввести название характеристики и его описание.

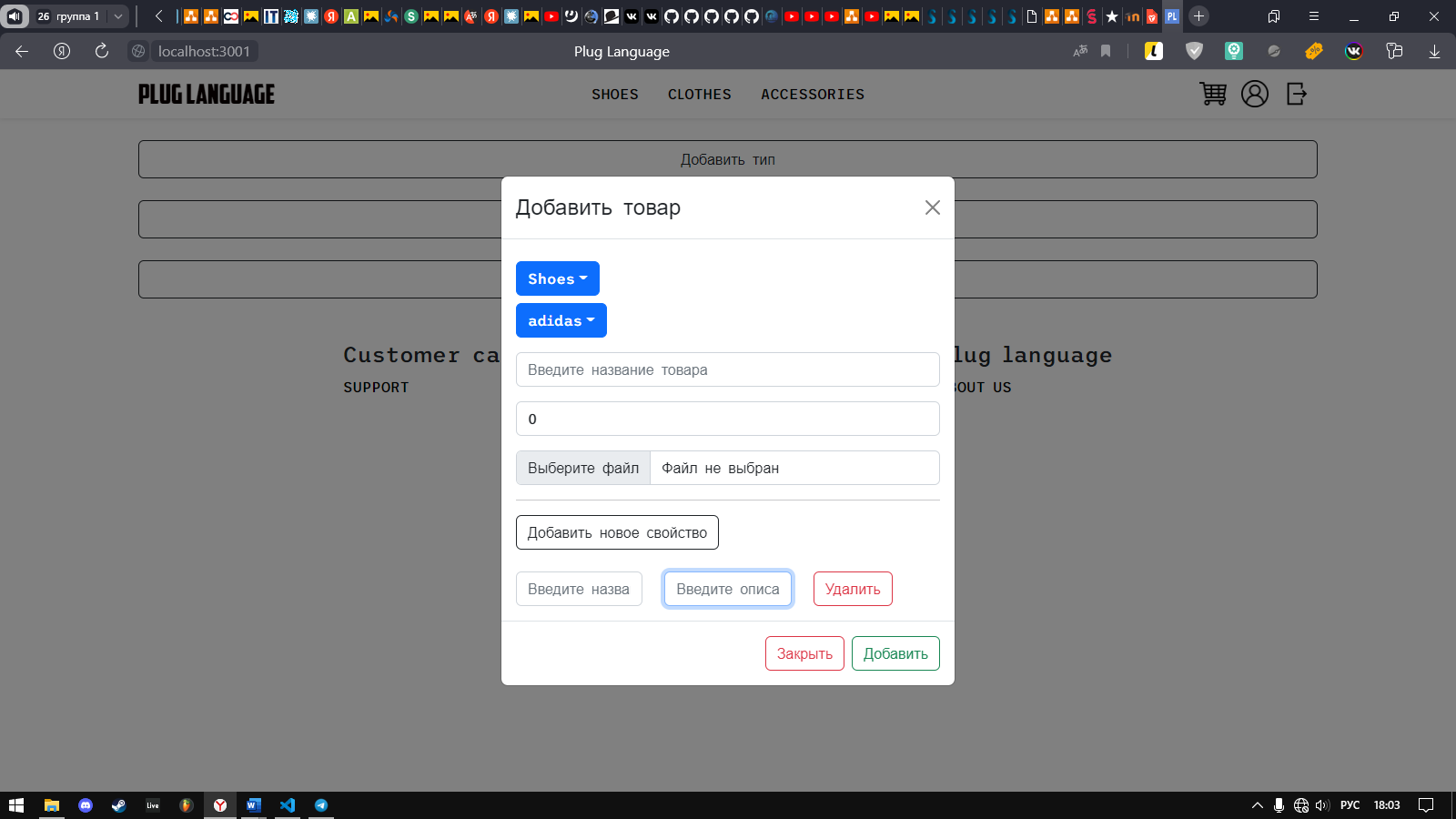


Рисунок 16 – Модальное окно добавление товара

При нажатии на кнопку «Добавить», введённые данные отправляются в базу данных.

На рисунке 17 представлен код добавления данных в базу данных.

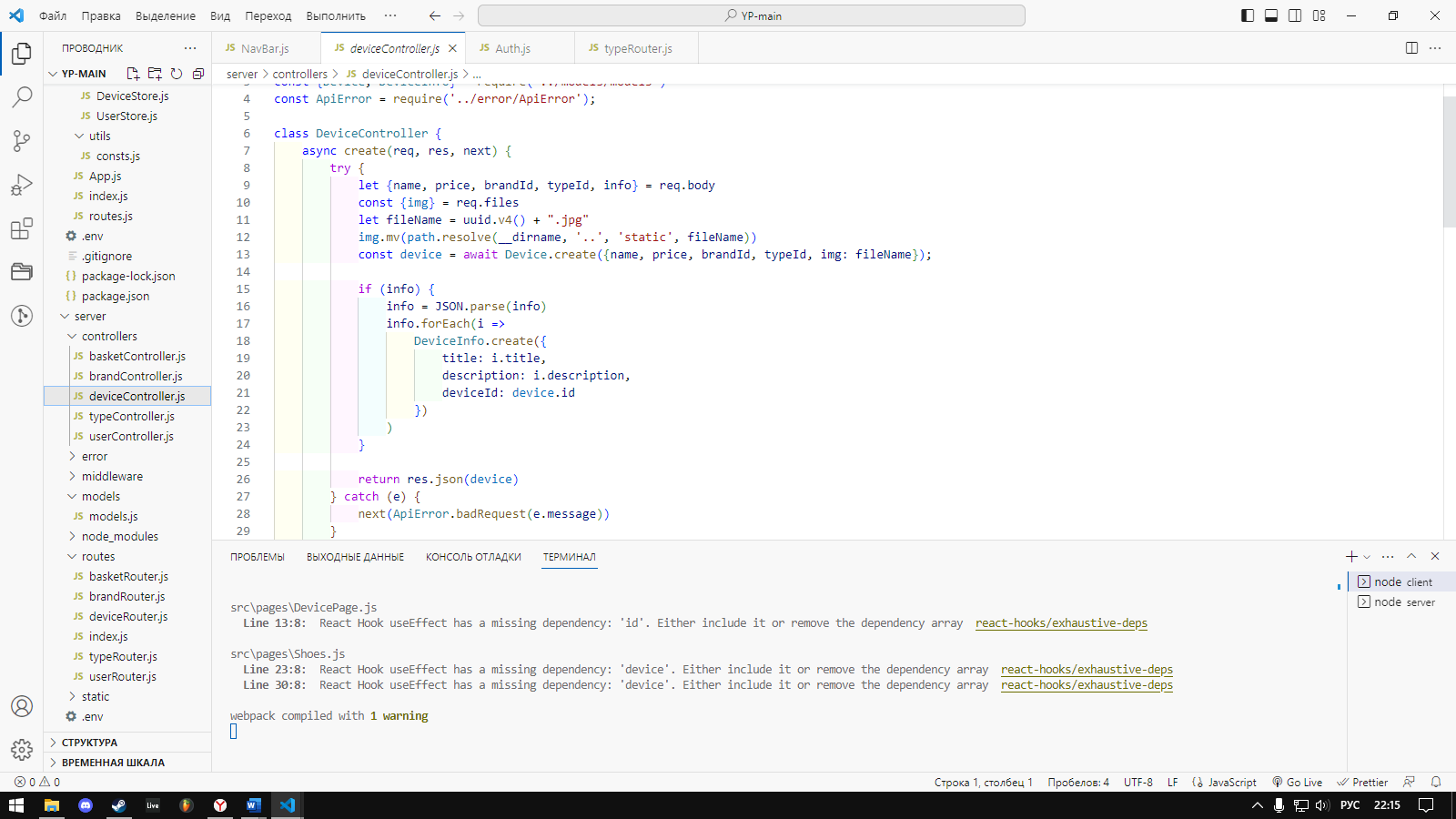


Рисунок 17 – Код добавления данных товара в базу данных

После того как данные пользователя приходят в базу данных, они сразу выводятся на сайте.

ЛИСТИНГ ПРОГРАММЫ

**«Добавление товара и получение»**

const uuid = require('uuid')

const path = require('path');

const {Device, DeviceInfo} = require('../models/models')

const ApiError = require('../error/ApiError');

class DeviceController {

    async create(req, res, next) {

        try {

            let {name, price, brandId, typeId, info} = req.body

            const {img} = req.files

            let fileName = uuid.v4() + ".jpg"

            img.mv(path.resolve(\_\_dirname, '..', 'static', fileName))

            const device = await Device.create({name, price, brandId, typeId, img: fileName});

            if (info) {

                info = JSON.parse(info)

                info.forEach(i =>

                    DeviceInfo.create({

                        title: i.title,

                        description: i.description,

                        deviceId: device.id

                    })

                )

            }

            return res.json(device)

        } catch (e) {

            next(ApiError.badRequest(e.message))

        }

    }

    async getAll(req, res) {

        let {brandId, typeId, limit, page} = req.query

        page = page || 1

        limit = limit || 9

        let offset = page \* limit - limit

        let devices;

        if (!brandId && !typeId) {

            devices = await Device.findAndCountAll({limit, offset})

        }

        if (brandId && !typeId) {

            devices = await Device.findAndCountAll({where:{brandId}, limit, offset})

        }

        if (!brandId && typeId) {

            devices = await Device.findAndCountAll({where:{typeId}, limit, offset})

        }

        if (brandId && typeId) {

            devices = await Device.findAndCountAll({where:{typeId, brandId}, limit, offset})

        }

        return res.json(devices)

    }

    async getOne(req, res) {

        const {id} = req.params

        const device = await Device.findOne(

            {

                where: {id},

                include: [{model: DeviceInfo, as: 'info'}]

            },

        )

        return res.json(device)

    }

}

module.exports = new DeviceController()