**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**



**МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

***Факультет Информационных технологий***

***Кафедра Информатики и информационных технологий***

**направление подготовки**

**09.03.02 «Информационные системы и технологии»**

**ЛАБОРАТОРНАЯ РАБОТА №** 14

**Дисциплина: «Backend»**

**Тема:** *Работа с кросс-доменными запросами веб-приложения на основе ASP.NET Core*

**Выполнил: студент группы: 231-339**

Карапетян Нвер Каренович\_\_\_\_\_\_\_\_\_\_\_

(Фамилия И.О.)



**Дата, подпись:** 21.03.25 \_\_\_\_\_\_\_\_\_\_\_\_

(Дата)(Подпись)

**Проверил:** \_\_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_*

(Фамилия И.О., степень, звание) (Оценка)

**Дата, подпись** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_\_\_*

(Дата)(Подпись)

**Москва**

**2025**

Цель:

Ознакомиться с работой с кросс-доменными запросами (CORS) в веб-приложениях на платформе ASP.NET Core для обеспечения безопасности и разрешения запросов с других источников**.**

Задачи:

* Настроить веб-приложение на ASP.NET Core для поддержки кросс-доменных запросов.
* Реализовать необходимые middleware или атрибуты для управления CORS-политиками.
* Протестировать работу приложения, отправляя запросы с различных источников и проверяя их успешное выполнение.

Ход работы

В современных веб-приложениях фронтенд и бэкенд часто развертываются на разных доменах. Это создает проблему кросс-доменных запросов, которые браузеры блокируют по умолчанию в соответствии с политикой безопасности Same-Origin Policy. Для легализации таких запросов используется механизм CORS (Cross-Origin Resource Sharing). В данной работе рассмотрена настройка CORS-политик в ASP.NET Core и их интеграция с клиентским приложением на React.

Настройка и применение CORS-политик

**Cross-Origin Resource Sharing (CORS)** — механизм, использующий дополнительные HTTP-заголовки, чтобы дать возможность агенту пользователя получать разрешения на доступ к выбранным ресурсам с сервера на источнике (домене), отличном от того, что сайт использует в данный момент.

В целях безопасности браузеры ограничивают Cross-Origin запросы, инициируемые скриптами. Например, XMLHttpRequest и Fetch API следуют политике одного источника (Same-Origin Policy). Это значит, что WEB-приложения, использующие такие API, могут запрашивать HTTP-ресурсы только с того домена, с которого были загружены, пока не будут использованы CORS-заголовки.

Для примера создадим небольшой проект на React, напишем простенький React-компонент, который будет представлять из себя таблицу, аналогичную таблице «Products» из нашей базы данных. С помощью fetch-запроса в хуке «useEffect» обратимся к нашей API, а именно к эндпоинту «api/products» и получим записи продуктов «data» из базы данных:

Листинг 1. Хук useEffect, выполняющий запрос на backend.

useEffect(() => {

        fetch(“https://localhost:7039/api/products”)

            .then(response => {

                if (!response.ok)

                    throw new Error(‘Ошибка загрузки данных’)

                return response.json()

            })

            .then(data => {

                setProducts(data)

            })

            .catch(err => {

                throw new Error(err)

            })

    }, []);

Теперь, если мы запустим наше веб-приложение, то обнаружим на странице пустую таблицу и ошибку (см. рис. 1).

Ошибка возникает из-за того, что API и веб-приложение располагаются на разных портах — 7039 и 5173 соответственно. И по умолчанию такие запросы блокируются.

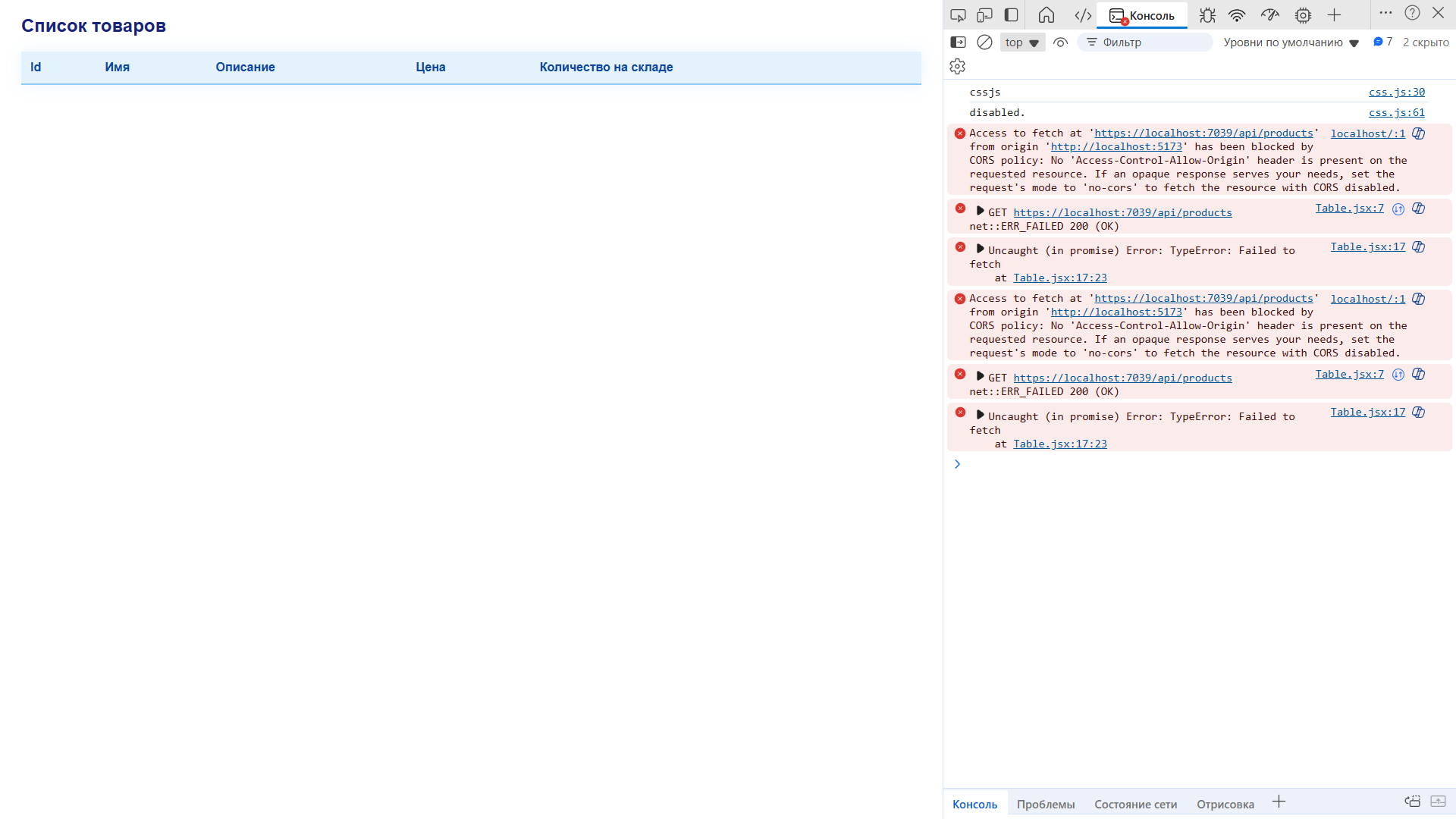


Рисунок . Ошибки при обращении к нашему API.

Для того, чтобы разрешить кросс-доменные запросы, необходимо добавить на бэкенде в основном скрипте «Program.cs» следующую настройку CORS:

Листинг 2. Создание и применение CORS-политики.

builder.Services.AddCors(options =>

    {

        options.AddPolicy("React-App", policy =>

        {

            policy.WithOrigins("http://localhost:5173");    // Адрес, на котором локально запускается наше веб-приложение

            policy.AllowAnyHeader();        // Разрешает использование любых заголовков в запросах.

            policy.AllowAnyMethod();        // Разрешает использование любых HTTP-методов (GET, POST, PUT, DELETE и т.д.).

        });

    });

...

app.UseCors("React-App");       // Применяем настройки CORS

Здесь используется три стандартных CORS-метода: WithOrigins, AllowAnyHeader и AllowAnyMethod.

Метод **WithOrigins** используется для явного указания доменов, с которых разрешены кросс-доменные запросы. В качестве параметра принимает строку или массив строк, каждая из которых представляет допустимый источник (origin). Например, policy.WithOrigins("http://localhost:5173") разрешает запросы только с домена http://localhost:5173. Это критически важно для безопасности, так как ограничивает доступ к API только доверенными клиентами. Если требуется разрешить несколько доменов, их можно перечислить через запятую: policy.WithOrigins("https://localhost:5173", "https://admin.localhost:5173").

Метод **AllowAnyHeader** разрешает клиенту отправлять любые HTTP-заголовки в запросе. Это упрощает разработку, так как не требует явного указания заголовков, таких как Content-Type или Authorization. Однако в production-среде рекомендуется использовать метод WithHeaders, который принимает массив строк с названиями разрешенных заголовков. Например, policy.WithHeaders("Content-Type", "Authorization") ограничивает заголовки только необходимыми, снижая риск злоупотреблений. Использование AllowAnyHeader оправдано только на этапе тестирования или для API с открытым доступом.

Метод **AllowAnyMethod** разрешает все HTTP-методы (GET, POST, PUT, DELETE и др.). Это удобно для API, поддерживающего полный спектр операций CRUD. Однако для повышения безопасности следует ограничить методы через WithMethods, указав только необходимые. Например, policy.WithMethods("GET", "POST") разрешит только чтение и создание данных. Такой подход минимизирует поверхность атаки, запрещая неиспользуемые методы, такие как DELETE или PUT.

Таким образом, произведя настройку кросс-доменных запросов с помощью пользовательской CORS-политики, мы разрешаем ошибку, возникшую на рис. 1. Перезагрузив страницу, мы увидим нашу страницу с таблицей данных, полученных из базы данных, запросом с бэкенда:

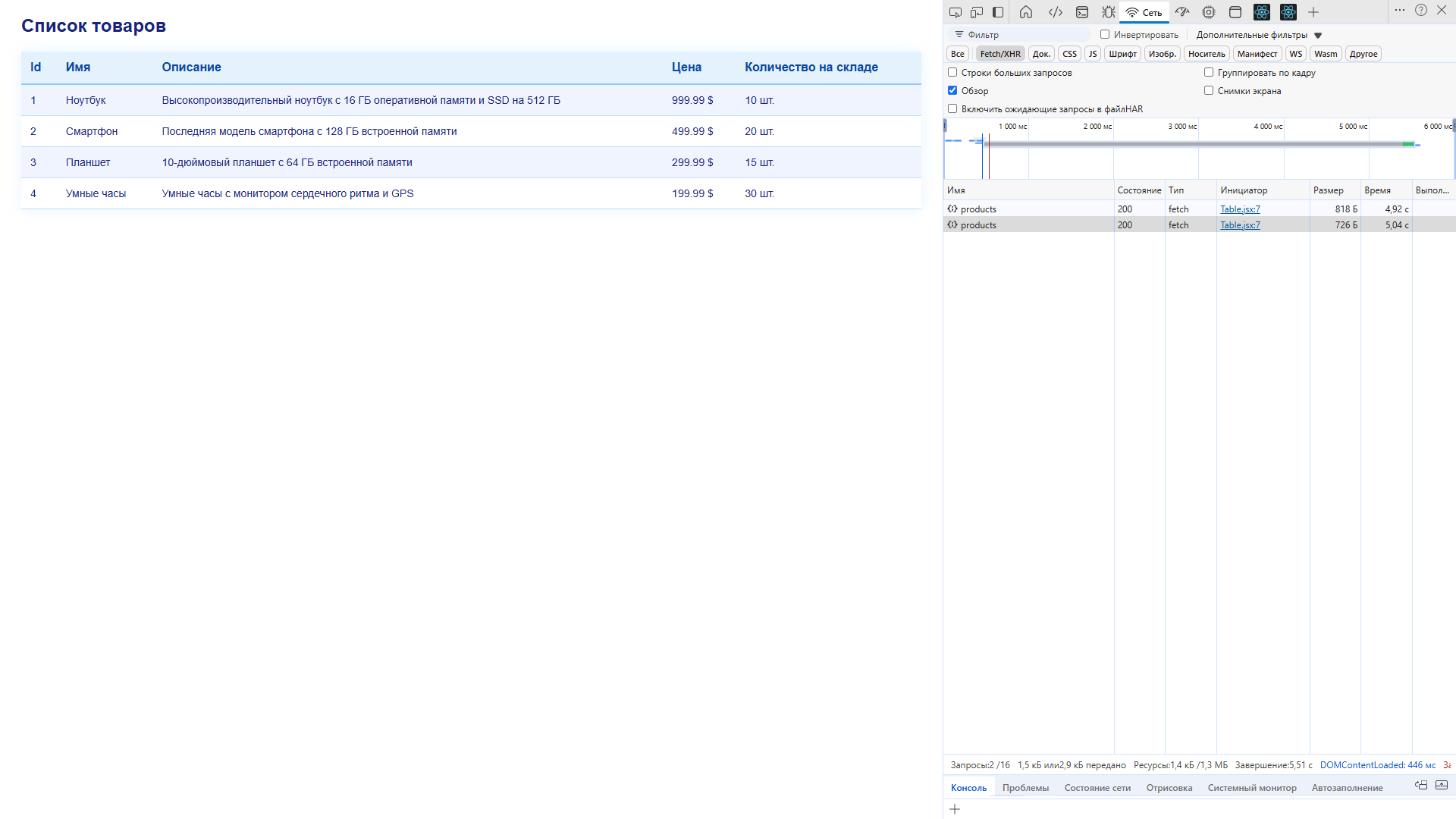


Рисунок . Успешный запрос.

Приложение

Листинг 3. Скрипт Program.cs.

using Laba14.Models;

using Microsoft.EntityFrameworkCore;

using Scalar.AspNetCore;

namespace Laba14

{

    public class Program

    {

        public static void Main(string[] args)

        {

            var builder = WebApplication.CreateBuilder(args);

            builder.Services.AddDbContext<ProductDbContext>(options =>

                options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection")));

            builder.Services.AddCors(options =>

            {

                options.AddPolicy("React-App", policy =>

                {

                    policy.WithOrigins("http://localhost:5173");    // Адрес, на котором локально запускается наше веб-приложение

                    policy.AllowAnyHeader();        // Разрешает использование любых заголовков в запросах.

                    policy.AllowAnyMethod();        // Разрешает использование любых HTTP-методов (GET, POST, PUT, DELETE и т.д.).

                });

            });

            builder.Services.AddControllers();

            builder.Services.AddOpenApi();

            var app = builder.Build();

            if (app.Environment.IsDevelopment())

            {

                app.MapOpenApi();

                app.MapScalarApiReference();

            }

            app.UseHttpsRedirection();

            app.UseAuthorization();

            app.UseCors("React-App");       // Применяем настройки CORS

            app.MapControllers();

            app.Run();

        }

    }

}

Листинг 4. Скрипт Table.jsx.

import  { useState, useEffect } from "react";

const Table = () => {

    const [products, setProducts] = useState([])

    useEffect(() => {

        fetch("https://localhost:7039/api/products")

            .then(response => {

                if (!response.ok)

                    throw new Error('Ошибка загрузки данных')

                return response.json()

            })

            .then(data => {

                setProducts(data)

            })

            .catch(err => {

                throw new Error(err)

            })

    }, []);

    return (

        <div style={{margin: '20px', fontFamily: 'Arial, sans-serif'}}>

            <h2 style={{color: '#1a237e'}}>Список товаров</h2>

            <table style={{width: '100%', borderCollapse: 'collapse', boxShadow: '0 0 20px rgba(33,150,243,0.1)'}}>

                <thead>

                <tr style={{backgroundColor: '#e3f2fd'}}>

                    <th style={tableHeaderStyle}>Id</th>

                    <th style={tableHeaderStyle}>Имя</th>

                    <th style={tableHeaderStyle}>Описание</th>

                    <th style={tableHeaderStyle}>Цена</th>

                    <th style={tableHeaderStyle}>Количество на складе</th>

                </tr>

                </thead>

                <tbody>

                {products.map((product, index) => (

                    <tr key={product.id} style={index % 2 === 0 ? evenRowStyle : oddRowStyle}>

                        <td style={tableCellStyle}>{product.id}</td>

                        <td style={tableCellStyle}>{product.name}</td>

                        <td style={tableCellStyle}>{product.description}</td>

                        <td style={tableCellStyle}>{product.price} $</td>

                        <td style={tableCellStyle}>{product.stock} шт.</td>

                    </tr>

                ))}

                </tbody>

            </table>

        </div>

    )

}

const tableHeaderStyle = {

    padding: '12px',

    textAlign: 'left',

    borderBottom: '2px solid #90caf9',

    fontSize: '16px',

    color: '#0d47a1',

    fontWeight: '600'

}

const tableCellStyle = {

    padding: '12px',

    textAlign: 'left',

    borderBottom: '1px solid #bbdefb',

    fontSize: '14px',

    color: '#1a237e'

}

const evenRowStyle = {

    backgroundColor: '#f0f4ff'

}

const oddRowStyle = {

    backgroundColor: '#fff',

    transition: 'background-color 0.3s ease',

    ':hover': {

        backgroundColor: '#e3f2fd'

    }

}

export default Table