**CORS и кросс-доменные запросы**

**Подключение CORS в приложении**

По умолчанию веб-браузеры в целях безопасности ограничивают ajax-запросы между различными доменами. Однако нередко возникает ситуация, когда необходимо выполнять запросы из приложения с одного адреса (или домена) к приложению, которое размещено по другому адресу. Для этого нам надо использовать технику, которая называется CORS (Cross Origin Resource Sharing).

### Добавление и настройка сервисов CORS

Возьмем простейший проект ASP.NET Core по типу Empty и в файле **Program.cs** определим следующий код:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var builder = WebApplication.CreateBuilder();    builder.Services.AddCors(); // добавляем сервисы CORS    var app = builder.Build();    // настраиваем CORS  app.UseCors(builder => builder.AllowAnyOrigin());    app.Map("/", async context => await context.Response.WriteAsync("Hello METANIT.COM!"));    app.Run(); |

Для подключения сервисов CORS в приложении вызывается метод **builder.Services.AddCors()**.

Чтобы задействовать CORS для обработки запроса вызывается метод **app.UseCors()**. Для конфигурации параметров CORS этот метод использует делегат, в который передается объект **CorsPolicyBuilder**. И с помощью этого объекта можно выполнить настройку CORS.

|  |  |
| --- | --- |
| 1 | app.UseCors(builder => builder.AllowAnyOrigin()); |

С помощью метода AllowAnyOrigin() мы указываем, что приложение может обрабатывать запросы с любых доменов/адресов.

В качестве ответа клиенту приложение будет отправлять строку "Hello METANIT.COM!".

### Тестирование CORS

Для тестирования создадим второй проект также ASP.NET Core по типу Empty и определим в этом проекте в файле **Program.cs** следующий код:

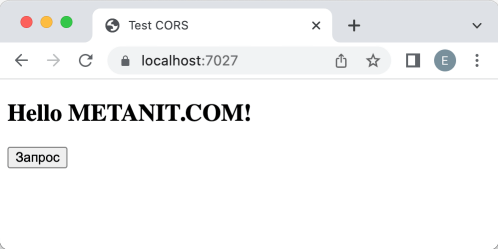
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | var builder = WebApplication.CreateBuilder(args);  var app = builder.Build();    app.UseDefaultFiles();  app.UseStaticFiles();    app.Run(); |

Здесь мы будем работать только со статическими файлами. В частности, определим в этом проекте в папке **wwwroot** html-страницу **index.html**, которая будет загружаться по умолчанию:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | <!DOCTYPE html>  <html>  <head>      <meta charset="utf-8" />      <title>Test CORS</title>  </head>  <body>      <h2 id="result"></h2>      <button id="btn" value="Запрос">Запрос</button>        <script>          const btn = document.getElementById("btn");          const result = document.getElementById("result");          btn.addEventListener("click", async () => {            try {              const response = await fetch("<https://localhost:7199/>");              if (response.ok) result.innerText = await response.text();          }          catch (e) {              result.innerText = e.message;          }          });      </script>  </body>  </html> |

В данном случае по нажатию на кнопку будет выполняться ajax-запрос к первому приложению, который, в моем случае, будет запускаться по адресу "https://localhost:7199/". Полученный от первого приложения ответ будет загружаться в элемент заголовка <h2 id="result">

Запустим оба проекта и во втором приложении нажмем на кнопку, чтобы получить ответ от первого приложения:



## Конфигурация CORS

Для обработки кроссдоменных запросов и работы CORS в прошлой теме код приложения выглядел следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var builder = WebApplication.CreateBuilder();    builder.Services.AddCors(); // добавляем сервисы CORS    var app = builder.Build();    // настраиваем CORS  app.UseCors(builder => builder.AllowAnyOrigin());    app.Map("/", async context => await context.Response.WriteAsync("Hello METANIT.COM!"));    app.Run(); |

В вызове **app.UseCors()** с помощью методов объекта **CorsPolicyBuilder** можно настроить конфигурацию CORS:

* **AllowAnyOrigin()**: принимаются запросы с любого адреса
* **AllowAnyHeader()**: принимаются запросы с любыми заголовками
* **AllowAnyMethod()**: принимаются запросы любого типа (GET/POST)
* **AllowCredentials()**: разрешается принимать идентификационные данные от клиента (например, куки)
* **WithHeaders()**: принимаются только те запросы, которые используют содержат определенные заголовки
* **WithMethods()**: принимаются запросы только определенного типа
* **WithOrigins()**: принимаются запросы только с определенных адресов
* **WithExposedHeaders()**: позволяет серверу отправлять на сторону клиента свои заголовки

### Определение адреса

Метод **AllowAnyOrigin()** позволяет установить взаимодействие с любым приложением по любому адресу. Однако подобное поведение может быть нежелательным. В этом случае мы можем ограничить круг адресов с помощью метода **WithOrigins()**:

|  |  |
| --- | --- |
| 1 | app.UseCors(builder => builder.WithOrigins("[http://example.com](http://example.com/)", "[http://google.com](http://google.com/)")); |

При чем, что важно, в конце названия домена **не должно быть конечного слеша**.

### Определение метода запроса

Метод **AllowAnyMethod()** позволяет принимать запросы любого типа (GET/POST). Также можно настроить принятие только определенного типа запросов:

|  |  |
| --- | --- |
| 1 | app.UseCors(builder => builder.WithOrigins("[https://localhost:7027](https://localhost:7027/)") .WithMethods("GET")); |

### Определение заголовков

Для разрешения запросов с любыми заголовками применяется метод **AllowAnyHeader()**. Следует отметить, что вместе с этим методом лучше также указывать и метод **AllowAnyMethod()** или **WithMethods()** для указания типа запроса:

|  |  |
| --- | --- |
| 1  2  3 | app.UseCors(builder => builder.WithOrigins("[https://localhost:7027](https://localhost:7027/)")                              .AllowAnyHeader()                              .AllowAnyMethod()); |

Если необходимо принимать запросы только с определенными заголовоками, то все требуемые заголовки надо передать в метод **WithHeaders()**:

|  |  |
| --- | --- |
| 1  2  3 | app.UseCors(builder => builder.WithOrigins("[https://localhost:7027](https://localhost:7027/)")                              .AllowAnyMethod()                              .WithHeaders("custom-header")); |

В данном случае необходимо, чтобы клиент отправлял в запросе заголовок "custom-header". Например, отправка данного заголовка в коде javascript с помощью функции fetch:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <h2 id="result"></h2>  <button id="btn" value="Запрос">Запрос</button>    <script>      const btn = document.getElementById("btn");      const result = document.getElementById("result");      btn.addEventListener("click", async () => {          try {              const response = await fetch("<https://localhost:7199/>", { headers: { "custom-header": "test" } });              if (response.ok) result.innerText = await response.text();          }          catch (e) {              result.innerText = e.message;          }      });  </script> |

### Получение заголовков на клиенте

Если сервер отправляет какие-то свои заголовки, то по умолчанию клиент их не получает. Чтобы на стороне сервера указать, какие заголовки может получать клиент, следует использовать метод **WithExposedHeaders()**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | var builder = WebApplication.CreateBuilder();    builder.Services.AddCors(); // добавляем сервисы CORS    var app = builder.Build();    // настраиваем CORS  app.UseCors(builder => builder.WithOrigins("[https://localhost:7027](https://localhost:7027/)")                               .AllowAnyMethod()                               .AllowAnyHeader()                               .WithExposedHeaders("custom-header"));    app.Run(async (context) =>  {      context.Response.Headers.Add("custom-header", "5678");      await context.Response.WriteAsync("Hello World!");  });    app.Run(); |

Сервер устанавливает заголовок custom-header и отправляет его клиенту. Чтобы клиент получил этот заголовок, он передается в метод WithExposedHeaders.

Затем на стороне клиента можно получить значение этого заголовка. Например, получение в коде JavaScript с помощью функции fetch:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | <!DOCTYPE html>  <html>  <head>     <meta charset="utf-8" />     <title>Test CORS</title>  </head>  <body>     <h2 id="result"></h2>     <button id="btn" value="Запрос">Запрос</button>       <script>          const btn = document.getElementById("btn");          const result = document.getElementById("result");          btn.addEventListener("click", async () => {              try {                  const response = await fetch("<https://localhost:7199/>");                  if (response.ok) {                      const headerTitle = "custom-header"; // название заголовка                      result.innerText = await response.text();                      if (response.headers.has(headerTitle)) {  // если заголовок есть                          console.log(response.headers.get(headerTitle));     // получаем его значение                      }                  }              }              catch (e) {                  result.innerText = e.message;              }          });     </script>  </body>  </html> |

Альтернативное получение через XMLHttpRequest:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | const btn = document.getElementById("btn");  const result = document.getElementById("result");  const headers = document.getElementById("headers");  const request = new XMLHttpRequest();    btn.addEventListener("click", function (e) {      request.open("GET", "<https://localhost:44313/>");      request.onreadystatechange = reqReadyStateChange;      request.send();  });    function reqReadyStateChange() {      if (request.readyState == 4) {          if (request.status == 200){              result.innerText = request.responseText;              // получаем заголовок              headers.innerText = request.getResponseHeader("custom-header");          }      }  } |

### Передача идентификационных данных

По умолчанию браузер не посылает никаких идентификационных данных. Подобные данные включают куки, а также данные HTTP-аутентификации. Для отправки идентификационных данных в кроссдоменном запросе на стороне клиента у объекта XMLHttpRequest необходимо установить свойство **withCredentials** равным true.

|  |  |
| --- | --- |
| 1  2  3 | const request = new XMLHttpRequest();  request.open("GET", "<https://localhost:44313/>");  request.withCredentials = true; |

Для получения данных на стороне сервера применяется метод **AllowCredentials()**. Этот метод устанавливает заголовок Access-Control-Allow-Credentials, который говорит браузеру, что сервер разрешает отправку идентификационных данных. При этом данный метод не может использоваться с методом AllowAllOrigin, то есть обязательно нужно указать набор адресов, с которыми будет взаимодействовать сервер. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | var builder = WebApplication.CreateBuilder();    builder.Services.AddCors(); // добавляем сервисы CORS    var app = builder.Build();    // настраиваем CORS  app.UseCors(builder => builder.WithOrigins("[https://localhost:7027](https://localhost:7027/)")                               .AllowCredentials());    app.Run(async (context) =>  {      var login = context.Request.Cookies["login"];  // получаем отправленные куки      await context.Response.WriteAsync($"Hello {login}!");  });    app.Run(); |

При отправке запроса с помощью функции fetch ей необходимо передать опцию **credentials** со значением **include**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <!DOCTYPE html>  <html>  <head>     <meta charset="utf-8" />     <title>Test CORS</title>  </head>  <body>     <h2 id="result"></h2>     <button id="btn" value="Запрос">Запрос</button>       <script>          const btn = document.getElementById("btn");          const result = document.getElementById("result");          document.cookie = "login=tom32"; // куки, которые будут отправляться          btn.addEventListener("click", async () => {              try {                  const response = await fetch("<https://localhost:7199/>", { credentials: "include"});                  if (response.ok)  result.innerText = await response.text();                }              catch (e) {                  result.innerText = e.message;              }          });     </script>  </body>  </html> |

Альтернативный вариант с помощью XMLHttpRequest:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | const btn = document.getElementById("btn");    const request = new XMLHttpRequest();  document.cookie = "login=tom32;";   // куки, которые будут отправляться    btn.addEventListener("click", function () {      request.open("GET", "<https://localhost:44313/>");      request.onreadystatechange = reqReadyStateChange;      request.withCredentials = true; //  устанавливаем отправку      request.send();  });  function reqReadyStateChange() {      if (request.readyState == 4) {          if (request.status == 200)              console.log(request.responseText);      }  } |

## Политики CORS

Для упрощения комплексной конфигации CORS ASP.NET Core позволяет определять **политики CORS**. Политики представляют набор правил взаимодействия сервера и клиента. А чтобы задействовать нужную политику, в метод app.UseCors передается ее название:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | var builder = WebApplication.CreateBuilder();    builder.Services.AddCors(options => options.AddPolicy("AllowLocalhost7027",  builder => builder                      .WithOrigins("[https://localhost:7027](https://localhost:7027/)")                      .AllowAnyHeader()                      .AllowAnyMethod())                 );    var app = builder.Build();    app.UseCors("AllowLocalhost7027");    app.Run(async context => await context.Response.WriteAsync("Hello client!"));    app.Run(); |

В метод AddCors передается делегат, который принимает объект **CorsOptions** - настройки CORS. У этого объекта вызывается метод **AddPolicy()**, который устанавливает политику. Первый параметр метода - произвольное название политики (в данном случае AllowLocalhost7027). Второй параметр - делегат, который для создания политики использует объект **CorsPolicyBuilder**. То есть это тот же самый объект, методы которого были рассмотрены в прошлой теме. Поэтому для него можно использовать все те же самые методы:

|  |  |
| --- | --- |
| 1  2  3  4  5 | builder.Services.AddCors(options => options.AddPolicy("AllowLocalhost7027", builder => builder                      .WithOrigins("[https://localhost:7027](https://localhost:7027/)")                      .AllowAnyHeader()                      .AllowAnyMethod())  ); |

Чтобы применить эту политику в метод app.UseCors передается ее название:

|  |  |
| --- | --- |
| 1 | app.UseCors("AllowLocalhost7027"); |

При этом можно определить набор политик и применять одну из них, меняя ее по мере необходимости:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | builder.Services..AddCors(options =>  {      options.AddPolicy("TestPolicy", builder => builder                         .WithOrigins("[https://localhost:7027](https://localhost:7027/)")                         .AllowAnyHeader()                         .AllowAnyMethod());      options.AddPolicy("ProductionPolicy", builder => builder                         .WithOrigins("[http://localhost.com](http://localhost.com/)")                         .AllowAnyHeader()                         .AllowAnyMethod());  }); |

## Глобальная и локальная настройка CORS

Если для обработки запросов в приложении применяются конечные точки, то мы можем для каждой конечной точки указать свои настройки CORS. Таким образом, мы можем установить глобально для всех ресурсов одни и те же настройки CORS, либо конкретизировать для каждого ресурса свои настройки.

В прошлых темах использовалась глобальная настройка CORS, которая предполагает, что настройки CORS передаются в метод app.UseCors():

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | var builder = WebApplication.CreateBuilder();    builder.Services.AddCors(); // добавляем сервисы CORS    var app = builder.Build();    // глобальная настройка CORS для всех ресурсов  app.UseCors(builder => builder.AllowAnyOrigin());    app.MapGet("/", async context =>  await context.Response.WriteAsync("Hello World!"));  app.MapGet("/home", async context =>  await context.Response.WriteAsync("Home Page!"));    app.Run(); |

Здесь вне зависимости, обращаемся ли мы к ресурсу "/" или к ресурсу "/home", к обоим ресурсам могут обращаться пользователи с любых адресов.

С помощью метода **RequireCors()** у объекта IEndpointConventionBuuilder мы можем установить настройки CORS для каждого конкретного маршрута. Данный метод в качестве параметра принимает либо название политики CORS, либо делегат с параметром CorsPolicyBuilder, с помощью который устанавливается конфигурацию CORS.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | var builder = WebApplication.CreateBuilder();    builder.Services.AddCors(options => {        options.AddPolicy("AllowTestSite", builder => builder          .WithOrigins("[https://localhost:7027](https://localhost:7027/)")          .AllowAnyHeader()          .AllowAnyMethod());        options.AddPolicy("AllowNormSite", builder => builder              .WithOrigins("[https://localhost.com](https://localhost.com/)")              .AllowAnyHeader()              .AllowAnyMethod());  });    var app = builder.Build();    app.UseCors();    app.MapGet("/", async context => await context.Response.WriteAsync("Hello World!"))      .RequireCors(options => options.AllowAnyOrigin());    app.MapGet("/home", async context => await context.Response.WriteAsync("Home Page!"))      .RequireCors("AllowNormSite");    app.MapGet("/about", async context => await context.Response.WriteAsync("About Page!"))      .RequireCors("AllowTestSite");    app.Run(); |

Таким образом, мы можем определить для каждого ресурса в приложении свою конфигурацию CORS.