

В этом руководстве описано, как вызвать веб-API ASP.NET Core с помощью JavaScript и [Fetch API](#).

Предварительные требования

- Изучите [Учебник. Создание веб-API](#).
- Опыт работы с CSS, HTML и JavaScript.

Вызов веб-API с помощью JavaScript

В этом разделе описано, как добавить HTML-страницу, содержащую формы для создания и администрирования элементов списка задач. Обработчики событий присоединяются к элементам на странице. При использовании обработчиков событий создаются запросы HTTP к методам действия веб-API. Функция `Fetch API` `fetch` инициирует каждый такой запрос HTTP.

Функция `fetch` возвращает объект [Promise](#), который содержит ответ HTTP, представленный в виде объекта `Response`. Распространенным подходом является извлечение текста ответа JSON путем вызова функции `json` в объекте `Response`. JavaScript изменяет страницу, используя сведения из ответа API.

Самый простой вызов `fetch` принимает один параметр, представляющий маршрут. Второй параметр (объект `init`) является необязательным. `init` используется для настройки запроса HTTP.

1. Настройте в приложении [обслуживание статических файлов](#) и [включите сопоставление файлов по умолчанию](#). Вставьте в метод `Configure` в файле `Startup.cs` следующий выделенный код:

C#	 Копировать
<pre>public void Configure(IApplicationBuilder app, IWebHostEnvironment env) { if (env.IsDevelopment()) { app.UseDeveloperExceptionPage(); } app.UseDefaultFiles(); app.UseStaticFiles(); app.UseHttpsRedirection();</pre>	

```

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});
}

```

2. Создайте папку *wwwroot* в корневом каталоге проекта.
3. Создайте папку *js* в папке *wwwroot*.
4. Добавьте HTML-файл *index.html* в папку *wwwroot*. Замените содержимое файла *index.html* следующей разметкой:

HTML

 Копировать

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>To-do CRUD</title>
    <link rel="stylesheet" href="css/site.css" />
</head>
<body>
    <h1>To-do CRUD</h1>
    <h3>Add</h3>
    <form action="javascript:void(0);" method="POST"
onsubmit="addItem()">
        <input type="text" id="add-name" placeholder="New to-do">
        <input type="submit" value="Add">

    </form>

    <div id="editForm">
        <h3>Edit</h3>
        <form action="javascript:void(0);" onsubmit="updateItem()">
            <input type="hidden" id="edit-id">
            <input type="checkbox" id="edit-isComplete">
            <input type="text" id="edit-name">
            <input type="submit" value="Save">
            <a onclick="closeInput()" aria-label="Close">&#10006;</a>
        </form>
    </div>

    <p id="counter"></p>

    <table>
        <tr>
            <th>Is Complete?</th>
            <th>Name</th>

```

```

        <th></th>
        <th></th>
    </tr>
    <tbody id="todos"></tbody>
</table>

<script src="js/site.js" asp-append-version="true"></script>
<script type="text/javascript">
    getItem();
</script>
</body>
</html>

```

5. Добавьте файл JavaScript с именем *site.js* в папку *wwwroot/js*. Замените содержимое файла *site.js* следующим кодом:

JavaScript	 Копировать
<pre> const uri = 'api/ToDoItems'; let todos = []; function getItem() { fetch(uri) .then(response => response.json()) .then(data => _displayItems(data)) .catch(error => console.error('Unable to get items.', error)); } function addItem() { const addNameTextbox = document.getElementById('add-name'); const item = { isComplete: false, name: addNameTextbox.value.trim() }; fetch(uri, { method: 'POST', headers: { 'Accept': 'application/json', 'Content-Type': 'application/json' }, body: JSON.stringify(item) }) .then(response => response.json()) .then(() => { getItem(); addNameTextbox.value = ''; }) .catch(error => console.error('Unable to add item.', error)); } function deleteItem(id) { </pre>	

```

    fetch(`${uri}/${id}`, {
      method: 'DELETE'
    })
    .then(() => getItems())
    .catch(error => console.error('Unable to delete item.', error));
  }

function displayEditForm(id) {
  const item = todos.find(item => item.id === id);

  document.getElementById('edit-name').value = item.name;
  document.getElementById('edit-id').value = item.id;
  document.getElementById('edit-isComplete').checked = item.isComplete;
  document.getElementById('editForm').style.display = 'block';
}

function updateItem() {
  const itemId = document.getElementById('edit-id').value;
  const item = {
    id: parseInt(itemId, 10),
    isComplete: document.getElementById('edit-isComplete').checked,
    name: document.getElementById('edit-name').value.trim()
  };

  fetch(`${uri}/${itemId}`, {
    method: 'PUT',
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(item)
  })
  .then(() => getItems())
  .catch(error => console.error('Unable to update item.', error));

  closeInput();

  return false;
}

function closeInput() {
  document.getElementById('editForm').style.display = 'none';
}

function _displayCount(itemCount) {
  const name = (itemCount === 1) ? 'to-do' : 'to-dos';

  document.getElementById('counter').innerText = `${itemCount} ${name}`;
}

function _displayItems(data) {
  const tBody = document.getElementById('todos');
  tBody.innerHTML = '';
}

```

```

_displayCount(data.length);

const button = document.createElement('button');

data.forEach(item => {
  let isCompleteCheckbox = document.createElement('input');
  isCompleteCheckbox.type = 'checkbox';
  isCompleteCheckbox.disabled = true;
  isCompleteCheckbox.checked = item.isComplete;

  let editButton = button.cloneNode(false);
  editButton.innerText = 'Edit';
  editButton.setAttribute('onclick', `displayEditForm(${item.id})`);

  let deleteButton = button.cloneNode(false);
  deleteButton.innerText = 'Delete';
  deleteButton.setAttribute('onclick', `deleteItem(${item.id})`);

  let tr = tbody.insertRow();

  let td1 = tr.insertCell(0);
  td1.appendChild(isCompleteCheckbox);

  let td2 = tr.insertCell(1);
  let textNode = document.createTextNode(item.name);
  td2.appendChild(textNode);

  let td3 = tr.insertCell(2);
  td3.appendChild(editButton);

  let td4 = tr.insertCell(3);
  td4.appendChild(deleteButton);
});

todos = data;
}

```

Может потребоваться изменение параметров запуска проекта ASP.NET Core для локального тестирования HTML-страницы:

1. Откройте файл *Properties\launchSettings.json*.
2. Удалите свойство `launchUrl`, чтобы приложение открылось через *index.html* — файл проекта по умолчанию.

В этом примере вызываются все методы CRUD в веб-API. Ниже приводится пояснение запросов веб-API.

Получение списка элементов задач

В следующем коде HTTP-запрос GET направляется по пути *api/TodoItems*:

```
fetch(uri)
  .then(response => response.json())
  .then(data => _displayItems(data))
  .catch(error => console.error('Unable to get items.', error));
```

Когда веб-API возвращает код состояния, указывающий на успешное выполнение, вызывается функция `_displayItems`. Каждый элемент списка задач в параметре массива, который принимается `_displayItems`, добавляется в таблицу с помощью кнопок **Изменить** и **Удалить**. Если запрос веб-API завершается сбоем, в консоли браузера регистрируется сообщение об ошибке.

Добавление элемента задачи

В приведенном ниже коде выполняется следующее:

- Переменная `item` объявляется для создания представления объектного литерала элемента списка задач.
- Для запроса Fetch настраиваются следующие параметры:
 - `method` определяет команду действия HTTP POST.
 - `body` определяет представление JSON текста запроса. JSON создается путем передачи литерала объекта, хранящегося в `item`, в функцию `JSON.stringify`.
 - `headers` определяет заголовки `Accept` и `Content-Type` запросов HTTP. Для обоих параметров устанавливается значение `application/json`, чтобы классифицировать тип носителя при получении и отправке соответственно.
- HTTP-запрос POST направляется по пути `api/TodoItems`.

```
function addItem() {
  const addNameTextbox = document.getElementById('add-name');

  const item = {
    isComplete: false,
    name: addNameTextbox.value.trim()
  };

  fetch(uri, {
    method: 'POST',
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    },
    // ...
  });
}
```

```
    body: JSON.stringify(item)
  })
  .then(response => response.json())
  .then(() => {
    getItems();
    addNameTextbox.value = '';
  })
  .catch(error => console.error('Unable to add item.', error));
}
```


Когда веб-API возвращает код состояния, указывающий на успешное выполнение, вызывается функция `getItems` для обновления таблицы HTML. Если запрос веб-API завершается сбоем, в консоли браузера регистрируется сообщение об ошибке.

Обновление элемента задачи

Обновление элемента списка задач аналогично его добавлению. Но есть два существенных отличия:

- Путь имеет суффикс с уникальным идентификатором обновляемого элемента. Например, `api/TodoItems/1`.
- Команда действия HTTP — это PUT, как указано в параметре `method`.

JavaScript


 Копировать

```
fetch(`${uri}/${itemId}`, {
  method: 'PUT',
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(item)
})
.then(() => getItems())
.catch(error => console.error('Unable to update item.', error));
```

Удаление элемента задачи

Чтобы удалить элемент списка задач, укажите для параметра запроса `method` значение `DELETE` и определите уникальный идентификатор элемента в URL-адресе.

JavaScript

 Копировать

```
fetch(`${uri}/${id}`, {
  method: 'DELETE'
})
.then(() => getItems())
```

```
.catch(error => console.error('Unable to delete item.', error));
```

Перейдите к следующему руководству, в котором описано, как создавать страницы справки по веб-API:

Начало работы с Swashbuckle и ASP.NET Core

Были ли сведения на этой странице полезными?

 Да  Нет