БИБЛИОТЕКА AXIOS. ПРОТОКОЛ WEBSOCKET

JavaScript

Модуль 5. Урок 3.

LUXOFT

LXFT LISTED NYSE

AXIOS ИЛИ FETCH?

Fetch — нативный низкоуровневый JavaScript интерфейс для выполнения HTTP-запросов с использованием обещаний через глобальную функцию fetch().

Axios — JavaScript-библиотека, основанная на промисах, для выполнения HTTP-запросов.

В современных браузерах fetch() работает из коробки, для старых браузеров написаны полифилы, а библиотека - это всегда дополнительные килобайты нагрузки.

GET-ЗАПРОС

Типичная задача во frontend'е — получение JSON данных с сервера.

Axios для этого требуется 1 действие,

Fetch - 2 действия: запрос + вызов метода .json() над результатом запроса.

```
fetch('http://localhost:3333/employees/search/findByName?name=Петр')
    .then(response => response.json())
    .then(json => console.log(json))

axios.get('http://localhost:3333/employees/search/findByName?name=Петр')
    .then(response => console.log(response));
```

ОБРАБОТКА ОШИБОК

Axios обрабатывает ошибки логично. Если сервер вернул ответ с HTTP статусом ошибки (например 404 или 500), то обещание будет отвергнуто.

```
axios.get(url)
   .then(result => console.log('success:', result))
   .catch(error => console.log('error:', error));
```

Fetch отвергнет обещание только если произошла сетевая ошибка (DNS не разрешил адрес, сервер недоступен, CORS не разрешен). В отстальных случаях ошибки не будет (включая HTTP статусы 404 или 500), поэтому для получения более интуитивного поведения такие ситуации придётся обрабатывать вручную.

```
fetch(url).then(response => {
      return response.json().then(data => {
         if (response.ok) return data;
         else return Promise.reject({status: response.status, data});
      });
   })
   .then(result => console.log('success:', result))
   .catch(error => console.log('error:', error));
```

POST-ЗАПРОСЫ

С axios всё просто, а с fetch уже не так: JSON обязан быть преобразован в строку, а заголовок Content-Type должен указывать, что отправляются JSON данные, иначе сервер будет рассматривать их как строку.

```
axios.post('/user', {
  firstName: 'Ivan',
  lastName: 'Ivanov'
});
```

```
fetch('/user', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    firstName: 'Ivan',
    lastName: 'Ivanov'
  })
});
```

СИНТАКСИС ASYNC/AWAIT

```
// Хотите использовать async/await?
// Добавьте слово `async` внешней функции
  async function getUser() {
    try {
      const response = await axios.get('/user?ID=12345');
      console.log(response);
    } catch (error) {
      console.error(error);
    }
  }
```

ФУНКЦИЯ AXIOS

```
// GET request for remote image
  axios({
     method:'get',
     url:'http://bit.ly/2mTM3nY'
  })
     .then(function (response) {
        console.log(response);
     });

// Send a GET request (default method)
  axios('/user/12345');
```

```
// Send a POST request
  axios({
     method: 'post',
     url: '/user/12345',
     data: {
        firstName: 'Fred',
        lastName: 'Flintstone'
     }
  });
```

МЕТОДЫ С НАЗВАНИЯМИ HTTP-МЕТОДА

axios.get(url[, config])

axios.delete(url[, config])

axios.head(url[, config])

axios.options(url[, config])

axios.post(url[, data[, config]])

axios.put(url[, data[, config]])

axios.patch(url[, data[, config]])

КАСТОМНАЯ КОНФИГУРАЦИЯ ЭКЗЕМПЛЯРА AXIOS

```
const server = axios.create({
    baseURL: 'https://some-domain.com/api/',
    timeout: 1000,
    headers: {'X-Custom-Header': 'foobar'}
});

server.get(`/search/findByName?name=Петр`)
    .then(res => console.log(res));
```

ГЛОБАЛЬНАЯ КОНФИГУРАЦИЯ AXIOS

```
axios.defaults.baseURL = 'https://api.example.com';
axios.defaults.headers.common['Authorization'] = AUTH_TOKEN;
axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';
```

ОБРАБОТКА ОШИБОК В AXIOS

```
axios.get('/user/12345')
  .catch(function (error) {
    if (error.response) {
      // Запрос был выполнен, и сервер ответил кодом ошибки – кодом за диапазоном 2xx
      console.log(error.response.data); // Детали ответа сервера – сервер может передать
          // дополнительную информацию о причинах ошибки
      console.log(error.response.status); // Статус ответа сервера
      console.log(error.response.headers); // Заголовки ответа сервера
    } else if (error.request) {
      // Запрос был отправлен, но ответа не было получено
      console.log(error.request);
    } else {
      // Что-то случилось в процессе создания запроса, запрос не был отправлен
      console.log('Error', error.message);
    }
    console.log(error.config);
  });
```

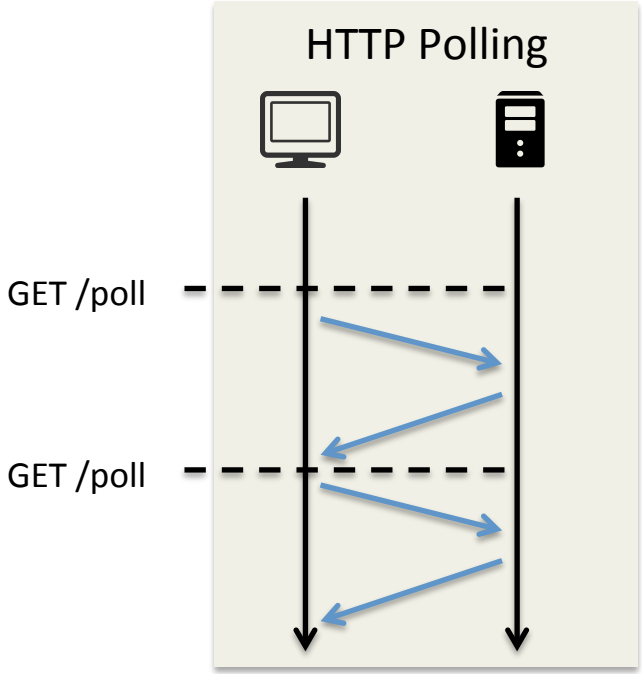
ОТМЕНА ЗАПРОСА

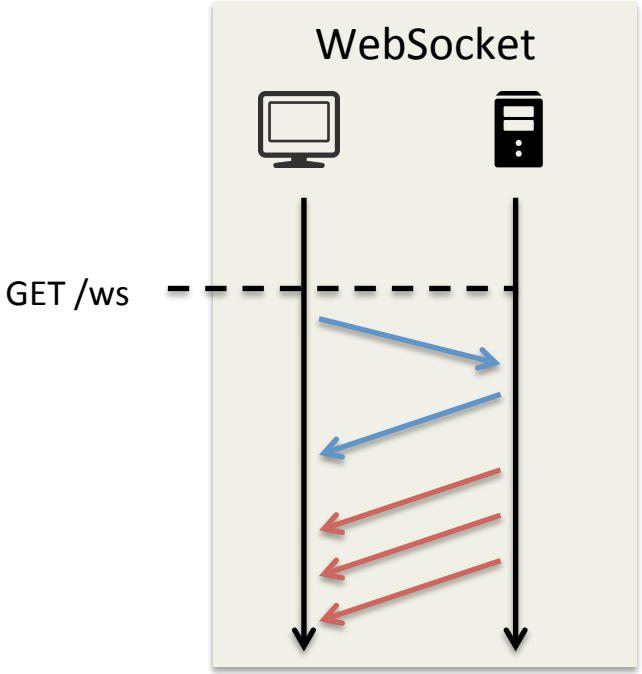
Вы можете отменить уже начатый запрос с помощью токена отмены − CancelToken:

```
const CancelToken = axios.CancelToken;
const source = CancelToken.source();

axios.get('/user/12345', {
   cancelToken: source.token
}).catch(function (thrown) {
   if (axios.isCancel(thrown)) {
      console.log('Request canceled', thrown.message);
   } else {
      // обработка ошибки
   }
});
axios.post('/user/12345', {  name: 'new name'  }, {  cancelToken: source.token })

// отмена запроса (сообщение − опциональный параметр)
source.cancel('Operation canceled by the user.');
```

ПРОТОКОЛ WEBSOCKET

WebSocket — протокол связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.





HTTP протокол

WebSocket протокол

главное отличие WebSocket от HTTP:
- в HTTP запрос инициирует клиент
- в WebSocket сам сервер может отправить что-то по своей инициативе

WEBSOCKET В БРАУЗЕРЕ

```
<script>
   var webSocket = new WebSocket('ws://localhost/echo');

   webSocket.onopen = function(event) {
      console.log('WebSocket connection is open');
      webSocket.send("Hello Web Socket!");
   };

   webSocket.onmessage = function(event) {
      console.log('Message received:  ' + event.data);
   };

   webSocket.onclose = function(event) {
      console.log('WebSocket connection is closed');
   };
</script>
```

КОГДА ИСПОЛЬЗОВАТЬ ПРОТОКОЛ WEBSOCKET?

- протокол WebSocket не является заменой HTTP
- WebSocket стоит использовать, если сервер должен отправлять обновления клиенту **в реальном времени**, а не по запросу
- примеры: чаты, котировки акций, "живая" статистика, интерактивные игры
- альтернативой Websocket является **polling** − частый опрос сервера используя HTTP, но частый polling будет перегружать сервер, а редкий − приводить к задержке в отображении обновлений
- еще одна альтернатива − long polling: отправляется HTTP-запрос на сервер, и соединение остается открытым до тех пор, пока не придет ответ: