

Asynchroniczne interakcje z serwerem

Projektowanie i programowanie systemów internetowych I

mgr inż. Krzysztof Rewak

13 maja 2018

Wydział Nauk Technicznych i Ekonomicznych

Państwowa Wyższa Szkoła Zawodowa im. Witelona w Legnicy

Plan prezentacji

1. Komunikacja frontend-backend
2. AJAX
3. Przykłady
4. Podsumowanie

Komunikacja frontend-backend

Przyptyw klasyczny

Komunikacja klienta z serwerem (lub frontendu z backendem) odbywa się poprzez protokół HTTP.

(wiadomo to przynajmniej od wykładu czwartego zatytułowanego *Obsługa zapytań, protokół HTTP*)

Przyptyw klasyczny

Oznacza to, że na każde zapytanie klienta zostanie zwrócona pewna odpowiedź z serwera.

Jak pobrać dane?

Jak pobrać dane?

- odświeżyć stronę ręcznie (F5)
- kliknąć w ``
- przesłać formularz
- kliknąć element, który wywoła `window.location.href = url`

Jak pobrać dane?

Każdy z powyższych sposobów odpytuje serwer w sposób synchroniczny.

Oznacza to, że użytkownik musi czekać na odpowiedź serwera i najczęściej zostanie przekierowany na inną stronę.

A tutaj?

Arkusz obecności i ocen

[illegible]

Pod maską

Narzędzia dla programistów – Krzysztof Rewak :: Wydział Nauk Technicznych i Ekonomicznych :: Państwowa Wyższa Szkoła Zawodowa im. Witelona w Legnicy – <http://pwsz.rewak.pl/oceny>

Inspektor Konsola Debugger Edytor stylów Wydajność Pamięć Sieć Dane Adblock Plus

Wszystkie HTML CSS JS XHR Czcionki Obrazy Media WebSocket Inne ☐ Trwałe dzienniki ☐ Wyłącz pamięć podręczną

Status	Metoda	Plik	Domena	Przyczyna	Typ
● 200	GET	collect?v=1&_v=j67&a=1239738094&t=pageview&s=3&dl=http://...	www.google-analytics.com	img	gif
● 200	GET	semesters	pwsz.rewak.pl	xhr	json
● 200	GET	courses?courseId=&groupId=&semesterId=2&studentId=	pwsz.rewak.pl	xhr	json
● 200	GET	groups?courseId=4&groupId=&semesterId=2&studentId=	pwsz.rewak.pl	xhr	json
● 200	GET	grades?courseId=4&groupId=9&semesterId=2&studentId=12345	pwsz.rewak.pl	xhr	json
● 200	GET	grades?courseId=4&groupId=9&semesterId=2&studentId=23456	pwsz.rewak.pl	xhr	json
● 200	GET	grades?courseId=4&groupId=9&semesterId=2&studentId=34567	pwsz.rewak.pl	xhr	json

Przypływ klasyczny

Okazuje się, że czasami zapytanie do serwera nie wymusza przeładowania bieżącej strony. Wówczas najczęściej mówimy o technologii AJAX.

AJAX

AJAX (ang. *Asynchronous JavaScript and XML*), czyli asynchroniczny JavaScript i XML, to technika wykorzystywana przy asynchronicznym odpytywaniu serwera od strony klienta.

Idea jest bardzo podobna jak przy odpytywaniu klasycznym:

- klient wysyła zapytanie na serwer
- zapytanie jest przetwarzane
- klient odbiera odpowiedź serwera











Różnica polega na tym, że zapytanie nie jest przeładowaniem URL strony, a przesyłane jest *w tle*.

Użytkownik pozostaje zatem w tym samym miejscu, w którym był oraz może wchodzić w dalsze interakcje z serwisem.

Wykorzystanie technologii AJAX ma wiele zalet.

Wszystkie zostaną przedstawione na przykładach znanego studentom internetowego systemu dla prowadzących zajęcia.

Oto lista kursów w CRUD-owej tabelce:

Przeszukaj tabelę...							
#	nazwa	kier./spec.	semestr	forma	aktywny?		+
PPO2	Projektowanie i programowanie obiektowe II	INF	IV	laboratorium	✓	 	
PPSI1	Projektowanie i programowanie systemów internetowych	INF/PAM	IV	wykład	✓	 	
PPSI1	Projektowanie i programowanie systemów internetowych	INF/PAM	IV	projekt	✓	 	
PPO1	Projektowanie i programowanie obiektowe I	INF	III	laboratorium	✗	 	
PZ	Projekt zespołowy	INF/SSK	VII	projekt	✗	 	

(CRUD to skrótowiec od *create, read, update and delete*)

Jeżeli okna edycji i potwierdzenie usunięcia otwierałyby się w oknach modalnych, wówczas prymitywne rozwiązanie polegałoby na wygenerowaniu $n + n$ różnych zestawów danych dla każdego z n rekordów.

Można też podłączyć zdarzenie na kliknięcie przycisku *edytuj* do metody, która wywoła asynchronicznie pobranie informacji na temat edycji. Wówczas ograniczymy kod do stworzenia dwóch okien modalnych, które będą w *locie* wypełniane tylko dla wybranego przypadku.

Zaleta numer jeden:

Można zmniejszyć rozmiar i redundancję (nawet generowanego) kodu.

Zaleta *due*

Oto arkusz ocen i obecności dla wybranej grupy:

⌕	L1	L2	L3	L4	Z1	L5	L6	L7	L8	Z2	L9	L10	L11	Z3	L12	L13	L14	P1	Ω	+
					3.5					4				2				5	4.0	
					3.5					2 > 4				2				5	3.5	
		+			3.5 >	+		+	+	5	+	+		4.5				5	5.0	
				+	4	+		+	+	4	+	+	+	5				5	5.0	
		+			4.5					4				5				5	4.5	
					5			+		5			+	5				5	5.0	
					2 > 3					4				2				5	3.5	
					4					3.5 >	+	+		4				5	4.5	
					3.5					2				2				5	3.0	
		+	+	+	4.5	+	+	+	+	4		+	+	2				5	4.5	
					2					2				2				2	2.0	
					3.5					2 > 3				2				5	3.5	

W tradycyjnym przepływie zapytań istniałyby dwie opcje: wysłanie całego arkusza naraz lub wysyłanie każdej oceny i obecności osobno. Pierwsze rozwiązanie byłoby bardzo podatne na pomyłki, a drugie - przeraźliwie powolne.

Można też podłączyć zdarzenie na dwuklik lub wciśnięcie klawisza *enter* w wybranej komórce oceny do metody, która wywoła asynchronicznie wysłanie informacji do backendu. Wówczas oceny będą przesyłane pojedynczo, ale bez zbędnego oczekiwania na przeładowanie strony.

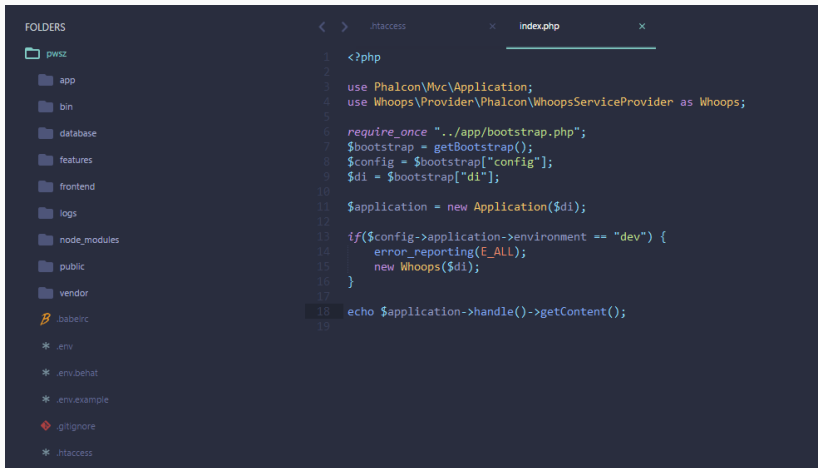
Zaleta *due*

Zaleta numer dwa:

Można zmniejszyć możliwość popełnienia błędu oraz zwiększyć szybkość korzystania z systemu internetowego. Inaczej: ulepszamy UX, *user experience*.

Zaleta *tre*

Oto struktura folderów projektu:



Tworząc klasyczną aplikację internetową, przynajmniej jeden programista musi znać zarówno technologie frontendowe i backendowe. Ponadto kontrolery najczęściej zwracają całe widoki, więc frontend jest uzależniony od backendu.

Korzystając z asynchronicznych zapytań można całkowicie oddzielić frontend od backendu. W przedstawionym przykładzie widać to na podziale na foldery `app` i `frontend`, ale mogą to być całkowicie osobne repozytoria, projekty i serwery.

Zaleta numer trzy:

Można odseparować warstwy aplikacji i przekazywać jedynie istotne dane.

A wady?

Korzystanie z asynchronicznych zapytań oczywiście ma swoje wady:

- klient musi mieć włączoną obsługę JavaScriptu (ale kto jej teraz nie ma?),
- robotom z wyszukiwarek ciężiej przeskanować taką witrynę (ale to można obejść tworząc mapę strony),
- zaburzony zostaje klasyczny przepływ ruchu na stronie internetowej, więc niektórzy użytkownicy mogą poczuć się zdezorientowani (co można oprogramować dodatkowo, *vide* router SPA na mojej stronie).

Przykłady

Vanilla JavaScript

Korzystając z czystego JavaScriptu można napisać następująco:

```
var xhr = new XMLHttpRequest();

xhr.open("GET", "/api/courses");

xhr.onreadystatechange = function() {
    console.log(xhr.responseText);
}

xhr.send(null);
```

Analogiczne zapytanie z wykorzystaniem popularnej biblioteki jQuery może wyglądać następująco:

```
$.ajax({  
  type: "GET",  
  url: "api/courses",  
  success: function(data) {  
    console.log(data);  
  },  
  error: function() {  
    console.log("Error");  
  }  
});
```

Ponadto istnieje funkcja `fetch` korzystająca z javascriptowych obietnic:

```
fetch("api/courses").then(function(response) {  
    return response.text();  
}).then(function(data) {  
    console.log(data);  
}).catch(function(error) {  
    console.log("Error: " + error);  
});
```

Nowoczesne frameworki

Pobieranie danych asynchronicznie jest podstawą działania współczesnych frameworków frontendowych.

Angular, React czy Vue są popularnymi rozwiązaniami, które często i coraz częściej są wykorzystywane przy budowaniu nowoczesnych systemów internetowych.

Oto przykład strony z FAQ jako komponent Vue:

```
<template>
  <div id="faq">
    <h1>FAQ</h1>

    <div class="question" v-for="question in questions">
      <div class="ui divider"></div>
      <h3 class="ui header">
        <i class="question circle outline icon"></i>
        {{ question.question }}
      </h3>
      <blockquote v-html="question.answer"></blockquote>
    </div>
  </div>
</template>
```



```
<style scoped>
  blockquote { padding: .5em; }
  .question:first-of-type > .ui.divider { display: none; }
  .question { padding: .5em 0; }
</style>
```

FAQ.vue

```
<script type="text/javascript">
  export default {
    data() {
      return {
        questions: [],
      }
    },
    created() {
      this.fetchInitialData()
    },
    methods: {
      fetchInitialData() {
        this.$http.get("faq").then(function(response) {
          this.questions = response.body.data
        })
      },
    },
  }
</script>
```

Filtrowanie adresów		FII	
Nagłówki	Ciasteczka	Parametry	Odpowiedź
Filtruj własności			
▼ JSON			
success: true			
message:			
▼ data: [...]			
▼ 0: {...}			
id: 1			
question: Napisałem(am) do Pana mejla i nie otrzymałem(am) jeszcze odpowiedzi. Co robić?			
answer: Uczelnianą skrzynkę mejlową mam podłączoną do prywatnego telefonu i komputera, więc staram się czytać wszystkie mejle na bieżąco. Odpisuję natomiast w wolnym od obowiązków życia zawodowego i prywatnego czasie. Proszę o uszanowanie mojej prywatności i nie podejmowanie prób kontaktowania się ze mną innymi kanałami niż uczelniany adres. Wszelkie wiadomości związane z uczelnią, a kierowane na adres prywatny, telefon, komunikator czy platformy społecznościowe, usuwam bez zastanawiania się czego dotyczą.			
▼ 1: {...}			
id: 2			
question: Napisałem(am) do Pana mejla i nie otrzymałem(am) nigdy odpowiedzi. Co robić?			
answer: Sugeruję sprawdzić jeszcze raz z jakiego adresu zostały wysłane poprzednie wiadomości. W ramach walki ze spamem - a mój służbowy adres niestety jest upubliczniony w internecie - moje filtry przepuszczają wiadomości wysyłane jedynie z domen @student.pwss.lgnica.edu.pl			
▼ 2: {...}			
id: 3			
question: Nie było mnie na zajęciach? Co robić, jak żyć?			
answer: W sytuacji idealnej najlepszą opcją jest to, że student informuje mnie o swojej nieobecności przed zajęciami. Wówczas ustalamy termin tzw. "odróbkki". W innych przypadkach będziemy sprawy rozwiązywali indywidualnie - w zależności od rodzaju zajęć, liczby grup i innych czynników.			
▼ Zawartość odpowiedzi			
1	{ "success": true, "message": "", "data": [{ "id": "1", "question": "Napisa\u0142em(am) do Pana		

FAQ (często zadawane pytania i odpowiedzi na nie)

❓ Napisalem(am) do Pana mejla i nie otrzymałem(am) jeszcze odpowiedzi. Co robić?

Uczelnianą skrzynkę mejlową mam podłączoną do prywatnego telefonu i komputera, więc staram się czytać wszystkie mejle na bieżąco. Odpisuję natomiast w wolnym od obowiązków życia zawodowego i prywatnego czasie. Proszę o uszanowanie mojej prywatności i nie podejmowanie prób kontaktowania się ze mną innymi kanałami niż uczelniany adres. Wszelkie wiadomości związane z uczelnią, a kierowane na adres prywatny, telefon, komunikator czy platformy społecznościowe, usuwam bez zastanawiania się czego dotyczą.

❓ Napisalem(am) do Pana mejla i nie otrzymałem(am) nigdy odpowiedzi. Co robić?

Sugeruję sprawdzić jeszcze raz z jakiego adresu zostały wysłane poprzednie wiadomości. W ramach walki ze spamem - a mój służbowy adres niestety jest upubliczniony w internecie - moje filtry przepuszczają wiadomości wysyłane jedynie z domenu @student.pwsz.legnica.edu.pl

❓ Nie było mnie na zajęciach? Co robić, jak żyć?

W sytuacji idealnej najlepszą opcją jest to, że student informuje mnie o swojej nieobecności przed zajęciami. Wówczas ustalamy termin tzw. "odróbki". W innych przypadkach będziemy sprawy rozwiązywać indywidualnie - w zależności od rodzaju zajęć, liczby grup i innych czynników.

Podsumowanie

Bibliografia i ciekawe źródła



https:

`//www.sitepoint.com/guide-vanilla-ajax-without-jquery/`



`http://api.jquery.com/jquery.ajax/`



`https://vuejs.org/v2/guide/index.html`

Pytania?

Kod prezentacji dostępny jest w repozytorium git pod adresem
<https://bitbucket.org/krewak/pwsz-ppsi>



Wszystkie informacje dot. kursu dostępne są pod adresem
<http://pwsz.rewak.pl/kursy/4>

