MODUŁ 6:
Aplikacje WWW oraz wprowadzenie do Node.js

Cel modułu

Moduł stanowi intensywne wprowadzenie w problematykę programowania aplikacji WWW po stronie serwera; innymi słowy – w zagadnienia programowania dynamicznych dokumentów WWW. Efektywne rozpoczęcie pracy wymaga opanowania dwóch obszarów omówionych w tym wprowadzeniu:

- Pojęcia, wzorce oraz ograniczenia związane z programowaniem funkcjonalności aplikacji po stronie serwera WWW;
- Wybrane środowisko programowania dostępne po stronie serwera – w naszym przypadku będzie to JavaScript i Node.js.

Poznamy przy tej okazji kilka istotnych zagadnień. Przede wszystkim, ponownie mamy tu do czynienia z tworzeniem niestatycznej strony WWW, tym razem strony dynamicznej, opartej na oprogramowaniu działającym po stronie serwera. Dotychczas, w module 3, obserwowaliśmy tylko działanie takich stron z zewnątrz, analizując zawartość żądań protokołowych związanych z pobieraniem takich stron. Po drugie, zapoznamy się z pojęciem wzorców projektowych i sposobem ich wspierania przez ramę programistyczną. Zapoznamy się też z podstawami programowania w Node.js, który będzie dla nas stanowił fundament dla oprogramowania warstwy serwerowej.

Efekty kształcenia

Niniejsza lekcja służy osiągnięciu następujących efektów kształcenia.

Wiedza

Po ukończonym kursie student / studentka:

- Wyjaśnia pojęcie ramy programistycznej w kontekście budowy aplikacji WWW,
- Wyjaśnia powiązania pomiędzy wzorcami projektowymi a ramami programistycznymi,
- Omawia wzorzec projektowy MVC oraz wzorze PRG,
- Objaśnia przebieg obsługi żądania skierowanego do aplikacji zaimplementowanej przy użyciu Node.js.

Umiejętności

Po ukończonym kursie student / studentka:

- Buduje prostą funkcjonalność serwerową przy zastosowaniu Node.js,
- Uzupełnia wiedzę na temat środowiska Node.js stosując anglojęzyczne źródła dostępne online.

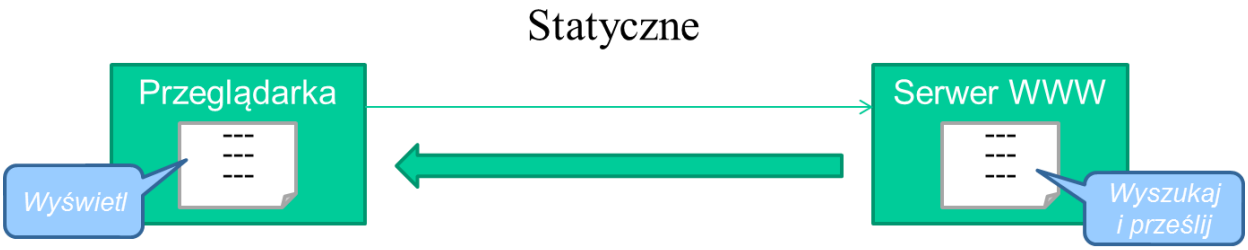
Uzasadnienie

Praktyczne zapoznanie się z ramą programistyczną (stanowiącą zwykle niezbędne narzędzie programisty aplikacji WWW) nie zawsze wymaga dogłębnej znajomości języka programowania. Kluczowe jest natomiast zawsze skuteczne poznanie jej architektury, pojęć, na których oparto jej działanie i wzorców projektowych, które są przezeń wspierane.

Aplikacje WWW - wprowadzenie

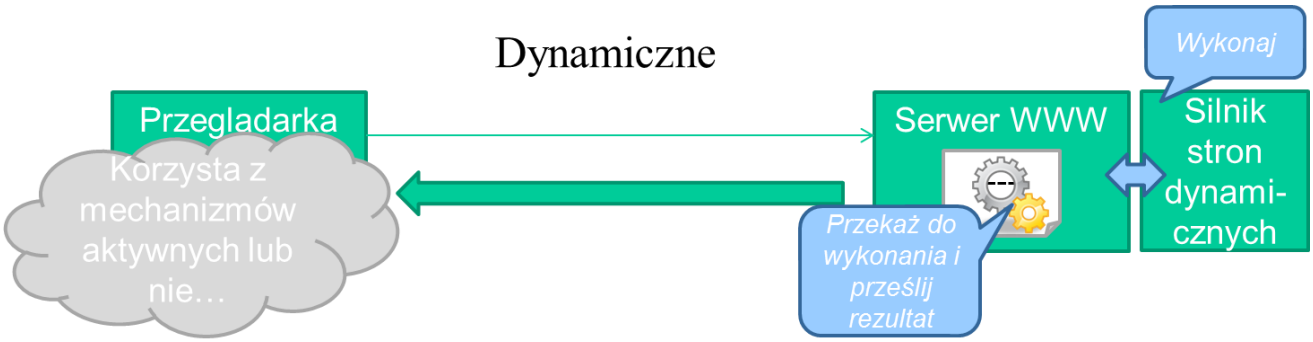
Dynamiczna strona WWW

Na początku historii systemu WWW, działanie oprogramowania po stronie serwera było bez porównania prostsze. Gotowe mechanizmy serwera plus pewna liczba ustawień konfiguracyjnych pozwalały mu na lokalizowanie na dysku i serwowanie dokumentów statycznych bez potrzeby sięgania po jakiekolwiek specyficzne dla danej witryny elementy logiki aplikacyjnej.



Wprowadzenie technologii dynamicznych dokumentów WWW zmieniło sytuację diametralnie – po stronie serwera potrzeba, niejako na jego „zapleczu" zbudować interfejsy, które pozwolą programowi generującemu stronę dynamiczną sięgnąć do wszystkich szczegółów zlecenia, którymi dysponuje serwer WWW, a następnie przekazać mu (celem wysyłki do klienta) dynamicznie wygenerowaną stronę (a być może też część nagłówków odpowiedzi protokołu HTTP)[1].

Na razie, jak zasugerowano poniższym rysunkiem, nie wnikamy w kwestię, czy po stronie klienta jest uruchamiana jakaś funkcjonalność aplikacyjna, czy też jest to raczej w pełni statyczny dokument.



Zauważmy, że teoretycznie jeden taki monolityczny program (wyposażony zapewne w mnóstwo instrukcji warunkowych) mógłby służyć obsłudze wszystkich żądań kierowanych do danego serwera (zależnie od metody żądania - GET, POST etc., nagłówków protokołowych oraz, przede wszystkim, adresu URL żądania). Rozwiązanie takie byłoby jednak nieakceptowalne ze względów panowania nad złożonością oraz zapewnienia łatwości utrzymania aplikacji.

Kontrola bieżąca	[ ]	*komentarz odpowiedzi*
Technologia dynamicznych stron WWW pozwala programiście:		
Odczytać całość komunikatu żądania włącznie z nagłówkami protokołowymi oraz istotnymi szczegółami żądania pochodzącymi z niższych warstw protokołowych.	x	Tak | Odpowiedź niepełna
Uruchamiać programy w środowisku przeglądarki klienta		Nie. W tym przypadku mówilibyśmy o stronach aktywnych

[1] Przykładem takiego rozwiązania jest standard interfejsu CGI (zainteresowanych odsyłamy do opisu zawartego w archiwalnej wersji kursu w module 4).

Generować wyłącznie ciało dokumentu HTML przekazywanego do przeglądarki klienta		Nie. Możliwości są tu szersze
Generować wyłącznie całość dokumentu HTML przekazywanego do przeglądarki klienta		Nie. Możliwości są tu szersze
Generować dokument HTML przekazywany do przeglądarki klienta oraz, w miarę potrzeb, również nagłówki protokołowe odpowiedzi	x	Tak | Odpowiedź niepełna

Wzorce projektowe a ramy programistyczne

Na przykładzie wyżej postawionego problemu możemy zaobserwować rozwój tzw. wzorców projektowych i ich wsparcia narzędziowego. Można to streścić w następujących punktach.

1. Pewien rodzaj problemów programistycznych staje się na tyle powszechny, że autorzy rozwiązań obserwują pewne powtarzalne elementy wspólne dla wszystkich zrealizowanych produktów. Pojawia się dążenie do ich „wyciągnięcia przed nawias", najlepiej poprzez wsparcie gotowym komponentem oprogramowania, ewentualnie parametryzowanym określonymi właściwościami.
2. Dla ww. problemów kształtują się wypracowane i sprawdzone rozwiązania, którym nadaje się nazwę.
3. Zależnie od poziomu ogólności i powszechności ww. wzorców, dąży się do skonstruowania dlań albo dedykowanego wsparcia w językach programowania, albo też wsparcia w postaci bibliotek i ram programistycznych.
4. Ponieważ niektóre problemy programistyczne mają wprawdzie powtarzalne, generyczne rozwiązania, ale ich pełna realizacja silnie zależy od konkretnego przypadku (np. od dokładnego zestawu atrybutów obiektu występującego w dziedzinie problemowej), to owa „parametryzacja" rozwiązania może bardzo mocno przenikać całe oprogramowanie. W takim przypadku dostarczenie bibliotek może nie wystarczyć dla stworzenia produktywnego i wygodnego rozwiązania. Zamiast tego, interfejsy ram programistycznych mogą przyjąć kształt DSL (*domain specific language*), czyli rodzaju odrębnego języka programowania zaprojektowanego specjalnie dla realizacji określonych potrzeb (np. – budowy warstwy prezentacyjnej aplikacji WWW).

Wzorzec MVC

Gdybyśmy zdecydowali się na bardziej łagodne wprowadzenie, zapewne zaczęlibyśmy od przykładu w technologii, która nie egzekwuje stosowania wzorców projektowych. Wówczas przykład przetwarzania danych przesłanych od klienta i wyświetlania ich potwierdzenia mógłby opierać się na raptem dwóch plikach dowolnie umieszczonych po stronie serwera:

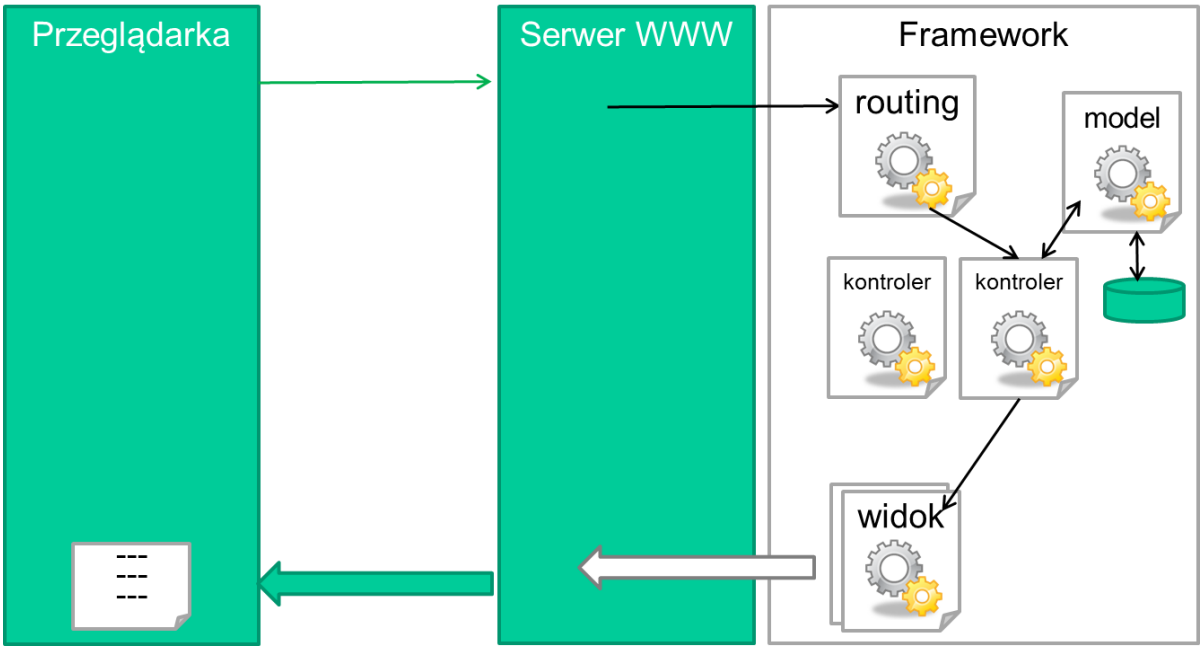
1. Pliku czystego HTML z formularzem służącym wysyłce danych na serwer.
2. Pliku w technologii dokumentów dynamicznych, stanowiącego stronę wyświetlaną użytkownikowi jako potwierdzenie przesłanych danych, a zarazem zawierającego odpowiednie instrukcje przetwarzania, w tym np. trwałego zapisu przesłanych przez użytkownika danych.

Ponieważ Node.js nie narzuca rygoru stosowania określonych wzorców projektowych i organizowania według nich struktury kodu, podobny minimalistyczny przykład przytoczono niżej.

Natomiast budowa bardziej złożonych aplikacji w praktyce skłania do wykorzystania ram programistycznych (*frameworks*), które wspierają separację zagadnień i wielokrotne użycie kodu, wprowadzając w tym celu często właśnie wzorzec MVC lub jego pochodne.

Przykładem realizacji wzorca MVC będzie dla nas rama Express.js, dostępna jako pakiet NPM. Nim zajmiemy się realizacją tego wzorca na gruncie Node.js/Express.js, przyjrzyjmy się jego koncepcji. Ogólnie, zamysł polega na tym, aby zdekomponować problem na jasno wyodrębnione aspekty, celem:

- Lepszego opanowania złożoności,
- Podziału odpowiedzialności pomiędzy twórcami aplikacji (np. projektant UI vs programista aplikacji).



Koncepcję wzorca można podsumować następująco.

Mechanizmy ramy zapewniają przekazywanie danych od i do serwera WWW oraz wyznaczają konwencję nazewniczą ułatwiającą zdefiniowanie powiązań pomiędzy współpracującymi komponentami. Dla odebranego, za pośrednictwem serwera WWW, żądania, jest tutaj wykonywane:

1. Komponent routingu bada adres URL żądania oraz metodę (POST, GET, PUT itd.). Na tej podstawie decyduje, do którego spośród potencjalnie wielu kontrolerów przesłać zlecenie obsłużenia żądania.
2. Kontroler uruchamia funkcjonalność odpowiednią dla otrzymanego żądania. W najprostszym przypadku mógłby on samodzielnie wygenerować odpowiedź do klienta, jednak w praktyce przekazuje on odpowiedzialność za wytworzenie dokumentu prezentowanego użytkownikowi do odpowiedniego komponentu widoku.
3. Widok również może obejmować wywołanie funkcjonalności generującej dynamiczną treść. Może przy tym korzystać z danych przygotowanych dlań na skutek wykonania funkcjonalności kontrolera.
4. Gdy zlecenie wymaga odczytu lub zapisu trwałych danych, kontroler odwołuje się do funkcjonalności modelu. Ten hermetyzuje w sobie zagadnienia zarządzania danymi – w tym m.in. zwykle odwzorowania obiektowo-relacyjne.

Kontrola bieżąca	( )	*komentarz odpowiedzi*
Jeśli w ramach obsługi żądania skierowanego do aplikacji WWW technologii realizującej wzorzec MVC niezbędne jest np. dokonanie pewnych obliczeń, to najbardziej naturalnym miejscem ich implementacji będzie komponent:		
routingu		Odpowiedź błędna

kontrolera	x	Tak
modelu		Nie. Model poprzestaje raczej na obsłudze trwałych danych
widoku		Nie. Unikamy wplatania logiki aplikacyjnej do widoku

Inne wzorce

Większość wzorców projektowych występujących w aplikacjach WWW to wzorce ogólnego zastosowania, znane też z innych rodzajów oprogramowania. Spośród specyficznych dla środowiska WWW wzorców, warto natomiast wspomnieć jeszcze o PRG (*Post, Redirect, Get*).

Wzorzec ten rozwiązuje problem niespójności nawigacji (i związanej z tym niewygody dla użytkownika oraz ewentualności niepożądanych powtórzeń operacji) związanej z realizacją żądań protokołu HTTP wołanych metodą POST.

Stosowanie ww. metody dla operacji powodujących zmianę stanu danych na serwerze (a więc np. wysłania formularza rejestracyjnego, zalogowania się użytkownika, włożenia towaru do koszyka pliku itp.) jest ważne, gdyż metoda ta gwarantuje, że każde jej wywołanie spowoduje faktyczne jej obsłużenie po stronie serwera. Jest to istotna różnica względem przeznaczonych do odczytu zasobów żądań metody GET, gdyż ta ostatnia dopuszcza możliwość korzystania z wcześniej wygenerowanej reprezentacji zasobu (zbuforowanej po stronie serwera lub przeglądarki) – a tym samym możliwość uniknięcia ponownego przetworzenia żądania przez serwer.

Są zatem rodzaje zleceń dla których możemy i powinniśmy stosować POST. Wiąże się z tym jednak następujący problem. Żądanie POST realizuje wówczas dwoistą rolę:

1) Służy przekazaniu zlecenia na serwer (i obsłużeniu przesłanych danych)
2) Podobnie jak żądania GET – zwraca użytkownikowi dokument, który zostanie mu wyświetlony w przeglądarce jako rezultat wykonanego zlecenia.

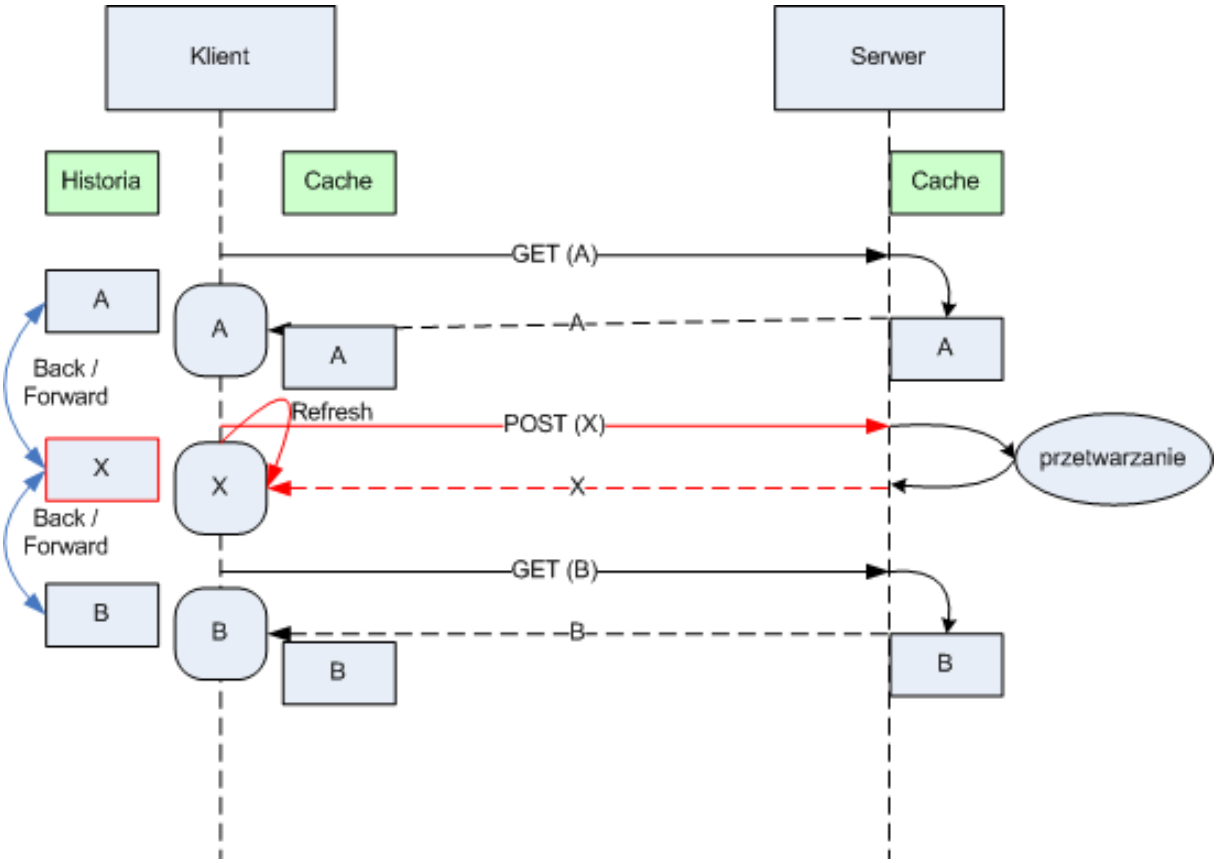
Wobec tego drugiego zadania, wołania POST będą, podobnie jak GET, zapisywane przez przeglądarkę wraz z ich URL w historii odwiedzonych stron.

Wobec tego pierwszego zadania natomiast – rezultaty przesłane jako odpowiedź z wołania POST nie będą buforowane.

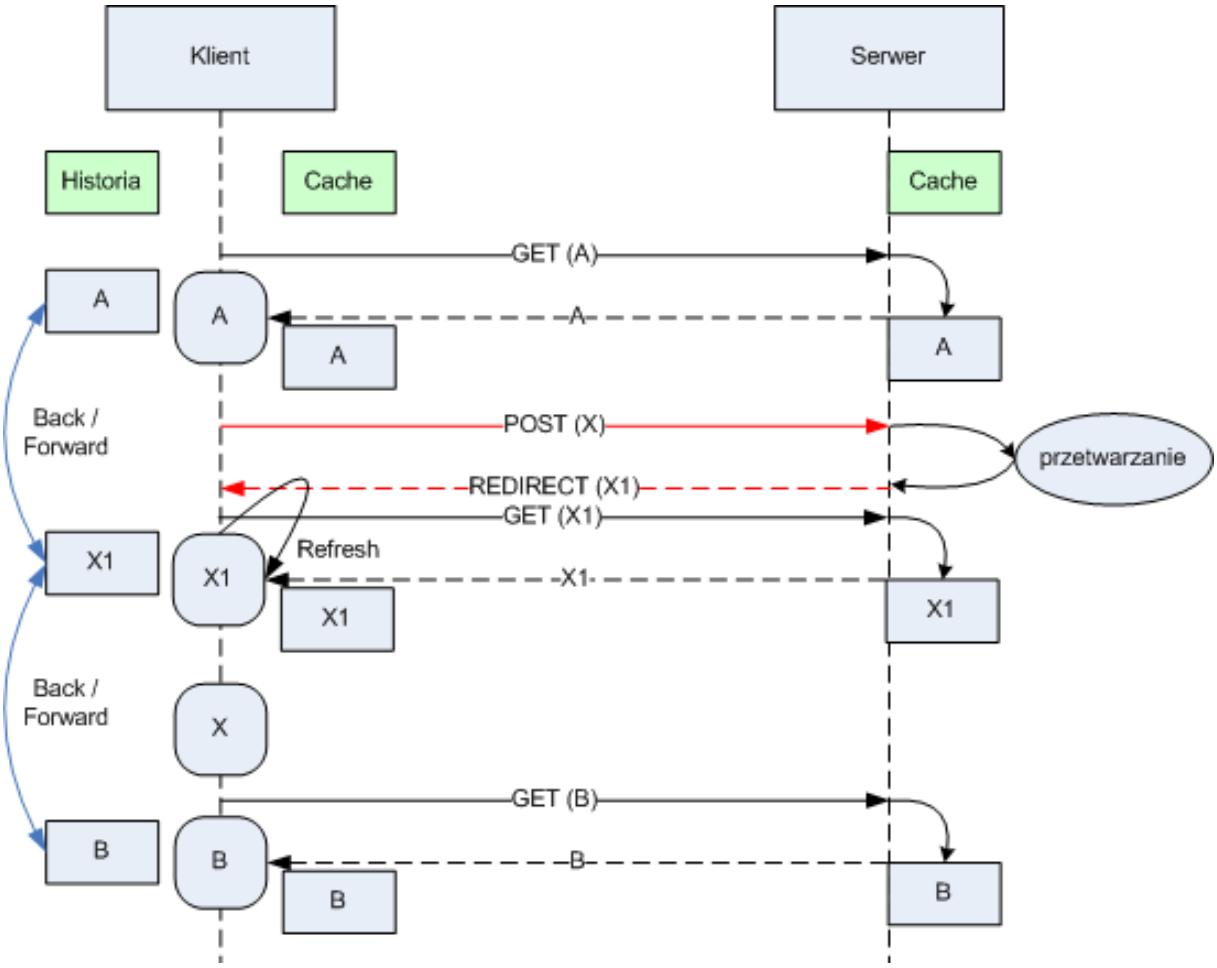
Jeśli więc obsługa żądania POST w bezpośredni sposób przesyła klientowi dokument-rezultat wywołania, to rodzi się następująca niedogodność. Prześledźmy poniższy diagram sekwencji:

1) Użytkownik pobiera (GET) stronę A (np. zawierającą formularz)
2) Użytkownik wydaje polecenie *Submit*, powodujące wysłanie danych metodą POST na serwer pod adres URL równy X. Strona X zostaje, podobnie jak pozostałe, zapisana w historii odwiedzonych stron.
3) Po odebraniu odpowiedzi z żądania POST użytkownik nawiguje dalej (GET) wybierając stronę B.

W przypadku, gdy użytkownik zechce się cofnąć w nawigacji (komenda Back) ze strony B do X lub odświeżyć wyświetloną stronę X, otrzyma komunikat, że strona X nie może zostać wyświetlona bez ponownego wykonania żądania POST.



Rozwiązaniem powyższego problemu jest zastosowanie w odpowiedzi z wywołania POST (X) zlecenia przekierowania (*redirect*) pod inny adres URL, tutaj oznaczony przez nas X1. Strona ta może, podobnie jak wcześniej rezultat wołania X, zawierać podsumowanie przesłanych danych. Co ważne, w przypadku takiego przekierowania, tylko docelowy adres przekierowania (a więc u nas tylko X1 a nie X) zostaje zapisany w historii odwiedzonych stron. A ponieważ to ponowne żądanie (X1) realizowane automatycznie przez przeglądarkę jest już wykonywane metodą GET, unikamy problemów związanych z istnieniem w historii nawigacji niebuforowanego zlecenia POST.



Kontrola bieżąca	[ ]	*komentarz odpowiedzi*
Do cech żądania HTTP realizowanego metodą POST zaliczamy:		
Buforowanie jego rezultatów przez przeglądarkę		Odpowiedź błędna. Czemu służy żądanie POST i czy jest sens je buforować?
Buforowanie jego rezultatów po stronie serwera		Odpowiedź błędna. Czemu służy żądanie POST i czy jest sens je buforować?
Niezapisywanie takiego żądania w historii przeglądarki		Niezupełnie. Jeśli nie zastosujemy dodatkowych zabiegów – żądanie to pojawi się w historii odwiedzanych stron
Przekazywanie go na serwer dokładnie tyle razy, ile razy zostało przez klienta wywołane	x	Tak | Odpowiedź niepełna
Możliwość przekazania parametrów w ciele komunikatu żądania	x	Tak | Odpowiedź niepełna

Node.js – wprowadzenie

Geneza i krótka charakterystyka

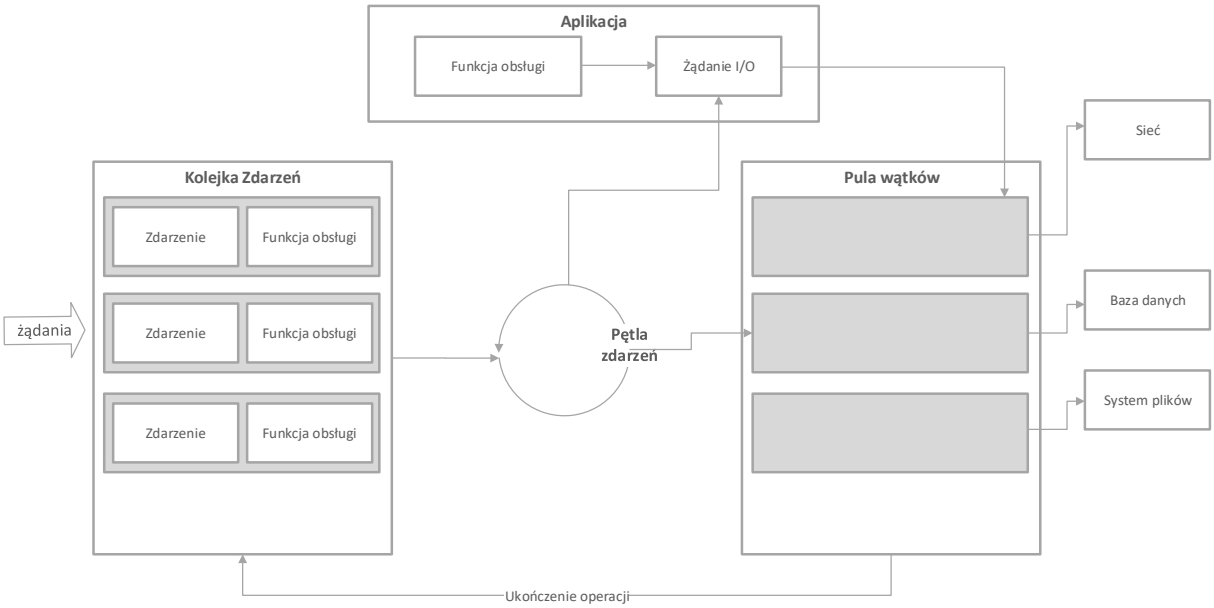
Node.js jest środowiskiem programowania opartym na języku JavaScript rozwijanym w formule open source. Wykorzystuje silnik „V8” stworzony oryginalnie dla przeglądarki Chrome. Towarzyszy mu środowisko menedżera pakietów NPM.

Node.js realizuje paradygmat full-stack JavaScript, w co wpisuje się m.in. wykorzystanie formatu JSON (JavaScript Object Notation) jako podstawowego formatu wymiany danych.

Pętla zdarzeń

Charakterystyczną cechą programowania w Node.js, wynikłą z architektury tego środowiska jest asynchroniczny styl programowania. Nie doświadczymy tego intensywnie w bieżącym ćwiczeniu, będzie to jednak wyraźnie widoczne później.

Rozwiązanie to jest odpowiedzią na kosztowność operacji wejścia-wyjścia. Są one stosunkowo długotrwałe, toteż gdyby realizować je w sposób synchroniczny, przeważająca część czasu działania programu przypadłaby na oczekiwanie na ukończenie takiej operacji. Aby zapewnić zdolność aplikacji do przetwarzania napływających żądań WWW, konieczne więc byłoby zrównoleglenie pracy przez powoływanie osobnych wątków albo zastosowanie odpowiedniego wzorca przy programowaniu jednowątkowym. Wzorcem takim jest *Reactor*. Jego realizacja w Node.js opiera się na tzw. pętli zdarzeń. Zamiast wstrzymywać wykonanie programu w oczekiwaniu na realizację operacji, umożliwia ona wyposażenie zlecenia w kod, który należy wykonać po ukończeniu operacji. Po ukończeniu taki kod trafi do kolejki zdarzeń, gdzie będzie oczekiwał na wykonanie w kolejce.



Koncepcyjnej jednowątkowości tego wzorca nie powinniśmy jednak utożsamiać z całkowicie jednowątkowym działaniem aplikacji; m.in. operacje I/O są realizowane wielowątkowo.

Instalacja środowiska

Ćwiczenia i projekt możemy realizować albo w lokalnie skonfigurowanym środowisku Node.js, albo w jednym ze środowisk dostępnych w chmurze. Tu zakładamy ten pierwszy scenariusz.

Node.js na pożądaną platformę możemy pobrać z witryny projektu: https://nodejs.org/en/download/

Dla upewnienia się, czy instalacja przebiegła pomyślnie, warto w oknie terminala / linii poleceń wywołać:

```
node -v
```

Powinna zostać wyświetlona wersja środowiska.

Wkrótce przyda nam się też wsparcie dla tworzenia i diagnostyki tworzonego kodu. W tym celu warto zaopatrzyć się w jedno ze środowisk IDE wspierających Node.js. Może to być np. JetBrains Web Storm, czy Visual Studio Code.

Uruchamianie kodu Node.js

W pierwszym kroku warto zapoznać się z tzw. trybem REPL (*Read, Eval, Print, Loop*). Wywołując w linii poleceń komendę node bez parametrów uruchomimy tryb interakcyjny, w którym w kolejnych wprowadzanych wierszach kodu możemy definiować typy i zmienne a następnie je wykorzystywać we wprowadzanych wyrażeniach czy instrukcjach.

W przypadku bardziej złożonego kodu zechcemy oczywiście raczej przygotować go w pliku a następnie uruchomić. W tym wypadku komendę node zaopatrzymy w argument w postaci nazwy pliku z kodem.

Jeśli mamy już zainstalowane wybrane przez nas IDE, to oba ww. tryby będziemy mogli realizować z jego pomocą, bez konieczności wywoływania osobnego okna linii poleceń.

Środowisko działania skryptu

Zapoznaj się ze środowiskiem widocznym dla naszego kodu przy pracy w ramach REPL. Znajdziemy tam obiekt `global`, który posiada szereg właściwości. Możemy się również przekonać, że `global === this`.

Mamy tu m.in. obiekt reprezentujący bieżący proces (właściwość `process`). Sprawdź, że pracę ze środowiskiem można zakończyć wołając nań operację `exit()`.

Znajdziemy tu także właściwość `console` z metodą `log()`, toteż można będzie stosować ten sam sposób prezentowania wyjścia co w przypadku programowania w środowisku przeglądarki.

Przykład – operacje na plikach

Zbudujmy prosty przykład wykonujący operacje plikowe.

```
const fs = require('fs');

fs.writeFileSync('hello.txt','Hello world!');
```

Pakiety i moduły

W powyższym przykładzie musieliśmy wyjść poza właściwości dostępne bezpośrednio w środowisku i załadować metodą `require()` jeden z modułów standardowych (*Node Core Modules*). Pełną ich liczbę i specyfikację znajdziemy na: https://nodejs.org/api/

Znacznie szerszy asortyment funkcjonalności znajdziemy w ekosystemie pakietów NPM (zob. http://npmjs.com). Celem wykorzystania takich modułów musimy wpierw zainstalować odpowiedni pakiet w naszym środowisku. Więcej na ten temat znajdziemy w kolejnej lekcji.

Prosty serwer HTTP

Poniżej pokazano szkielet implementacji prostego serwera HTTP, wykorzystującego do pobierania żądań i konstruowania odpowiedzi moduł `http`.

```
const http = require('http');

function obsluga(req, res){

      console.log(req);

}

const server = http.createServer(obsluga);

server.listen(3000, localhost);
```

Szkielet ten powołuje instancję serwera i łączy ją z funkcją callback, która będzie uruchamiana dla każdego nadchodzącego żądania. Następnie rozpoczyna nasłuch na wskazanym porcie i adresie (tu – localhost).

W ww. szkielecie nie konstruujemy jeszcze żadnej odpowiedzi dla klienta – przykład ten pokazuje natomiast poprzez wyprowadzenie zawartości obiektu `req` na ekran, jakimi informacjami o nadchodzącym żądaniu dysponuje programista.

Tę instrukcję wyświetlania możemy w dalszych krokach zastąpić instrukcjami konstruującymi odpowiedź – np.

```
res.setHeader('Content-Type', 'text/html');

res.write('<html>');

//tu kolejne res.write(…); konstruujące dokument wynikowy

res.end(); // zamknięcie i przekazanie odpowiedzi do wysyłki
```

Powyższy przykład może posłużyć jako zaczątek rozwiązania zadania związanego z tą lekcją. Przydatny będzie w tym wypadku odczyt szczegółów żądania – m.in. właściwość req.url .

Podsumowanie

W niniejszym module połączyliśmy kilka odrębnie wcześniej prezentowanych zagadnień: dokumenty znacznikowe, arkusze stylów oraz mechanizm wykonywania żądań HTTP i uzupełniliśmy je o nowy komponent, jakim jest budowa aplikacji WWW poprzez programowanie dynamicznych dokumentów WWW. W tym celu zastosowano język JavaScript oraz środowisko Node.js. Temat ten będziemy rozwijać w kolejnym module, omawiając m.in. zagadnienia utrwalania danych, udogodnienia ramy Express oraz zagadnienia modularyzacji aplikacji.

Zadanie

Zadanie z niniejszego modułu jest dwuczęściowe:

1) Zaopatrz się w działające środowisko dla budowy aplikacji w Node.js. Wykonaj za jego pomocą następujące zadanie.
Zaimplementuj (bez stosowania dodatkowych frameworków) prosty serwer http, który będzie przyjmował żądania zawierające nazwę żądanej operacji (dodaj, odejmij, pomnóż, podziel) i dwa argumenty oraz wypisywał w postaci minimalistycznej strony HTML wynik takiej operacji.
Szczegóły realizacji nie są narzucone. Argumenty mogą pojawić się np. w URL jako tzw. *query string* (część adresu URL znajdującą się po znaku „?")

2) Ta część nawiązuje z kolei do projektu zaliczeniowego. Na podstawie swojego aktualnego rozeznania w obszarze technologii aplikacji WWW oraz wiedzy z zakresu inżynierii oprogramowania, sformułuj zestaw wymagań, które Twoim zdaniem należałoby wyegzekwować od wykonawcy realizującego aplikację WWW o tematyce zgodnej z tematem Twojego projektu zaliczeniowego. Uwaga! Nie są to wytyczne na zgodność z którymi będzie oceniana Twoja implementacja (która może mieć np. znacznie mniejszy zakres niż nakreślony w tych wymaganiach). Zależy nam tutaj po prostu na zmierzeniu się z problemem specyfikacji wymagań z punktu widzenia zleceniodawcy.