

**CELEM PRACY JEST PRZEDSTAWIENIE TECHNOLOGII
NODE.JS NA PRZYKŁADZIE ZAPROJEKTOWANIEJ
APLIKACJI PRZEZNACZONEJ DO ZARZĄDZANIA
CODZIENNYMI ZADANIAMI.**

Spis treści

1	Przedstawienie ogólne, ideologii oraz przeznaczenia technologii Node.js	3
1.1	Wstęp	3
1.2	Przeznaczenie	4
1.3	Modułowość	4
2	Omówienie architektury Node.js	6
2.1	Paradygmat	6
2.2	Asynchroniczność	7
2.3	Architektura komunikacji	8
3	Założenia i specyfikacja aplikacji	12
4	Opracowanie aplikacji	15
4.1	Mean Stack	15
4.1.1	Wstęp	15
4.1.2	MongoDB	15
4.1.3	ExpressJS	15
4.1.4	AngularJS	16
4.2	Komunikacja	16
4.2.1	Rozwiązywanie problemu komunikacji	16
4.2.2	Kod po stronie klienta	17
4.2.3	Kod po stronie serwera	20
4.3	Wykorzystane moduły	23
4.3.1	Moduły zewnętrzne	23
4.3.2	Moduły wewnętrzne	24
4.4	Diagramy funkcji	24
5	Testy aplikacji	25
5.1	Uzyskanie dostępu do statycznych zasobów serwisu	25
5.2	Rejestracja w serwisie	25

5.3	Potwierdzenie adresu email	26
5.4	Logowanie się w serwisie	27
5.5	Resetowanie hasła użytkownika	28
5.6	Zmiana danych użytkownika	28
5.7	Utworzenie nowej tablicy z zadaniami	29
5.8	Wysyłanie zaproszenia do tablicy	30
5.9	Akceptacja zaproszenia	31
5.10	Odrzucenie zaproszenia	31
5.11	Wyrzucenie użytkownika z tablicy	32
5.12	Dodanie zadania	33
5.13	Dodanie statusu zadania	35
5.14	Usuwanie zadania	37
5.15	Opuszczanie tablicy	37
5.16	Usuwanie tablicy	38
6	Podsumowanie otrzymanych wyników i wnioski na temat środowiska	40
	Bibliografia	41
	Spis rysunków	42

Rozdział 1

Przedstawienie ogólne, ideologii oraz przeznaczenia technologii Node.js

1.1 Wstęp

Node.js czyli cross-platformowy, działający niezależnie od środowiska język programowania napisany w językach c/c++ oraz javascript wydany 27 marca 2009 roku, zaprojektowany przez Ryan'a Dahl'a. Pozwala na tworzenie serverów i narzędzi sieciowych działających po stronie serwera. Przed powstaniem języka kod javascriptowy był wykonywany głównie przez przeglądarkę internetową po stronie klienta, co pozwalało na bezproblemowa manipulacje kodu źródłowego strony przez użytkownika dając możliwość wykonywania złośliwych skryptów, naruszenie bezpieczeństwa baz danych lub uzyskania dostępu do chronionych zasobów servera. Środowisko Node.js może działać niezależnie od środowiska uruchomieniowego. Jest ono zgodne z wieloma systemami operacyjnymi jak Linux, macOS, Microsoft Windows, Non-Stop, czy serwerami Unix. Język ten cieszy się dużą popularnością oraz pozytywnym odbiorem wśród użytkowników dzięki czemu mimo względnie krótkiego okresu życia środowiska zaowocowało ogromną ilością projektów open-source, tysiącami członków należących do społeczności okołojęzykowej oraz powstaniem wydarzeń poruszających tematy okołosrodowiskowe takimi jak NodeConf, Node Interactive lub Node Summit. Obecnie wiele największych firm korzysta z serwerów napisanych w języku node.js. Ich przykładami są między innymi Groupon, IBM, LinkedIn, Microsoft, Netflix, PayPal, Yahoo. Najpopularniejszymi API (application programming interface) wspierającymi edycję oraz debugowanie kodu node.js są Atom, Brackets, JetBrains, Microsoft Visual Studio, NetBeans czy Nodeclipse.

1.2 Przeznaczenie

Node.js zalecany jest do tworzenia aplikacji:

- z dużą liczbą operacji wejścia/wyjścia,
- strumieniowania danych np. video,
- Single Page Applications (SPA),
- udostępniających API w formacie JSON,
- z intensywną wymianą danych w czasie rzeczywistym na wielu urządzeniach np. portale społecznościowe.

Ponieważ jest on szybki i lekki, może być stosowany do pisania między innymi bramki API. API to skrót od Application Programming Interface; opisuje jak poszczególne elementy lub warstwy oprogramowania powinny się komunikować. W praktyce to najczęściej biblioteka oferująca metody, które umożliwiają realizację określonych zadań. Node.js pozwala na zoptymalizowanie pracy oraz uzyskanie skalowalności dzięki asynchronicznemu przetwarzaniu danych dostarczanych do aplikacji w związku z czym idealnie nadaje się do obsługi komunikacji wymagającej pracy w czasie rzeczywistym. Funkcje napisane w node.js wykonują się równolegle, korzystając z tak zwanych wywołań zwrotnych (ang. callback), przeciwnie jak w językach takich jak np. php gdzie program jest wykonywany synchronicznie - linia po linii. Dzięki temu nie powstaje problem blokowania określonych funkcjonalności programu w czasie pracy innych niezależnych jego części. Przy pomocy wywołań zwrotnych możemy zapewnić zasygnalizowanie uzyskanych wyników lub zwrócenie, bądź obsługę błędu powstałego w czasie działania bloku kodu.

1.3 Modułowość

Praca z Node.js opiera się głównie o korzystanie ze zbioru zdefiniowanych w ramach modułów funkcji wspierających określone funkcjonalności. Zapewniają one prace między innymi z plikami systemowymi, z urządzeniami wejścia/wyjścia, protokołami internetowymi (dns, http, tcp, tls/ssl, udp), plikami binarnymi, źródłami danych oraz funkcjami kryptograficznymi. Zmniejszają one złożoność, a co za tym idzie nakład pracy przy tworzeniu własnej funkcjonalności. Dzięki wsparciu package managera (od roku 2010) nazywanego npm programiści mogą bez przeszkód

udostępniać napisane przez siebie moduły i biblioteki lub w prosty sposób zainportowywać ogólnie dostępne moduły i używać ich w swoich projektach. Najpopularniejszymi modułami wykorzystywanymi w celu poprawy jakości oraz zmniejszenia nakładów pracy przy wytwarzaniu oprogramowania są Express.js, Socket.IO, Hapi.js, Sails.js czy Meteor. Npm jest automatycznie włączony w środowisko Node.js. Jest obsługiwany za pomocą linii komend systemu operacyjnego. Moduły są zapisane w formacie CommonJS oraz zawierają pliki informacyjne w formacie Json. Ilość ogólnie dostępnych modułów przekracza obecnie 477,000. Jest to spowodowane możliwością przez każdego użytkownika Node.js, bez potrzeby wcześniejszej rejestracji czy przejścia jakiekolwiek procedury wstępnej udostępnienia napisanego przez siebie kodu. W związku z tym, wiele dostępnych modułów jest niskiej jakości, może zawierać elementy złośliwego oprogramowania lub nie być bezpiecznym dla naszego systemu operacyjnego. Należy bezwzględnie brać te czynniki pod uwagę w przypadku korzystania z nieznanych modułów i najlepiej najbardziej znacząco ograniczyć korzystanie z nich bez wcześniejszej weryfikacji kodu źródłowego. Zabezpieczeniami w celu ochrony użytkowników, które dostarcza npm jest usuwanie pakietów, które zostały zgłoszone przez użytkowników jako naruszające ogólne zasady bezpieczeństwa oraz możliwość wglądu w raporty statystyczne odnośnie ilości pobrań lub ilości zależnych od modułu innych pakietów. Kolejnym zagrożeniem jakim niesie korzystanie z pakietów udostępnionych przez innych użytkowników jest możliwość usunięcia udostępnionego pakietu z repozytorium npm, w konsekwencji uniemożliwiając naszej aplikacji dalsze korzystanie z pakietu. Sytuacja taka miała miejsce, kiedy skrypt zwany „left-side” z którego korzystało ponad 2,486,696 deweloperów został usunięty z repozytorium powodując tak zwany efekt domina będący przyczyną błędów w kolejnych aplikacjach deweloperów. Npm korzysta, tak jak i inne globalnie działające narzędzia JavaScriptowe z plików zależności w formacie json. Opisuję one wersję wykorzystywanych modułów i pozwalają za pomocą jedno linowej komendy na szybką i łatwą instalację wszystkich używanych pakietów w lokalnym środowisku.

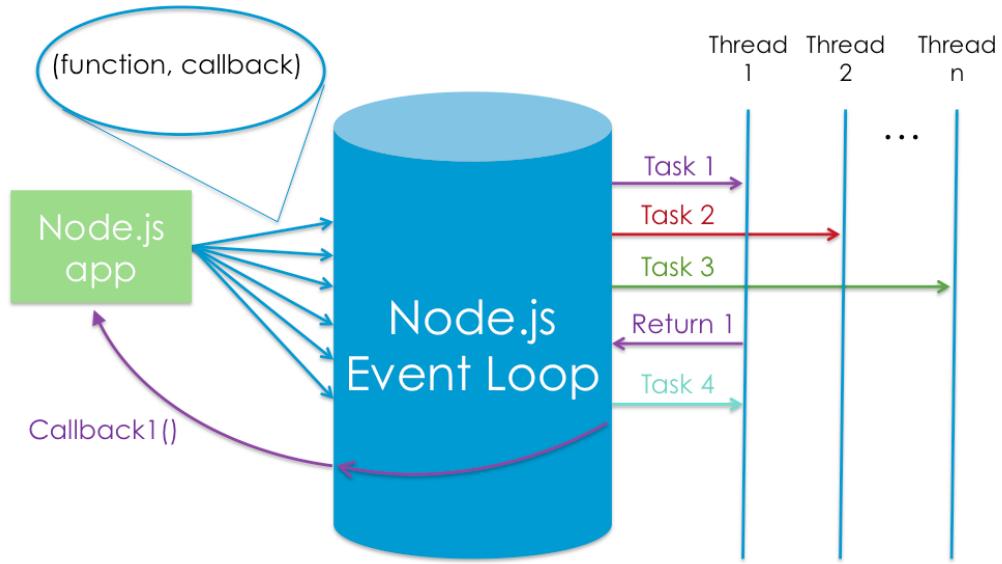
Rozdział 2

Omówienie architektury Node.js

2.1 Paradygmat

Architektura Node.js pozwala na tworzenie oprogramowania sterowanego zdarzeniami (event-driven programming). Jest to paradygmat programowania w którym kolejność wykonywania kodu zależy od zdarzeń mających miejsce w czasie życia aplikacji (run time) na przykład interakcji użytkownika czy otrzymania określonych sygnałów. W przypadku języka Node.js, kiedy aplikacja pełni role serwera paradygmat ten najczęściej dotyczy przetwarzania zapytań otrzymywanych ze strony klienta oraz uruchomionych przez nie zdarzeń-funkcji. W aplikacji sterowanej zdarzeniami należy wyróżnić pętle główną, która jest odpowiedzialna za obsługę zachodzących w czasie rzeczywistym zdarzeń czyli wywoływanie triggerów jako wywołań zwrotnych. Są to wyodrębnione części kodu, które po wykonaniu swojego zadania zwracają określoną wartość lub obiekt. W Node.js jest to na przykład funkcja nasłuchująca określonego adresu pod który klienci kierują określone zapytania. Pozwala to na wykonywanie wielu zadań jednocześnie i niezależnie. Zapewnia przyśpieszenie wykonywania skomplikowanych funkcji programu. Daje możliwość przykładowo jednoczesnego zapisywania danych do bazy, przetwarzania innej części zapytania i przygotowywanie odpowiedzi w jednym okresie czasu. Node.js zapewnia w ten sposób działanie asynchroniczne, bez bezpośredniego użycia technologii wielowątkowej. Wychodzi na przeciw problemowi tworzenia oraz kontrolowania aplikacji współbieżnych spełniających zadanie serwera, które są trudne do zaimplementowania w wielu językach programowania oraz często prowadziły do niesatysfakcjonującej wydajności. Język został stworzony na silniku V8 javaScript napisanym w języku C++, wyprodukowanym przez Google, wykorzystywany w przeglądarkach Google Chrome który porzuca tradycyjną ideę interpretowania kodu javaScriptowego linia po linii zapewniając w zamian kompilacje do odpowiednio zoptymalizowanego kodu

maszynowego przed jego wykonaniem, a co za tym idzie większą wydajnością podczas działania programu. Zapewnia on również odpowiednie zarządzanie pamięcią dla obiektów, zciągając z programistów odpowiedzialność alokowania oraz zwalniania zajętych zasobów.



Rysunek 2.1: Pętla zdarzeń w środowisku Node.js - źródło: <http://galaxy.agh.edu.pl>

2.2 Asynchroniczność

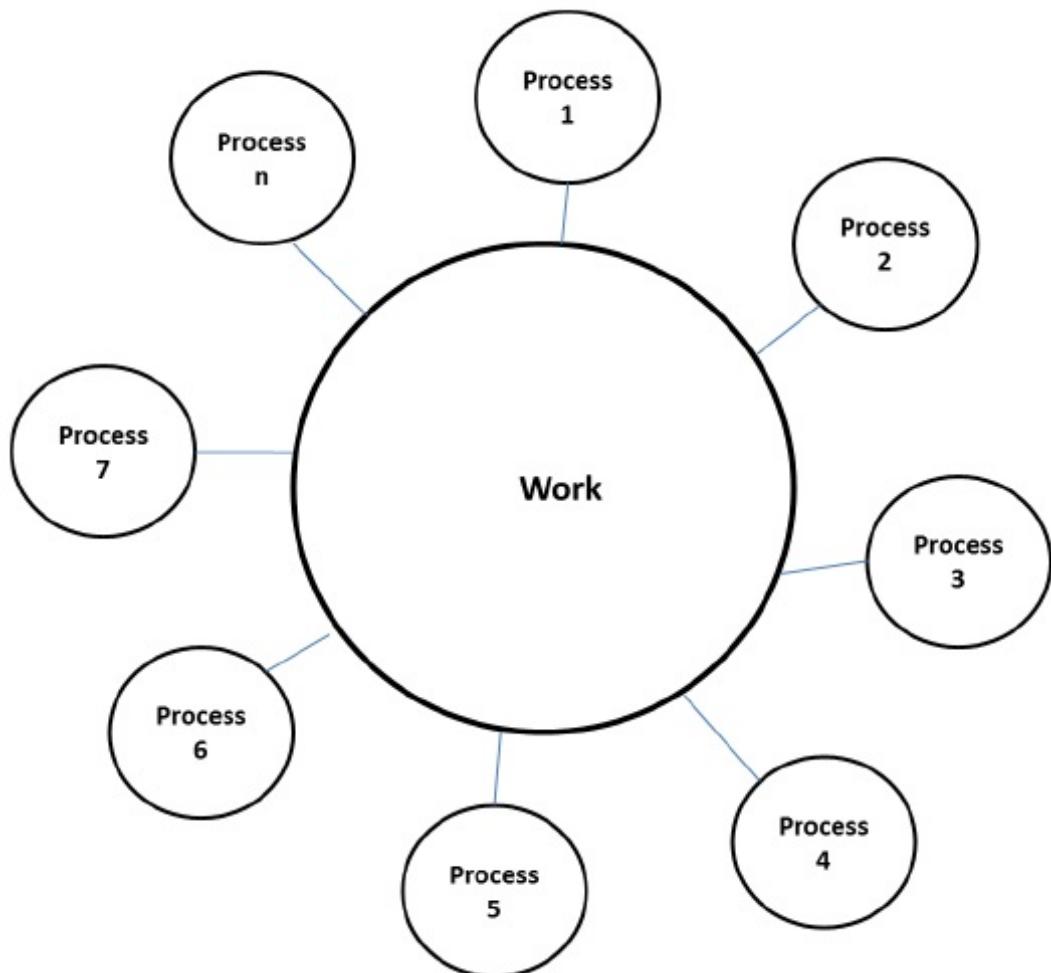
Node.js pracuje wykorzystując tylko jeden wątek używając technologii asynchronicznego wejścia/wyjścia. Zapewnia to synchronizacje wykonywania wielu operacji bez konieczności czekania na zakończenie operacji i zwolnienie dostępu do zasobu poprzez zarządzanie żądaniami wejścia/wyjścia w oderwaniu od wątków wykonywania. Standardowe operacje synchroniczne powodują zablokowanie dalszego wykonywania wątku do czasu zakończenia przetwarzania. W rezultacie określony wątek może zainicjować tylko jedno zadanie wejścia/wyjścia w jednym momencie. Wywołanie funkcji API nie czeka na wyniki, dzięki czemu nie blokujemy wątku wykonawczego. Po zakończonym wywołaniu uruchomiona zostaje funkcja zwrotna lub ogłoszone zostaje zdarzenie w poszczególnych częściach wykonawczych. Pomimo, że Node.js działa na jednym wątku z pętlą zdarzeń, dzięki asynchroniczności potrafi on obsłużyć więcej zapytań niż np. serwer HTTP Apache. Uzyskanie wielowątkowości pomimo korzystania tylko z jednego wątku jest zapewnione dzięki użyciu wzorca

projektowego obserwator, w którym jeden obiekt nazywany przedmiotem obserwowanym określa zależności względem obserwujących go innych obiektów, poprzez wywoływanie ich metod dla określonych własnych stanów. W celu obsługi wszystkich asynchronicznych funkcji oraz wątku głównego Node.js korzysta z biblioteki multiplatformowej języka c - libuv. Biblioteka ta została stworzona specjalnie na potrzebny Node.js, ale jest również wykorzystywana w wielu innych technologiach np racer: (obsługa serweroó w języku ruby), czy Trevi: (silnik do obsługi aplikacji internetowych w języku swift). Wykorzystuje ona określony zbiór wątków do równoległej obsługi wielu operacji wejścia/wyjścia bez wzajemnego ich blokowania. Wadą pracy Node.js przy wykorzystaniu tylko jednego wątku jest brak skalowania pod względem możliwości zwiększenia ilości rdzeni procesorów na których wykonuje się program, bez użycia dodatkowych modułów takich jak cluster (zapewnia możliwość łatwego tworzenia pochodnych procesów, które współdzielą porty serwera), StrongLoop PM (zapewnia zarządzanie procesem produkcyjnym aplikacji Node.js ze wsparciem odpowiedniego zarządzania zasobami, wdrożeniami wielo hostowymi oraz interfejsem graficznym), czy PM2 (zarządzanie procesem produkcyjnym, ze wsparciem odpowiedniego zarządzania zasobów z możliwością nieprzerwanego działania aplikacji przy użyciu przenoszenia jej między domenami bez potrzeby zatrzymywania pracy). Kolejną możliwością na uniknięcie tego ograniczenia jest zmiana ilości wątków należących do zbioru wykorzystywanego przez bibliotekę libuv. Wątki te działają na wielu rdzeniach systemu na którym działa serwer. Działanie Node.js jednocześnie na wielu procesach jest zapewnione właśnie dzięki zbiorowi wątków dostarczanych przez bibliotekę libuv. Wątek główny przydziela zadania dla wątków kolejno ze współdzielonej kolejki funkcji, które wątki pochodne mają za zadanie wykonać. Kiedy wątek pochodny zakończy wykonywanie przydzielonego mu zadania informuje o tym wątek główny poprzez wywołanie określonego wywołania zwrotnego. Z przyczyny odpowiedzialności przez wątek główny do odebrania wszystkich wywołań zwrotnych, funkcja wykonywana przez jeden wątek pochodny może wstrzymać działanie całej asynchronicznej operacji, a co za tym idzie zmniejszyć jej całkowitą wydajność. Biblioteka libuv zajmuje się odpowiednim podziałem zadań oraz przydzieleniem zasobów tak aby w jak najlepszy sposób wywalczyć nakład pracy między wieloma wątkami.

2.3 Architektura komunikacji

W tradycyjnym podejściu odpowiedzialne za obsługę przychodzących połączeń są określone wątki lub procesy systemu operacyjnego, które w porównaniu z technologią Node.js wymagają względnie więcej zaalokowanych zasobów. W celu obsługi za-

pytania przychodzącego do aplikacji Node.js, program rejestruje się w systemie operacyjnym, aby od każdego przychodzącego połączenia otrzymać odpowiednie wywołanie zwrotne, przez co nie potrzebuje procesów czy wątków aby uzyskać możliwość obsługi klientów, a tylko własnej pętli głównej, do której wykonywania powraca po przetworzeniu każdego wywołania wstecznego. W czasie pracy każde przychodzące połączenie otrzymuje odpowiednią ilość zasobów na stercie programu, więc nie ma również potrzeby alokowania zasobów dla każdego z wątków czy procesów osobno. Najbardziej popularna metoda na zarządzanie komunikacją między instancjami korzystającymi z serwera używającego technologii Node.js jest



Rysunek 2.2: Model programowania współbieżnego wykorzystywany przez Node.js - źródło: www.tutorialspoint.com/parallel_algorithm/

wykorzystanie frameworku Restful Api popularnie zwanego Rest. Jest to wzorzec architektury oprogramowania który opisuje sposób operowania zapytaniami pomiędzy Api, w prosty sposób poprzez obsługę zadań oraz odpowiedzi. Został on następcą protokołu sieciowego SOAP (Simple object Access protocol), stając się wiodącym standardem. Pozwala on na eliminację zbędnej pracy i czasu wymaganego na integracje. Daje możliwość na stworzenie komunikacji bez potrzeby wiedzy na temat instancji korzystających z przesyłanych zasobów - może integrować środowiska napisane w różnych językach działające na różnych platformach. Wykorzystuje proste zapytania przy użyciu protokołu http, oraz jego popularnie stosowanych metody takich jak post, get, put, delete oraz bardziej złożonych i używanych żadziej jak options, head, trace oraz connect. Opis typu zasobów jakie wymieniamy jest określony w nagłówku informacji przez kod statusu http. Poniższa tabelka prezentuje poszczególne typy statusów:

1xx Informational		
100 Continue	101 Switching Protocols	102 Processing (WebDAV)
2xx Success		
★ 200 OK	★ 201 Created	202 Accepted
203 Non-Authoritative Information	★ 204 No Content	205 Reset Content
206 Partial Content	207 Multi-Status (WebDAV)	208 Already Reported (WebDAV)
226 IM Used		
3xx Redirection		
300 Multiple Choices	301 Moved Permanently	302 Found
303 See Other	★ 304 Not Modified	305 Use Proxy
306 (Unused)	307 Temporary Redirect	308 Permanent Redirect (experimental)
4xx Client Error		
★ 400 Bad Request	★ 401 Unauthorized	402 Payment Required
★ 403 Forbidden	★ 404 Not Found	405 Method Not Allowed
406 Not Acceptable	407 Proxy Authentication Required	408 Request Timeout
★ 409 Conflict	410 Gone	411 Length Required
412 Precondition Failed	413 Request Entity Too Large	414 Request-URI Too Long
415 Unsupported Media Type	416 Requested Range Not Satisfiable	417 Expectation Failed
418 I'm a teapot (RFC 2324)	420 Enhance Your Calm (Twitter)	422 Unprocessable Entity (WebDAV)
423 Locked (WebDAV)	424 Failed Dependency (WebDAV)	425 Reserved for WebDAV
426 Upgrade Required	428 Precondition Required	429 Too Many Requests
431 Request Header Fields Too Large	444 No Response (Nginx)	449 Retry With (Microsoft)
450 Blocked by Windows Parental Controls (Microsoft)	499 Client Closed Request (Nginx)	
5xx Server Error		
★ 500 Internal Server Error	501 Not Implemented	502 Bad Gateway
503 Service Unavailable	504 Gateway Timeout	505 HTTP Version Not Supported
506 Variant Also Negotiates (Experimental)	507 Insufficient Storage (WebDAV)	508 Loop Detected (WebDAV)
509 Bandwidth Limit Exceeded (Apache)	510 Not Extended	511 Network Authentication Required
598 Network read timeout error	599 Network connect timeout error	

Rysunek 2.3: Kody statusu protokołu http - źródło: <http://mokandra.blogspot.fi/2015/10/http-status-code-definitions.html>

W celu wymiany zasobów framework wykorzystuje najpopularniejsze sposoby opisu struktury służące do opisu obiektów takie jak najczęściej spotykany, najprostszego w użyciu format json (JavaScript Object Notation) oraz ostatnio coraz mniej popularny format xml (eXtensible Markup Language). Różnica pomiędzy dwiema formami jest taka że xml jest językiem nie zdefiniowanych znaczników, a json tylko sposobem na przedstawienie właściwości obiektu. Porównanie obydwu struktur: Struk-

tura w formacie json opisująca listę pracowników:

```
1 {"employees": [  
2     {"firstName": "John", "lastName": "Doe" },  
3     {"firstName": "Anna", "lastName": "Smith" },  
4     {"firstName": "Peter", "lastName": "Jones" }  
5 ]}
```

Ta sama struktura opisana w formacie xml: <employees>

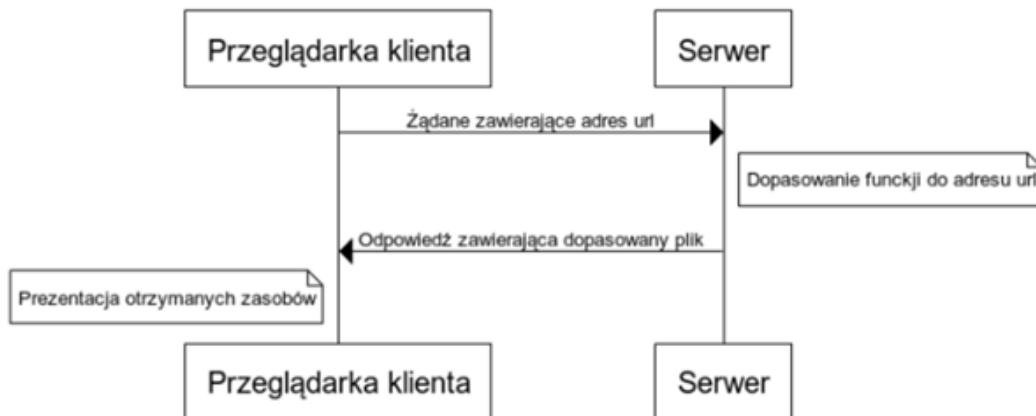
```
1 <employee>  
2     <firstName>John</firstName> <lastName>Doe</lastName>  
3 </employee>  
4 <employee>  
5     <firstName>Anna</firstName> <lastName>Smith</lastName>  
6 </employee>  
7 <employee>  
8     <firstName>Peter</firstName> <lastName>Jones</lastName>  
9 </employee>  
10 </employees>
```

Rozdział 3

Założenia i specyfikacja aplikacji

TODO opisac apke w taki sam sposob jak opis na IO, zarowno frontend i backend,, waznaczy zadania aplikacji i jak serwer bleble

Uniwersalny dla calej aplikacji schemat procesu pobrania statycznych zasobow z serwera wykorzystuje metode get. Do zasobow tych nalezy miedzy innymi plik html zawiernajacy warstwe prezetacji aplikacji, plik css zawierajacy styl warstwy prezetacji czy plik zawiernajacy funkcje wykorzystywane przez aplikacje w jezyku javaScript. Proces prezetyuje sie nastepujaco:

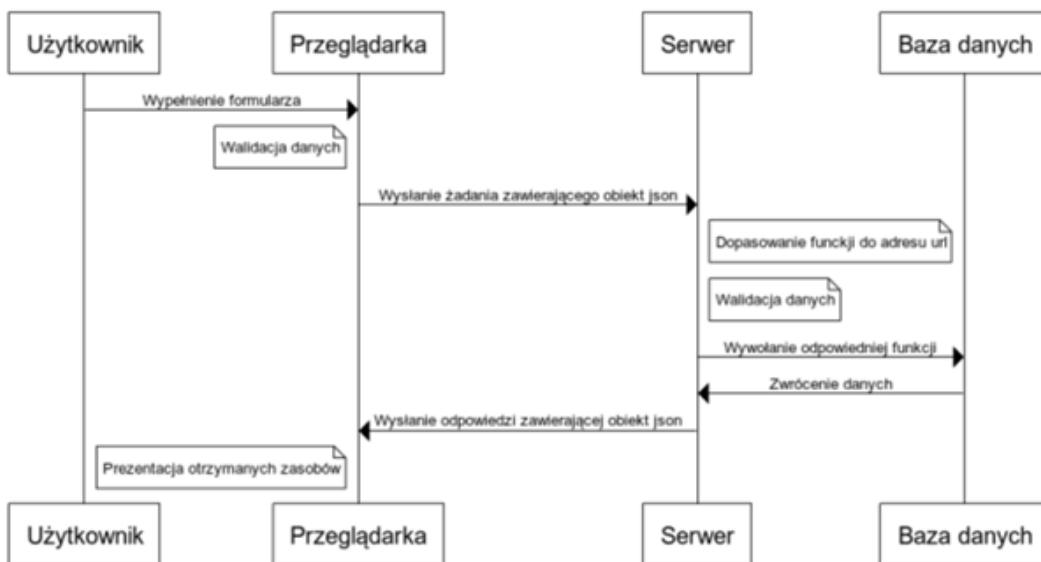


Rysunek 3.1: Pobranie statycznych zasobow źrodlo: Opracowanie wlasne

1. Klient przy uzytku przegladarki wysyla pod określony adres url serwera zapytanie zawiernajace porzadany zasob.
2. Serwer otrzymuje zapytanie i dopasowuje określona funkcje po adresie url serwera.

3. Funkcja wybiera dopasowane do zapytania zasoby i wysyla odpowiedz do aplikacji.
4. Aplikacja odbiera i zaczyna korzystac z zasobow.

Uniwersalny dla całej aplikacji schemat procesu wymiany danych określonych dla specyficznego użytkownika wykorzystuje metodę post. Przykładowe dane wymieniane przez aplikacje to login, hasło, informacje odbędące się zadanego użytkownika, kometarze do zadanego użytkownika. Przepływ danych wygląda następująco:



Rysunek 3.2: Wymiana zasobów w formacie JSON źródło: Opracowanie własne

1. Dane zostają pobrane od użytkownika w określonym formularzu w warstwie frontendowej.
2. Dane zostają zwalidowane pod kątem poprawności po stronie użytkownika.
3. Jeśli dane są poprawne zostają zorganizowane w obiekcie JSON i wysłane do serwera, w przeciwnym wypadku zostaje zwrocony błąd.
4. Serwer odbiera zapytanie i przekazuje je do odpowiedniej funkcji.
5. Dane zostają zwalidowane pod kątem poprawności po stronie serwera.

6. Jesli dane sa poprawne serwer realizuje określona funkcje przy pomocy zapytan baz danych, w przeciwnym wypadku zostaje przygotowany komunikat o błędzie.
7. Odpowiedz na zapytanie bazodanowe zostaje odebrane i zostaja przygotowane dane do wysłanie.
8. Przygotowane dane zostaje wyslane w określonej odpowiedzi.
9. Warstwa frontendowa otrzymuje odpowiedz i zostaje odsiwezona warsta prezentacji aplikacji.

Rozdział 4

Opracowanie aplikacji

4.1 Mean Stack

4.1.1 Wstęp

Do opracowania rozwiązania zdecydowałem się skorzystać z Mean Stack. Skrót odnosi się do frameworków oraz technologii Mongodb, Express, AngularJS oraz NodeJs. Współpracując razem zapewniają bardzo szybkie efektywne tworzenie skalowalnych aplikacji wielopratformowych. Do użycia wszystkich wymagany jest tylko jeden język programowania - javaScript, zarówno do obsługi warstwy front-endowej aplikacji jak i back-endowej. Wszystkie technologie są dostępne w pełni bezpłatnie.

Technologia Node.js została już opisana powyżej, więc zajmę się opisem pozostałych.

4.1.2 MongoDB

Napisany w języku c++ system zarządzania bazą danych zorientowanym na dokumenty. Operuje na nierelacyjnych bazach danych. Używa struktur Json jako schematów budowy bazy danych - więc udostępnia całkowitą dowolność w budowie struktury wymaganej bazy danych. Wykorzystuje proste, ale zapewniające szerokie możliwości zapytania baz danych.

4.1.3 ExpressJS

Framework służący do szybkiego wymagającego jak najmniejszych nakładów pracy wytwarzania zarówno back-end'u aplikacji internetowych jak i aplikacji mobilnych.

Dostarcza zbiór określonych klas i metod. Jest to najpopularniejszy framework do tworzenia serwerów w technologii Node.js.

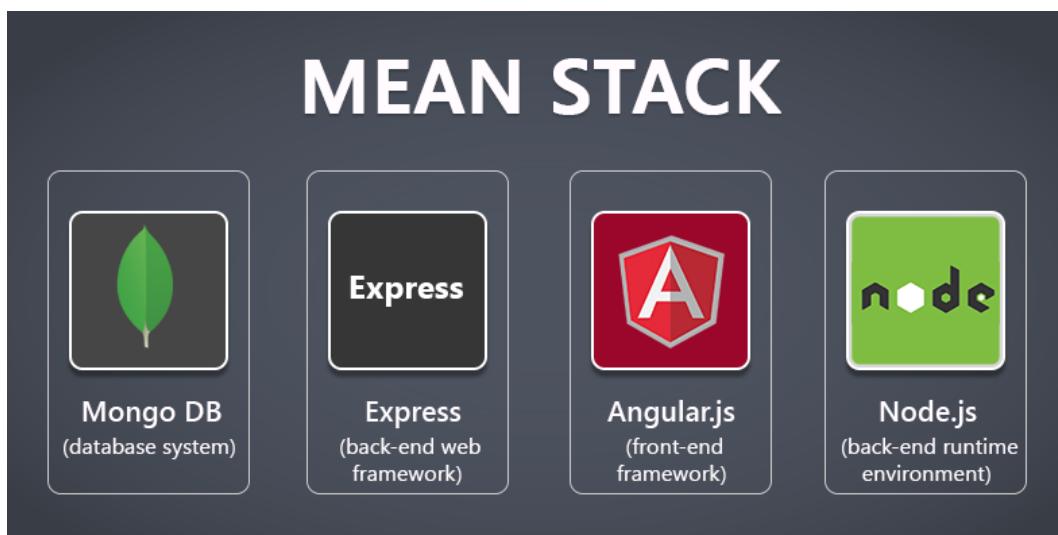
4.1.4 AngularJS

Wykorzystujący wzorzec projektowy MVC (Model View Controller) polegający na oddzieleniu od siebie poszczególnych warstw aplikacji - logiki, widoku oraz modelu komunikacji, framework wykorzystujący dodatkowe tagi w języku html w celu prostego w obsłudze i nie wymagającego dodatkowej logiki napisanej w języku javaScript tworzenia dynamicznych stron internetowych.

4.2 Komunikacja

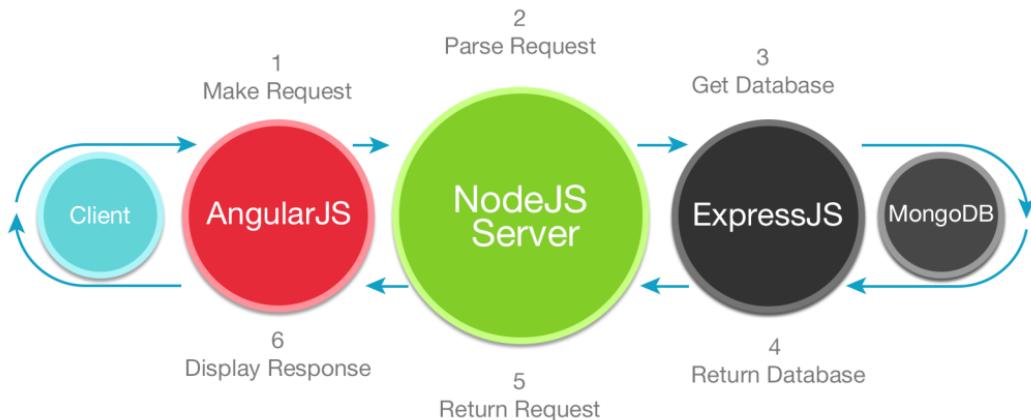
4.2.1 Rozwiązywanie problemu komunikacji

Do komunikacji między warstwą frontendową i backendową po stronie klienta wykorzystałem wysokopoziomową bibliotekę ajax oraz technologie Restful api. Ajax polega na zarządzaniu asynchroniczną komunikacją. Dzięki temu aplikacja może wykonywać inne funkcje mimo oczekiwania na odpowiedź ze strony serwera, za sprawą odseparowania warstwy wymiany danych od pozostałych warstw ap-



Rysunek 4.1: Przedstawienie technologii MEAN Stack - źródło: <http://codecondo.com/7-good-reasons-to-use-mean-stack-in-your-next-web-project/>

likacji. Do opisu wymienianych struktur użyłem standardu json, ponieważ wymaga on mniejszych nakładów pracy od standradu xml. Powyższe frameworki współpracują ze sobą w bardzo intuicyjny i przejrzysty sposób. Angular prezentuje dynamiczną aplikację internetową użytkownikowi, odpowiada za przyjmowanie danych i z pomocą ajax'a wysyła oraz odbiera dane wysypane do serwera. Serwer NodeJS z użyciem technologii express wykorzystując metodę Routingu dopasowuje zapytanie do odpowiednich funkcji serwisu, przetwarza otrzymane dane, przy współpracy z bazą danych zarządzaną przez Mongodb przechowuje informacje użytkowników serwisu oraz w odpowiedzi zwraca odpowiednie dane spowrotem do warstwy frontowej prezentującej dane użytkownikowi.



Rysunek 4.2: Przepływy komunikacji w MEAN Stack źródło: <https://www.dealfuel.com/seller/mean-stack-tutorial/>

4.2.2 Kod po stronie klienta

Przykładowy kod wykonywany po stronie klienta wysyłający zapytanie do serwera, żądające określonego zasobu przy użyciu frameworku Angularjs: Plik w języku javaScript zawierający definicje modelu i kontrolera:

```

1 var main = angular.module("main", []);
  // create angular module
2
3 main.controller('SignInController', function($scope) {
  // create controller for module
  
```

```

4  $scope.login="";           // inicialize login as field in
   model
5  $scope.password="";       // inicialize password as
   field in model
6  $scope.SignIn=function(){  // callback function as
   field in model
7  if ($scope.login && $scope.password) {      // if both
   field are filled
8  var xhr = new XMLHttpRequest();           //create request
   object
9  xhr.open("POST", http://site.com/authorization, true); // set
   method type, domain address and is it
   asynchronous
10 xhr.setRequestHeader("Content-type", "application/json");
   // set header with information of content type
11 req.send(JSON.stringify({ login: $scope.login,
   password: $scope.password})); // Send json
   object with login and password
12 req.onreadystatechange = function() { // create
   callback, called on every state change
13 if (req.readyState === 4) { // readyState 4 is 4
   when respond is got
14 if (req.status === 200) { // http status code
   200 - ok
15 var resObj = JSON.parse(this.responseText); // parse
   response text to json object
16 $scope.login=resObj.login; // set password from
   response
17 $scope.password=$resObj.password; // set password
   from response
18 alert("Login and password are valid"); // alert
   user
19 }
20 }
21 else if (req.status === 401){ // http status code
   200 - Unauthorized
22 $scope.invalid=req.responseText; // set response
   message containing error in view model
23 $scope.$apply(); // apply changes on view
   model

```

```

24         }
25     else {           // all other http statusses
26         alert(req.status+":req.responseText); // alert user
27     }
28 }
29 }
30 }
31 }
32 }

```

Plik w języku html zawierający definicje widoku:

```

1 <!DOCTYPE html>
2 <!-- Apply view to model -->
3 <body ng-app = "main">
4   <!-- Create form for request -->
5   <form class="main" name="form">
6     <table>
7       <tr>
8         <td style="width:50%;">
9           Login
10          </td>
11          <td style="width:50%">
12            <!-- Get user login -->
13            <input type = "text" ng-model = "login" name="login"
14              <!-- Set field validation -->
15              required ng-pattern='/[a-zA-Z0-9._-]+$/' ng-
16              maxlength="20" ng-minlength="3">
17          </td>
18        </tr>
19        <tr>
20          <td style="width:50%;">
21            Password
22          </td>
23          <td style="width:50%">
24            <!-- Get user password -->
25            <input type = "text" ng-model = "password" name="
26              password"
27              <!-- Set field validation -->
28              required ng-pattern='/[a-zA-Z0-9._-]+$/' ng-

```

```

27      maxlength="20" ng-minlength="3">
28      </td>
29  </tr>
30  <tr>
31      <td colspan="2">
32          <!-- show error message field from server only if
33              invalid field exist -->
34          <p ng-show="invalid">{{invalid}}</p>
35          <!-- show static error message only if one of user
36              field are invalid -->
37          <p ng-show="form.login.$invalid || form.password.
38              $invalid">Invalid login or password</p>
39      </td>
40  </tr>
41 </table>
42 </form>
43 </body>

```

4.2.3 Kod po stronie serwera

Przykładowy kod wykonywany po stronie serwera obsługujący odebrane zapytanie od klienta przy użyciu technologii Node.js oraz frameworków express i mongodb:

```

1 var express = require('express');           // load express module
2 var bodyParser = require('body-parser');    // load module for
   parsing json data
3 var DataBase = require('mongodb').MongoClient; // load MongoDB
   database module
4 var Promise = require('promise'); // load asynchronous returns
   from functions module
5 var DataBaseUrl = "mongodb://localhost:27017/db"; //dataBase
   address
6
7 var port = 8081;           //application port
8 var app = express();       // create express configuration object
9 app.use(bodyParser.json()); // load bodyParser to app object
10 app.use(bodyParser.urlencoded({ extended: true })); // for
   parsing application/x-www-form-urlencoded
11

```

```

12 // connect to database function, returns database handler
13 function Connect() {
14     return new Promise(function (fulfill, reject) {
15         DataBase.connect(dataBaseUrl, function(err, db) {
16             if (err) {
17                 reject(err);
18             }
19             fulfill(db);
20         });
21     });
22 }
23
24 // check if database contains given user
25 var Authorization=function (login, password) {
26     return new Promise(function (fulfill, reject) {
27         Connect().done(function (db) {
28             db.collection("Users").findOne({ login: login,
29                 password: password }, function (err, result) {
30                 if (err) {
31                     reject(err);
32                 } else {
33                     if (result != null) {
34                         fulfill(true);
35                     } else {
36                         fulfill(false);
37                     }
38                 }
39                 db.close();
40             });
41         });
42     });
43 }
44
45 // check is text format correct
46 function TextValidation(text, min, max) {
47     if (min === undefined) min = 0;
48     if (max === undefined) max = Infinity;

```

```

49     return (text !== undefined) && (/^[\w\W]{3,20}$/.test(
50         text) && text.length >= min && text.length <= max;
51     }
52 // check is user credentials data correct
53 function UserValidation(body) {
54     return new Promise(function(fulfill, reject) {
55         if (!TextValidation(body.login, 3, 20) || !
56             TextValidation(body.password, 3, 20)) {
57             fulfill(false);
58             return;
59         }
60         Authorization(body.login, body.password).done(function (
61             authentication) {
62             fulfill(authentication);
63         });
64     });
65 //function run when server gets post request with url /
66 // authorization
66 app.post('/authorization', function (req, res) {
67     UserValidation(req.body).done(function(valid) {
68         if (valid) {
69             res.setHeader('Content-Type', "text/html");
70             res.statusCode = 200;
71             res.end();
72         }
73         else {
74             res.setHeader('Content-Type', "text/html");
75             res.statusCode = 401; // error has happened
76             res.write("Invalid login or password");
77             res.end();
78         }
79     });
80 });
81
82 //start server
83 var server = app.listen(port, function () {

```

```
84  MakeLog('Server listening');
85});
```

4.3 Wykorzystane moduły

4.3.1 Moduły zewnętrzne

Zewnętrzne moduły, zainstalowane za pomocą zbioru repozytorium npm które użyłem do realizacji rozwiązania to:

1. express - opisany powyżej
2. fs - file stream, odpowiada za obsługę zapisu oraz wczytywania plików. Wykorzystywany w celu wczytywania plików statycznych aplikacji takich jak index.htm (strona główna w formacie html), main.js (funkcje w języku javaScript), style.css (plik arkuszu stylów, odpowiadający za styl prezentacji), favicon (ikona serwisu wyświetlna w oknie przeglądarki), czy losowo wybieranych zdjęć wyświetlanych w tle serwisu.
3. path - obsługa różnic między systemami operacyjnymi. Dzięki temu modułowi możemy zapewnić całkowitą sprawność naszego serwera nie zależnie od środowiska uruchomieniowego. Pozwala niwelować różnice w lokalizacji plików systemowych, czy łącznikach pomiędzy poszczególnymi folderami przy specyfikacji ścieżki do pliku.
4. body-parser - dostarcza możliwość analizowania danych załączonych do odesłanego przez serwer żądania, wykorzystany został w celu odczytów poszczególnych wartości odebranych w formacie json.
5. promise - odpowiada za operowanie wynikami otrzymanymi w wyniku asynchronicznych funkcji. Dzięki temu modułowi nie musimy synchronicznie czekać na otrzymanie zwrotu z kolejnych funkcji, natomiast możemy przy użyciu wywołań zwartnych zareagować po otrzymaniu określonego wyjścia.
6. cookie-parser - zapewnia możliwość wykorzystania plików cookie dla specyficznego klienta. Dzięki temu modułowi możemy operować, ustawiać, usuwać lub edytować wartości plików cookie, które zostaną przydzielone dla konkretnego użytkownika w ramach całej sesji komunikacji z użytkownikiem lub przez określony przez nasz czas.

4.3.2 Moduły wewnętrzne

Wewnętrzne moduły, czyli takie które zostały napisane przeze mnie specjalnie na użytku projektu to:

1. emails.js (wykorzystujący zewnętrzny moduł mongodb) - moduł zapewnia komunikacje z bazą danych w technologii mongodb, poprzez wywoływanie odpowiednich funkcji. Został dostosowany bezpośrednio do obsługi bazy danych dla wykonywanego projektu. Dzięki temu plik główny serwera nie musi znać budowy, ani wykonywać operacji bezpośrednio na dokumentach bazy danych.
2. db.js (wykorzystujący zewnętrzny moduł nodemailer) - dostarcza obsługę wysyłania wiadomości email do użytkowników serwisu w przypadku znięcia określonych sytuacji takich jak na przykład otrzymanie nowego zapytania czy zmiana statusu zadania. Moduł dostarcza jedną, prostą w obsłudze funkcje zapewniającą obsługę wiadomości emial.

4.4 Diagramy funkcji

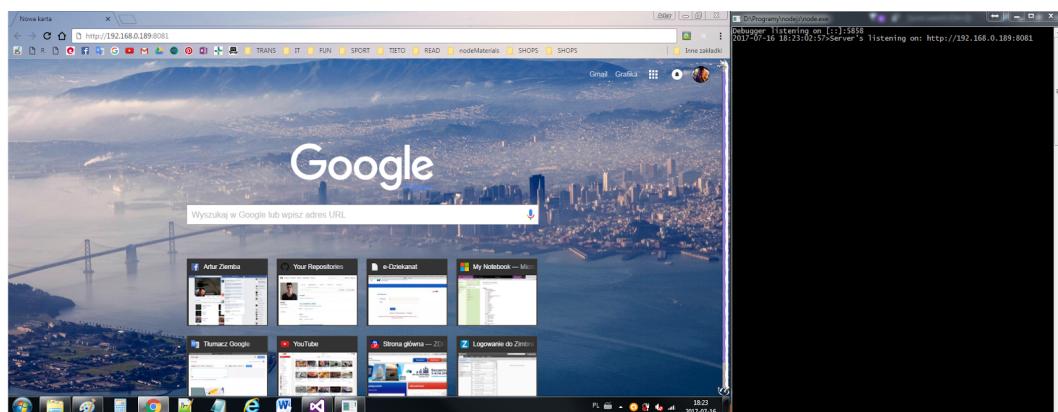
TODO

Rozdział 5

Testy aplikacji

Po wykonaniu aplikacji zostały przeprowadzone testy manualne sprawdzające poprawność dostarczanej przez serwis funkcjonalności. Kolejne przypadki testowe:

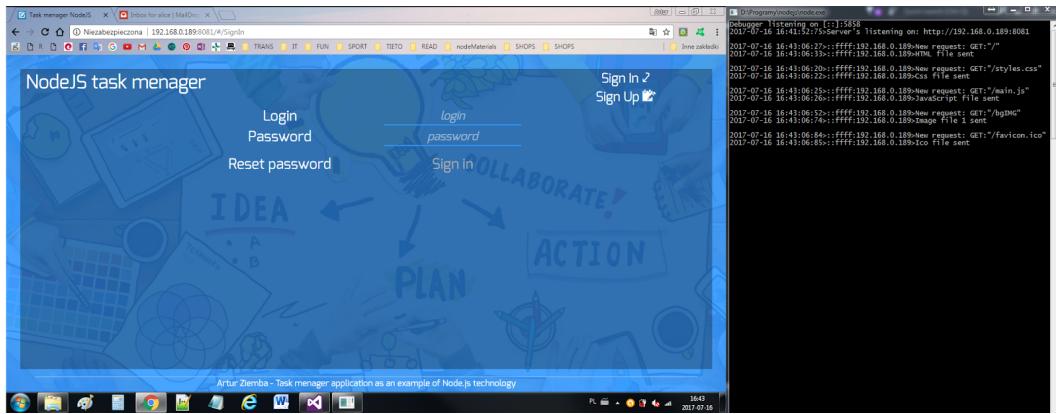
5.1 Uzyskanie dostępu do statycznych zasobów serwisu



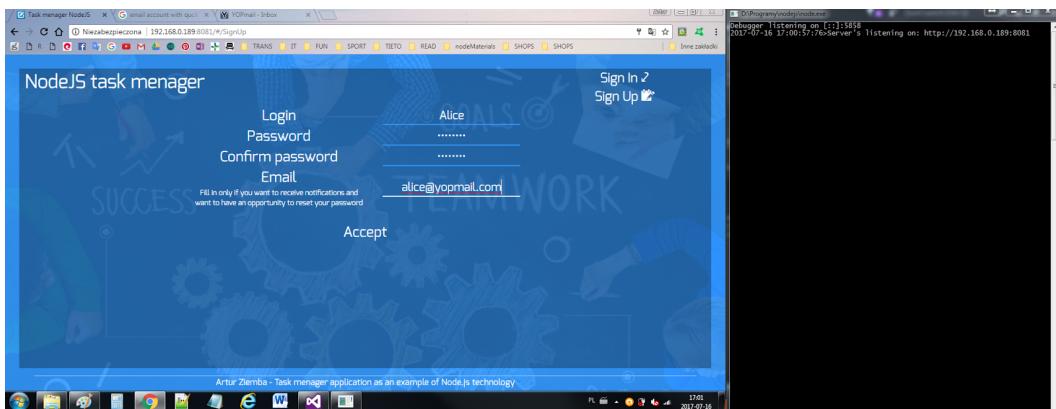
Użytkownik wysyła żądanie pod adres serwera.

5.2 Rejestracja w serwisie

1. Użytkownik wypełnia formularz rejestracyjny na stronie serwisu i wysyła dane do serwera.



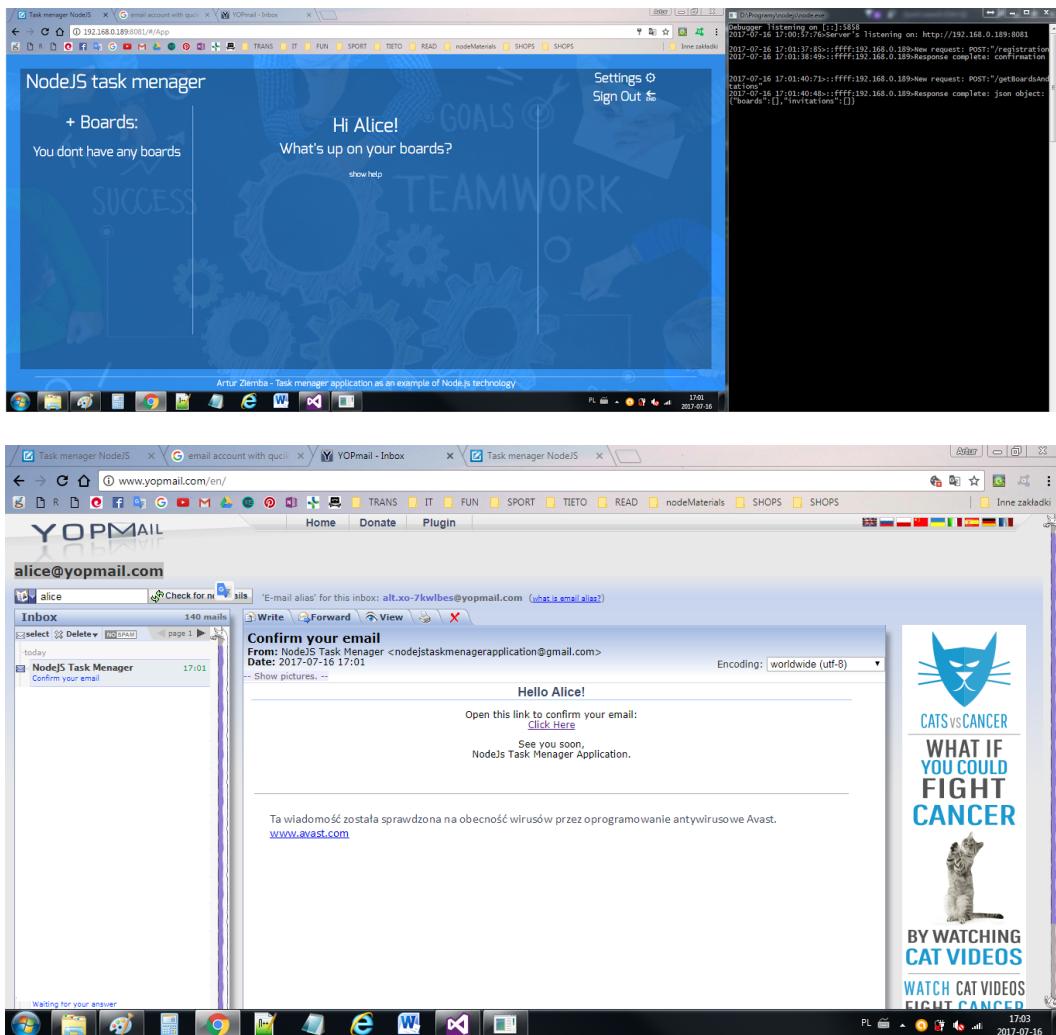
Użytkownik otrzymuje odpowiedź zawierającą pliki statyczne serwisu (plik html, css, js, zdjęcie w tle i favicon).



2. Serwer sprawdza poprawność otrzymanych danych i w odpowiedzi przesyła stronę główną aplikacji dostępna po zalogowaniu na konto lub informację o błędzie.

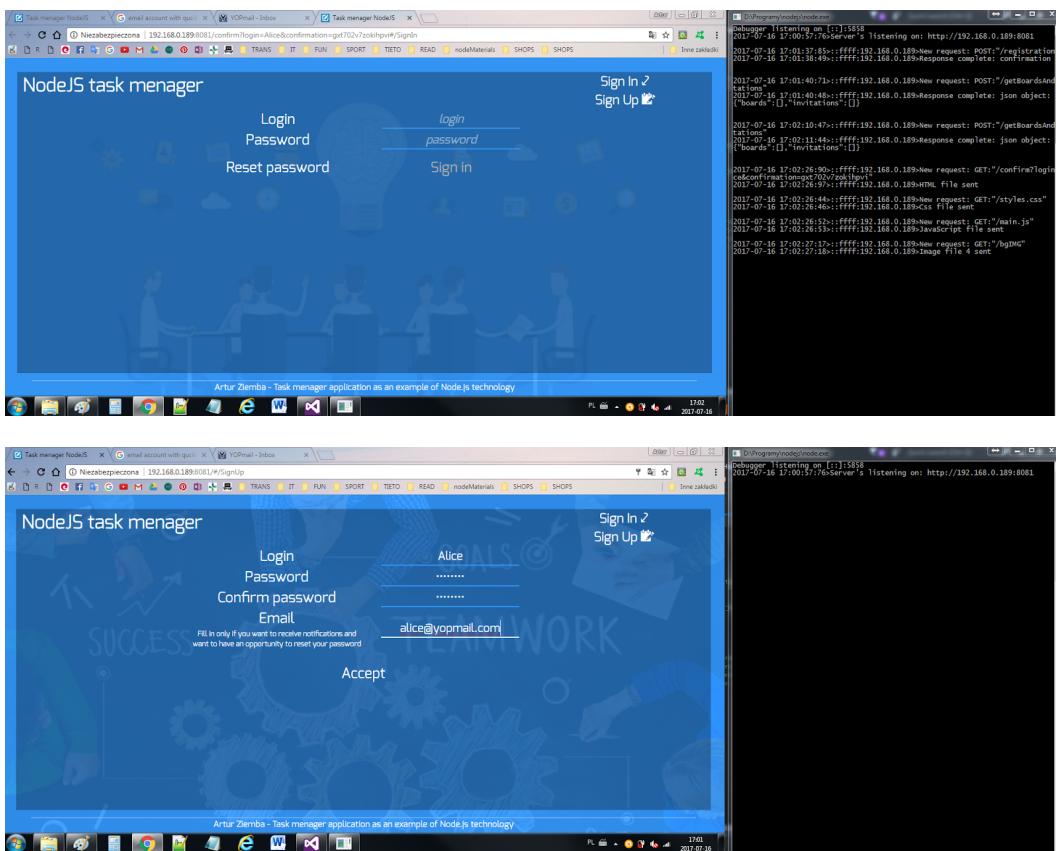
5.3 Potwierdzenie adresu email

1. Po podaniu adresu email użytkownik otrzymuje wiadomość email zawierającą link aktywacyjny.
2. Po kliknięciu w link, serwer sprawdza czy otrzymany w linku kod jest prawidłowy dla podanego użytkownika. Następnie aktywuje adres email dla odpowiedniego użytkownika i wysyła w odpowiedzi stronę główną aplikacji oraz wiadomość email o potwierdzeniu.



5.4 Logowanie się w serwisie

1. Użytkownik wypełnia formularz na stronie serwisu i przesyła żądanie do serwera.
2. Serwer sprawdza istnienie konta w serwisie. W odpowiedzi użytkownik otrzymuje stronę główną aplikacji lub błąd o niepoprawnych danych. W czasie bycia zalogowanym co określony czas zostaje wysłane żądanie aktualizacji danych. Aktualizacja następuje również każdorazowo po otrzymaniu pomyślnego potwierdzenia wykonania operacji przez serwer.

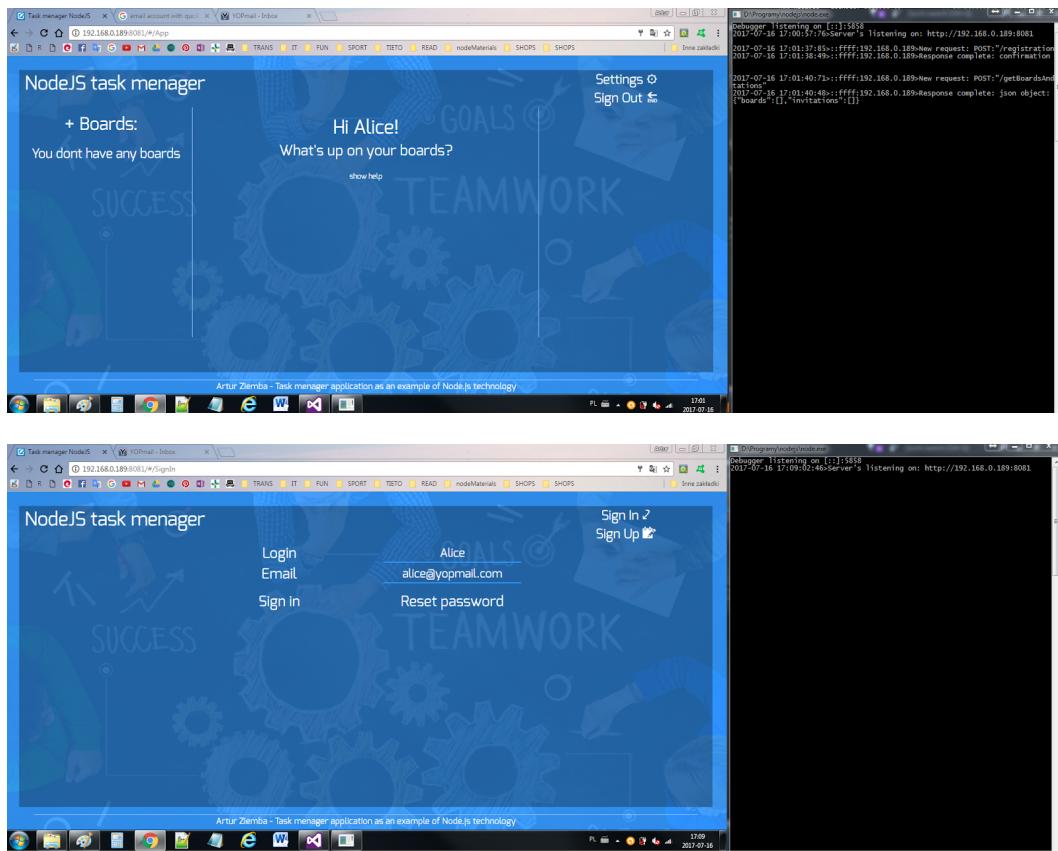


5.5 Resetowanie hasła użytkownika

1. Funkcja jest dostępna po potwierdzeniu adresu email dla konta. Po wypełnieniu odpowiedniego formularza na stronie głównej serwisu użytkownik wysyła żądanie do serwera.
2. Serwer sprawdza podane dane. W odpowiedzi wysyła informacje o przetworzeniu żądania i wiadomość email zawierającą nowe hasło do serwisu.

5.6 Zmiana danych użytkownika

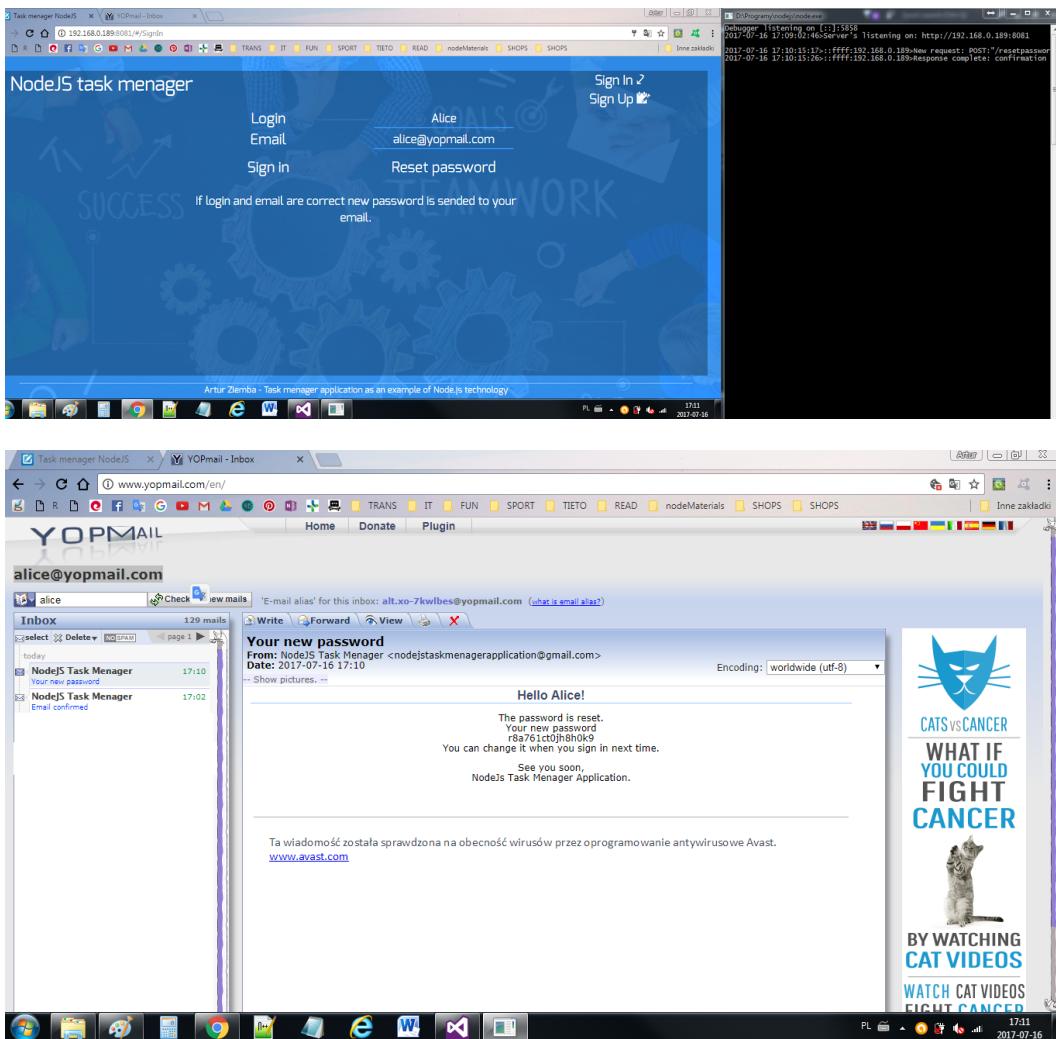
1. Po zalogowaniu się i wypełnieniu odpowiedniego formularza na stronie serwisu użytkownik wysyła żądanie do serwera.
2. Serwer sprawdza poprawność danych. Jeśli są poprawne aktualizuje je. W przeciwnym wypadku zwraca komunikat o błędzie. Jeśli zmieni się również



adres email użytkownik musi go ponownie zweryfikowac. W odpowiedzi serwer wysyła stronę główną aplikacji.

5.7 Utworzenie nowej tablicy z zadaniami

1. Po zalogowaniu się użytkownik wybiera opcje dodania tablicy, wpisuje wymagane dane i wysyła je do serwera.
2. Serwer sprawdza poprawność danych. Jeśli są poprawne serwer tworzy nową tablicę dla użytkownika i wysyła potwierdzenie w odpowiedzi. W przeciwnym wypadku odpowiada komunikatem o błędzie.



5.8 Wysyłanie zaproszenia do tablicy

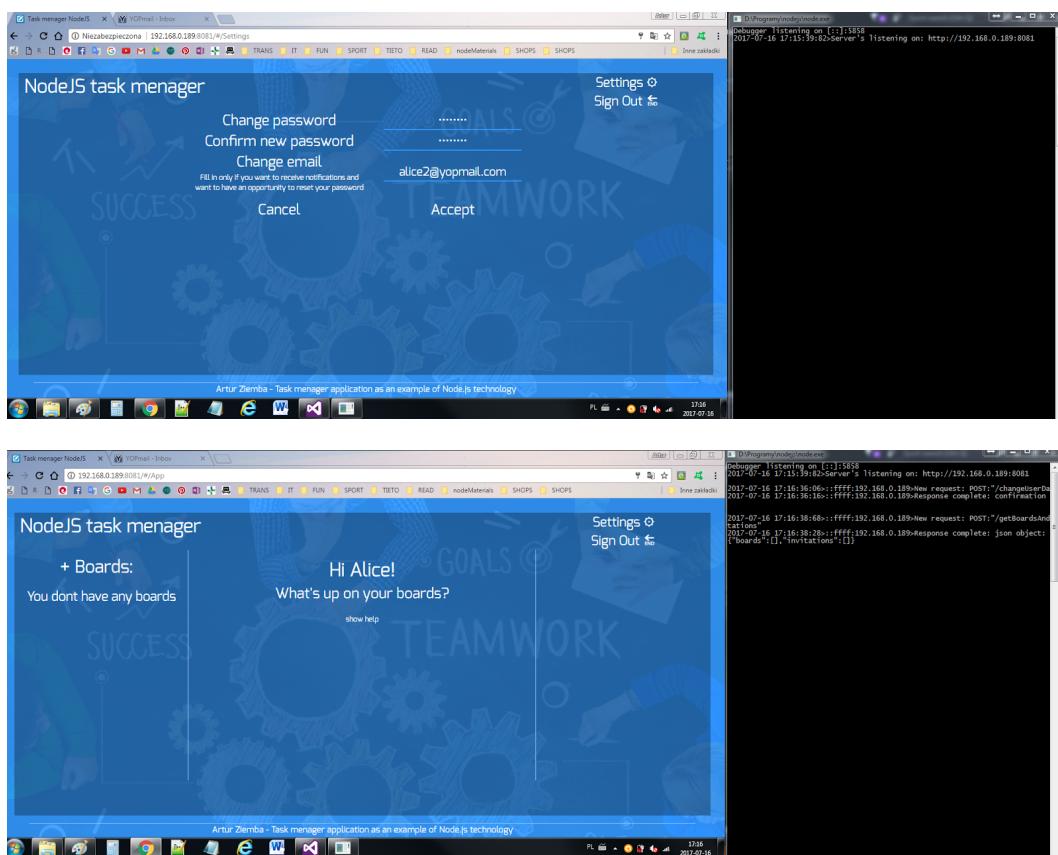
1. Po zalogowaniu się użytkownik wybiera opcje wysłania zaproszenia do wybranej tablicy, wypełnia dane i wysyła żądanie do serwera.
2. Serwer sprawdza poparwość danych. W odpowiedzi wysyła potwierdzenie o wysłanym zaproszeniu, lub komunikat o błędzie.

5.9 Akceptacja zaproszenia

1. Po zalogowaniu się użytkownik wybiera opcje zaakceptowania oczekującego zaproszenia i wysyła żądanu do serwera.
2. Serwer sprawdza poprawność danych. Odpowiada potwierdzeniem lub komunikatem o błędzie.

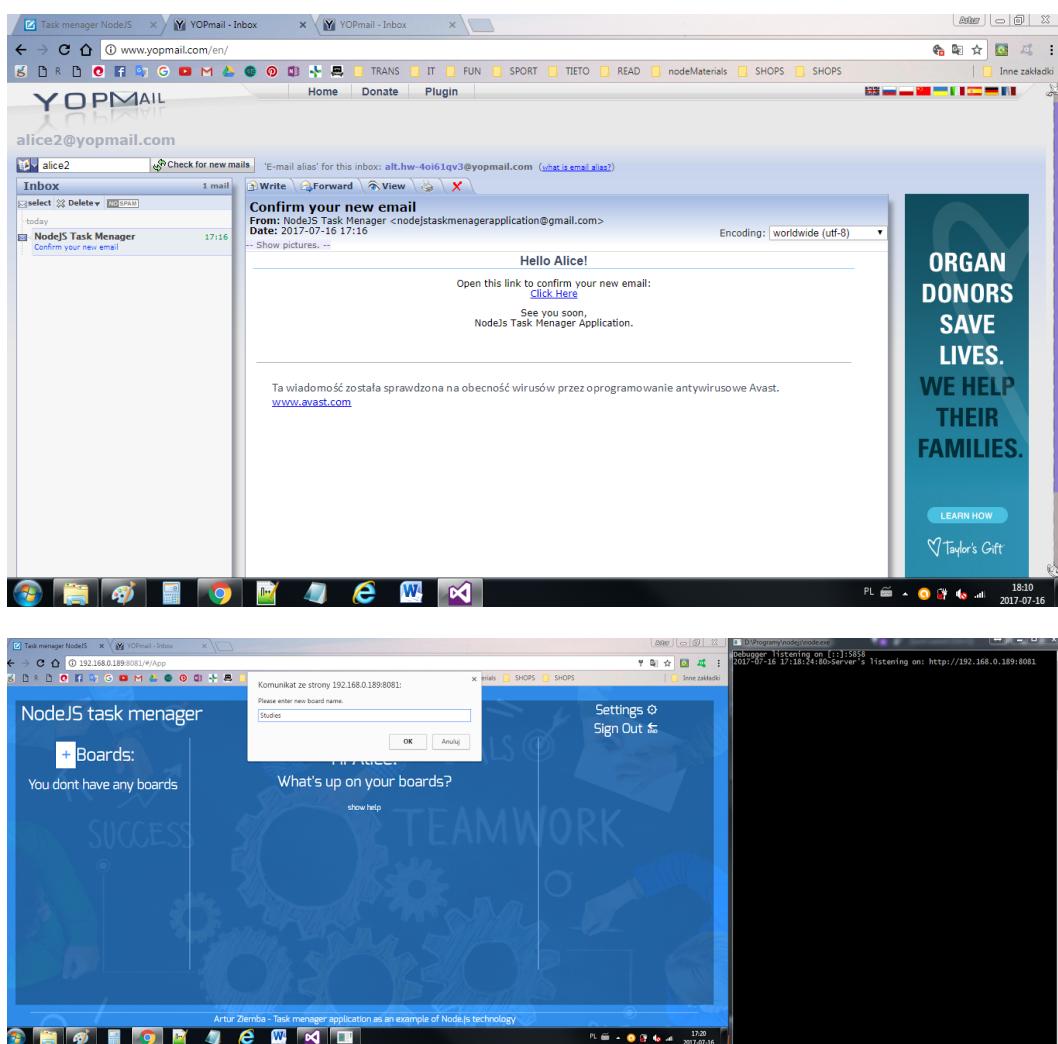
5.10 Odrzucenie zaproszenia

1. Po zalogowaniu się użytkownik wybiera opcje odrzucenia oczekującego zaproszenia i wysyła żądanu do serwera.
2. Serwer sprawdza poprawność danych. Odpowiada potwierdzeniem lub komunikatem o błędzie.



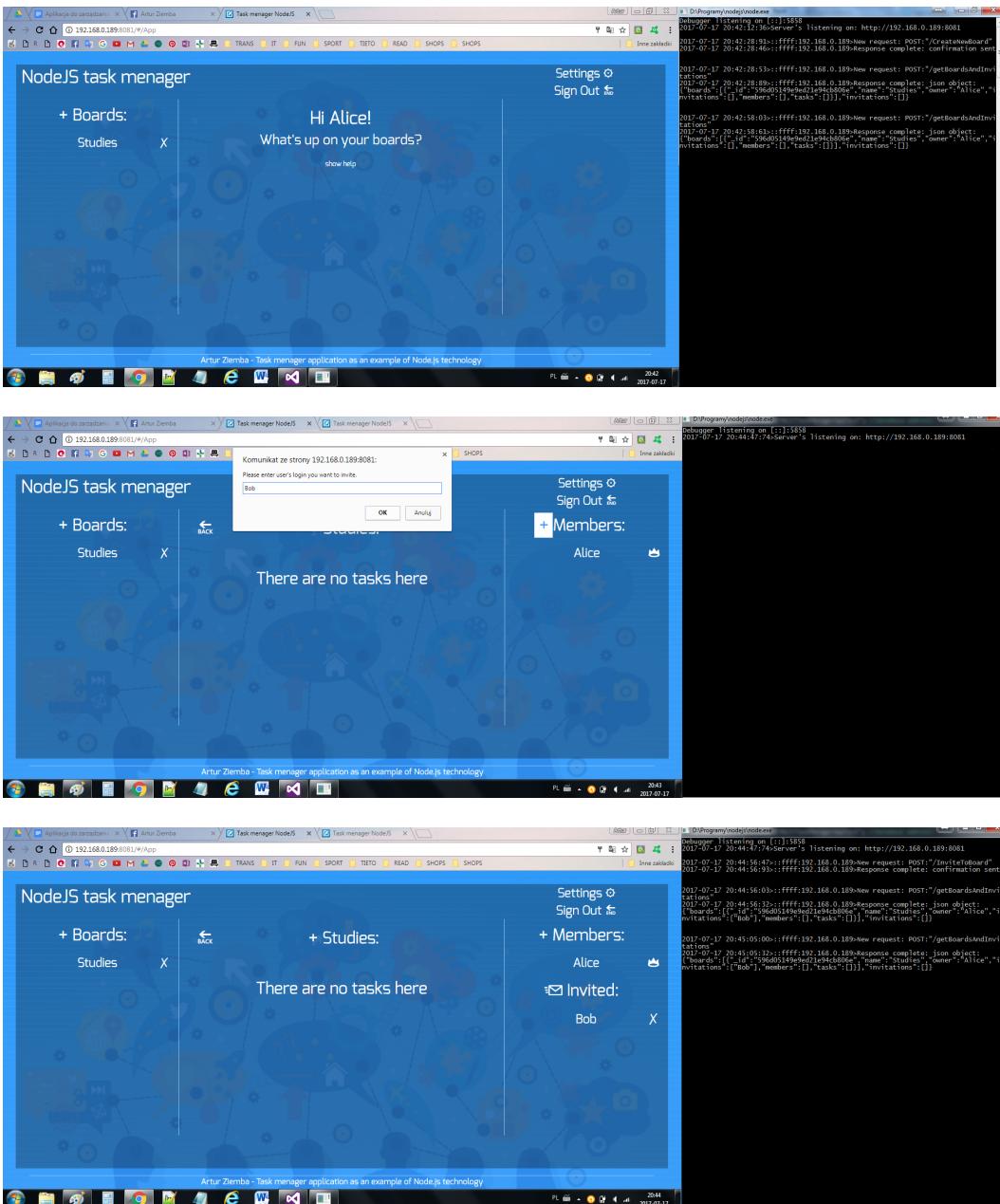
5.11 Wyrzucenie użytkownika z tablicy

1. Po zalogowaniu się użytkownik wybiera opcje wyrzucenia użytkownika z określonej tablicy i wysyła żądanie do serwera.
2. Serwer sprawdza poprawność danych. Odpowiada potwierdzeniem lub komunikatem o błędzie.

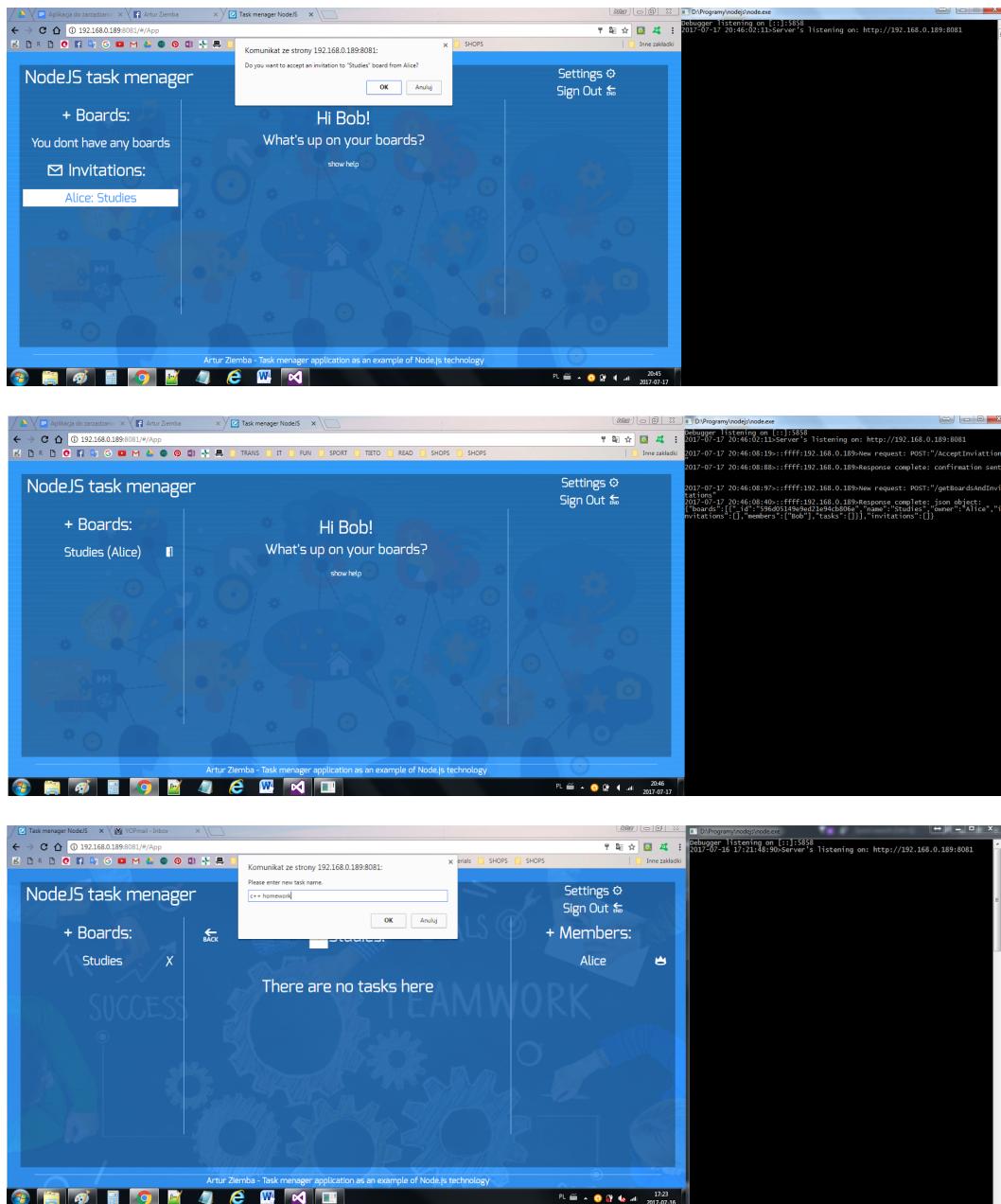


5.12 Dodanie zadania

1. Po zalogowaniu i wybraniu odpowiedniej tablicy użytkownik wybiera opcje dodania zadania, wypełnia dane i przesyła żądanie do serwera.

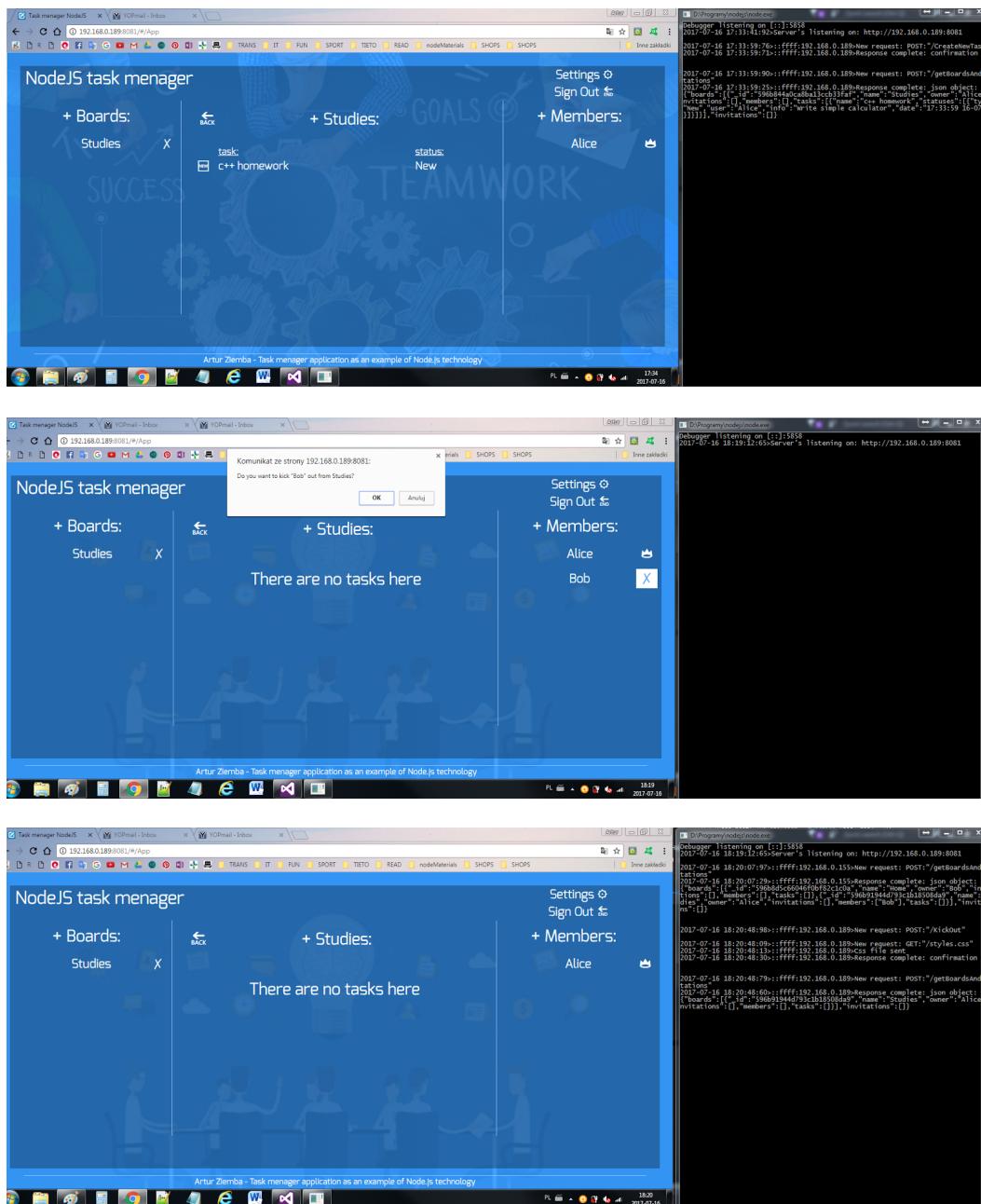


2. Serwer sprawdza poprawność danych. Odpowiada potwierdzeniem lub komunikatem o błędzie.

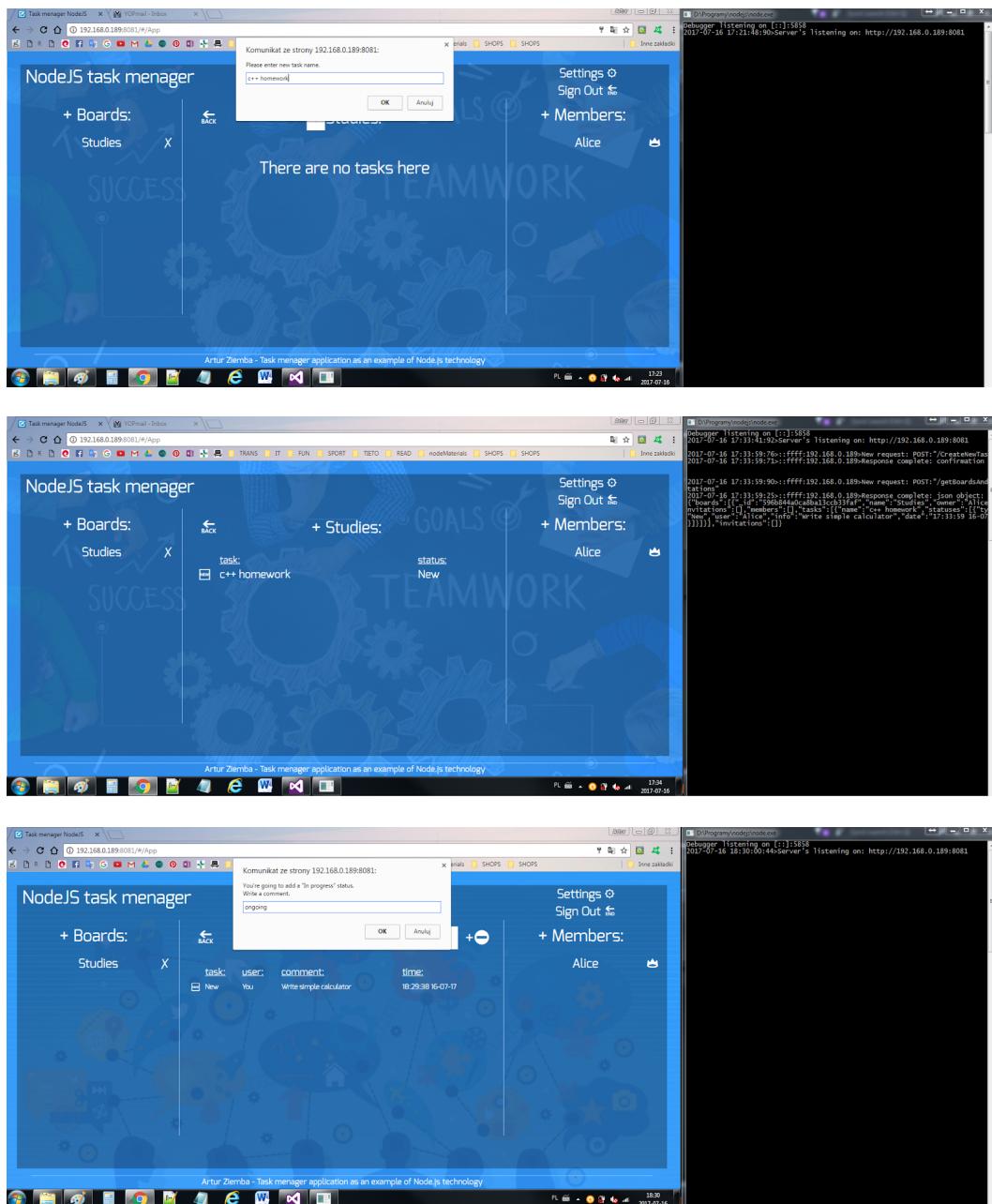


5.13 Dodanie statusu zadania

1. Po zalogowaniu, wybraniu odpowiedniej tablicy i zadania użytkownik wybiera opcje dodania nowego statusu, uzupełnia dane i przesyła żądanie do serwera.



2. Serwer sprawdza poprawność danych. Odpowiada potwierdzeniem lub komunikatem o błędzie.

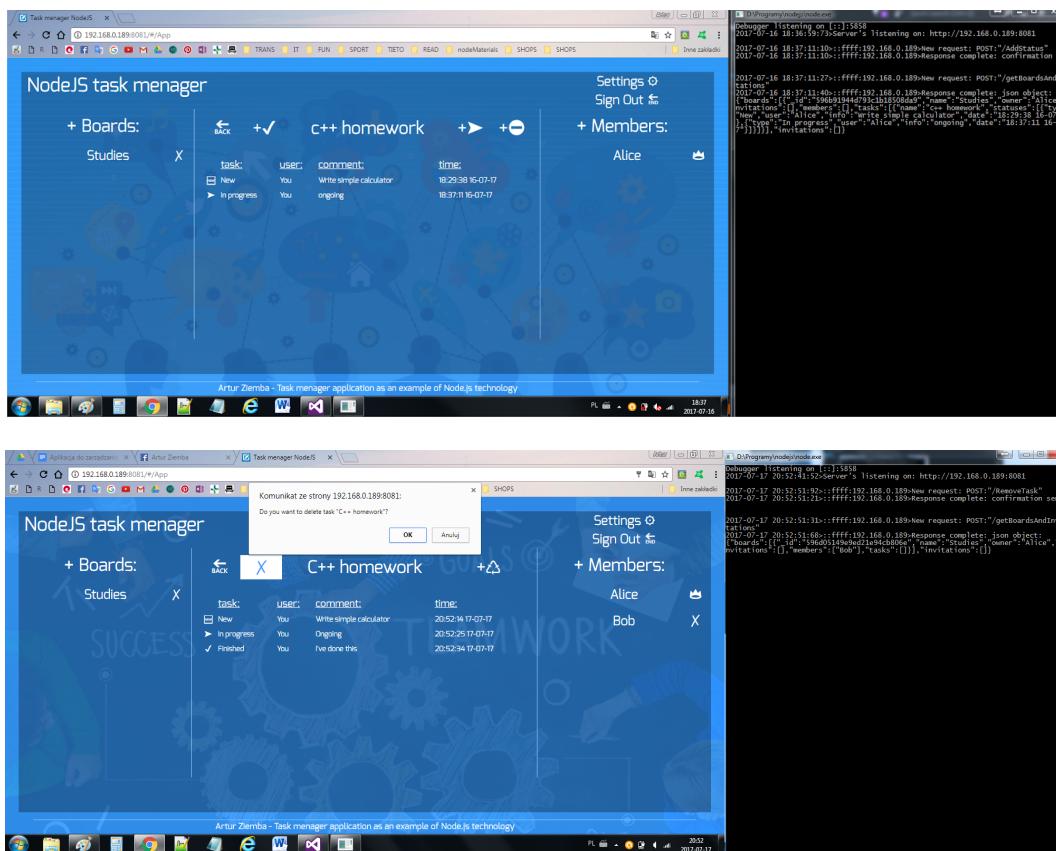


5.14 Usuwanie zadania

1. Po zalogowaniu, wybraniu odpowiedniej tablicy i zakończonego zadania użytkownik wybiera opcje usuwania zadania i przesyła żądanie do serwera.
 2. Serwer sprawdza poprawność danych. Odpowiada potwierdzeniem lub komunikatem o błędzie.

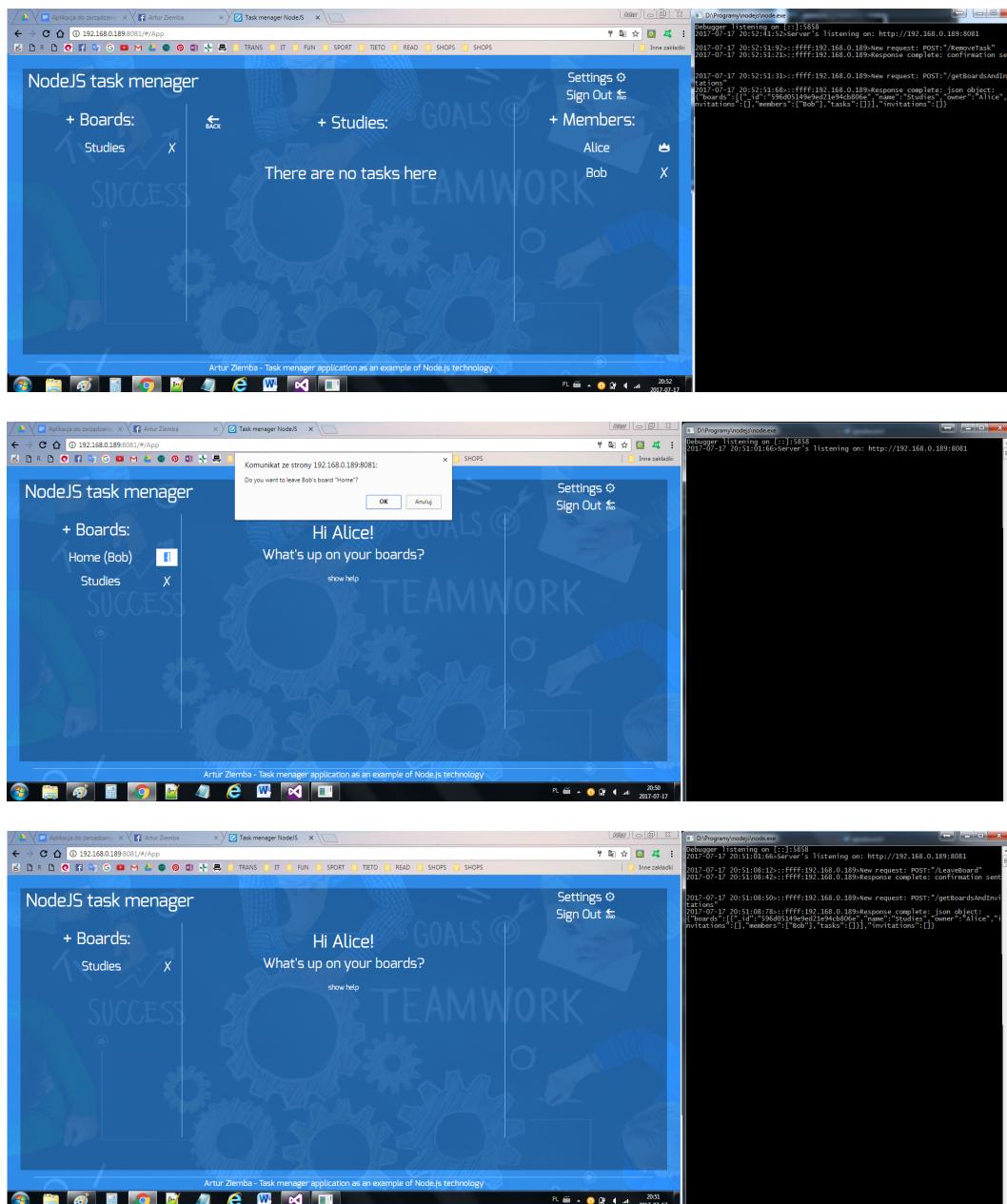
5.15 Opuszczanie tablicy

1. Po zalogowaniu użytkownik wybiera opcje opuszczenia tablicy, której jest członkiem i wysyła żądanie do serwera.
 2. Serwer sprawdza poprawność danych. Odpowiada potwierdzeniem lub komunikatem o błędzie.

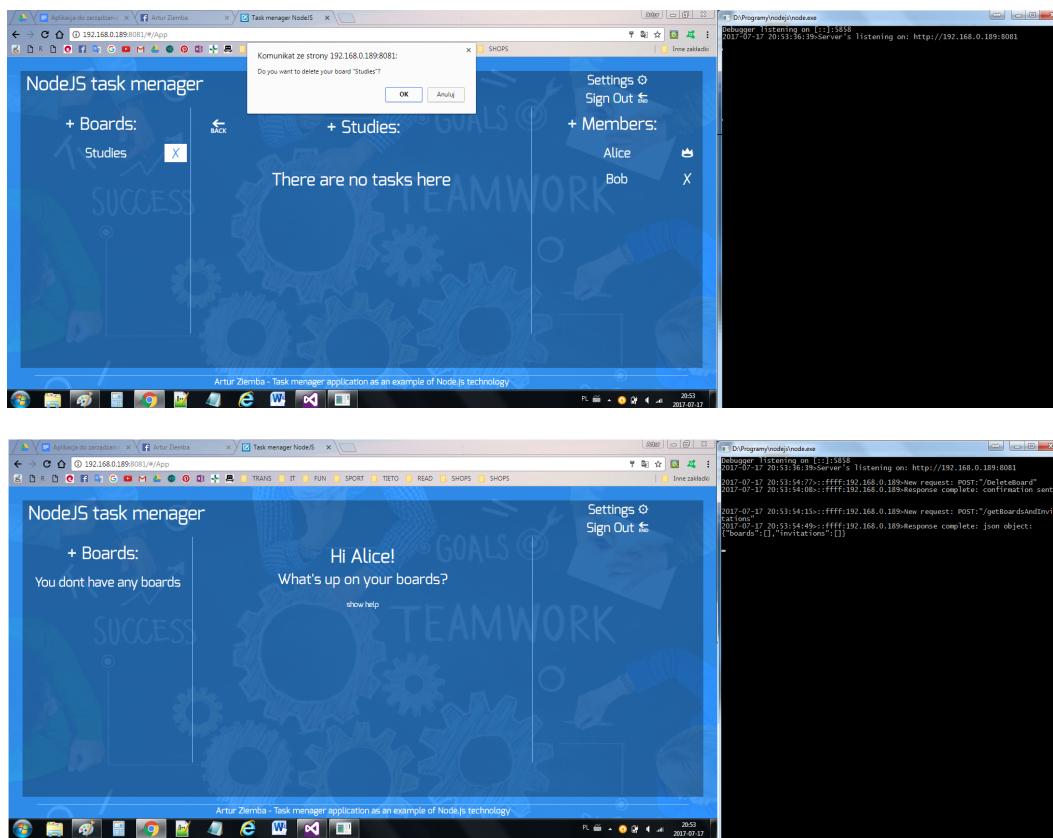


5.16 Usuwanie tablicy

1. Po zalogowaniu użytkownik wybiera opcje usunięcia własnej tablicy i wysyła żądanie do serwera.



2. Serwer sprawdza poprawność danych. Odpowiada potwierdzeniem lub komunikatem o błędzie.



Rozdział 6

Podsumowanie otrzymanych wyników i wnioski na temat środowiska

Wykoanana aplikacja spełnia zadana w warunkach funkcjonalnosc. Zostały osiągnięte wszystkie początkowe założenia oraz wymagania aplikacji przy użyciu możliwości dostarczanej przez technologie Node.js. Stworzony projekt doskonale nadaje się do użytkownia przez zorganizowane zespoły do wykonywania określonych zadań. Móglby być na przykład używany do zarządzania pracą zespołów deweloperskich pracujących w różnych metodykach takich jak scrum czy kanban. Dostarcza przejzysty interfejs oraz nieskomplikowaną prezetację danych dla końcowych użytkowników, przez co nie wymaga praktycznie żadnych szkoleń lub procesów wdrożenioowych w celu zdobycia umiejętności biegłej obsługi narzędzia. Środowisko Node.js wydaje się być bardzo nowoczesnym oraz mającym przed sobą świetlana przyszłość środowiskiem. Niewątpliwie największymi zaletami tej technologii jest łatwość budowania wymagających serwisów internetowych poprzez użycie asynchronicznej obsługi wejścia/wyjścia, które pozwala na przetwarzanie wielu funkcji w jednym czasie tworząc wyjątkowo szybkie w działaniu aplikacje. Zagadnienie to wymaga jednak umiejętności w projektowaniu i analizowaniu funkcji programowania asynchronicznego. Kod pisany jest w szeroko używanym języku javaScript co zbudowało ogromną społeczność zainteresowaną technologią Node.js. Dzięki temu możemy z łatwością zdobyć materiały przydatne w procesie poznawczym środowiska. Globalne repozytorium npm gwarantuje obsługę wielu funkcjonalności, bez potrzeby długiego szukania możliwych rozwiązań. Wszystko jest dostępne bezpłatnie, gotowe do wdrożenia w tworzonej aplikacji. Mimo że jest to technologia stosunkowo młoda obecnie Node.js jest używany, a co za tym idzie sprawdzony przez największe

firmy IT w celu obsługi ich aplikacji i serwerów. Należy być świadomym wszystkich zalet i wad przy wyborze określonej technologii. Node.js nie nadaje się niestety do każdego typu projektu. Nie jest efektywnym środowiskiem jeśli chodzi o korzystanie z obliczeń intensywnie wykorzystujących procesor. Jest on jednak że idealnym rozwiązaniem w przypadku pracy nad wieloma rozwiązaniami webowymi. Osobiście praca w Node.js sprawiła mi wielką radość i stała się moją ulubioną technologią dzięki zapewnionej prostocie i możliwych do uzyskania bez większego wysiłku ogromnych możliwościach.

Bibliografia

- [1] Ethan Brown *Web Development with Node and Express: Leveraging the JavaScript Stack 1st Edition* ISBN: 9781491949306
- [2] Robert Onodi *MEAN Blueprints* ISBN: 9781783553945
- [3] Dokumentacja języka programowania Node.js. Stan na dzień: 2017-07-20
<https://nodejs.org/en/docs>

Spis rysunków

2.1	Pętla zdarzeń w środowisku Node.js - źródło: http://galaxy.agh.edu.pl	7
2.2	Model programowania współbieżnego wykorzystywany przez Node.js - źródło: www.tutorialspoint.com/parallel_algorithm/	9
2.3	Kody statusu protokołu http - źródło: http://mokandra.blogspot.fi/2015/10/http-status-code-definitions.html	10
3.1	Pobranie statycznych zasobów źródło: Opracowanie własne	12
3.2	Wymiana zasobów w formacie Json źródło: Opracowanie własne	13
4.1	Przedstawienie technologii MEAN Stack - źródło: http://codecondo.com/7-good-reasons-to-use-mean-stack-in-your-next-web-project/	16
4.2	Przepływ komunikacji w MEAN Stack źródło: https://www.dealfuel.com/seller/mean-stack-tutorial/	17