Akademia Górniczo-Hutnicza

Wzorce projektowe

Serwer piggy-back z zastosowaniem AngularJS i NodeJS

Ewa Hechsman
Agata Sidło
Katarzyna Wilczak
Laura Żuchowska

2018/2019

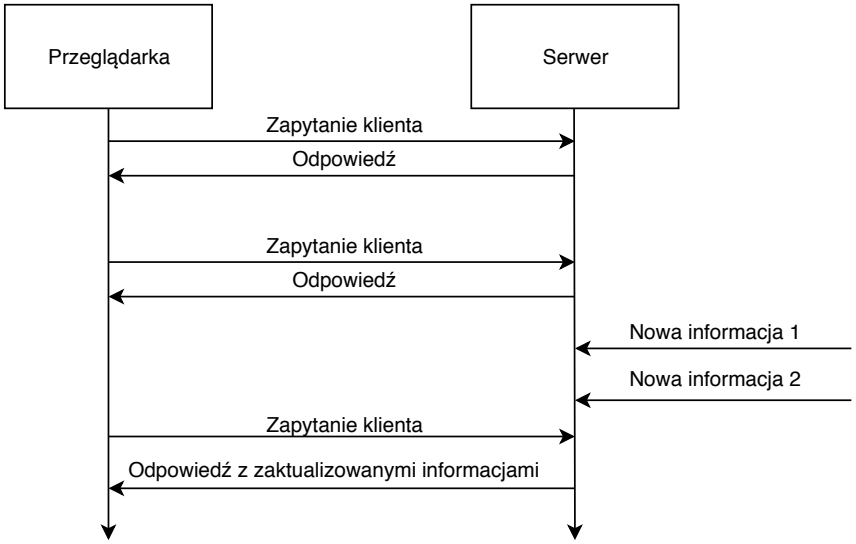
Spis treści

1 Ogólny opis projektu

Celem projektu jest dostarczenie biblioteki implementującej mechanizm server-push z wykorzystaniem techniki piggy-back. Technika ta polega na wysłaniu przez klienta zapytania do serwera, który od razu odpowiada. Przy ponownym zapytaniu wysłanym przez tego samego klienta, serwer wraz z odpowiedzią wysyła informacje, które pojawiły się od chwili ostatniego zapytania.

Opis metody piggyback

Metoda piggyback jest jedną z metod wykorzystujących *Reverse AJAX*. Zalicza się do nich m.in. metody 'Comet', 'Polling', 'Long Polling'. Polega na odsyłaniu przez serwer odpowiedzi na zapytanie klienta wraz z dodatkowymi informacjami/eventami pojawiającymi się pomiędzy kolejnymi zapytaniami tzw. 'na doczepkę'. Działanie metody piggyback przedstawiono na diagramie:



Rysunek 1: Schemat metody piggy-back

Zalety:

- pozwala na usunięcie zbędnych zapytań klienckich (takich, które nie wymagają zwracania żadnych danych)
- wprowadzana jest mała ilość zmian w aplikacji klienckiej (w porównaniu z innymi metodami typu *Reverse AJAX*, np. *Polling*

Wady:

- może upłynąć dużo czasu zanim pewna zmiana na serwerze będzie widoczna dla klientów, np. klient czyta blog, więc przez dłuższy okres nie wysyła żadnych żądań, więc serwer nie może odesłać mu nowych informacji

2 Wykorzystane technologie

- **AngularJS** - jest to framework wykorzystywany głównie przy tworzeniu aplikacji typu Single Page Applications, oparty na języku programowania JavaScript
- **NodeJS** - środowisko do tworzenia aplikacji internetowych, w którym może być uruchamiana zarówno strona serwera, jak i aplikacja kliencka

Wykorzystane biblioteki

- **express** - pozwala na modyfikację otrzymanego zapytania przed wysłaniem odpowiedzi
- **fs** (file system) - wykorzystany w celu zapisu oraz odczytu pliku przechowującego informacje dotyczące wysyłanych przez klientów eventów
- **events** - potrzebny do emitowania eventów do testowania
- **bodyParser** - potrzebny do parsowania requestu klienta

3 Opis implementacji

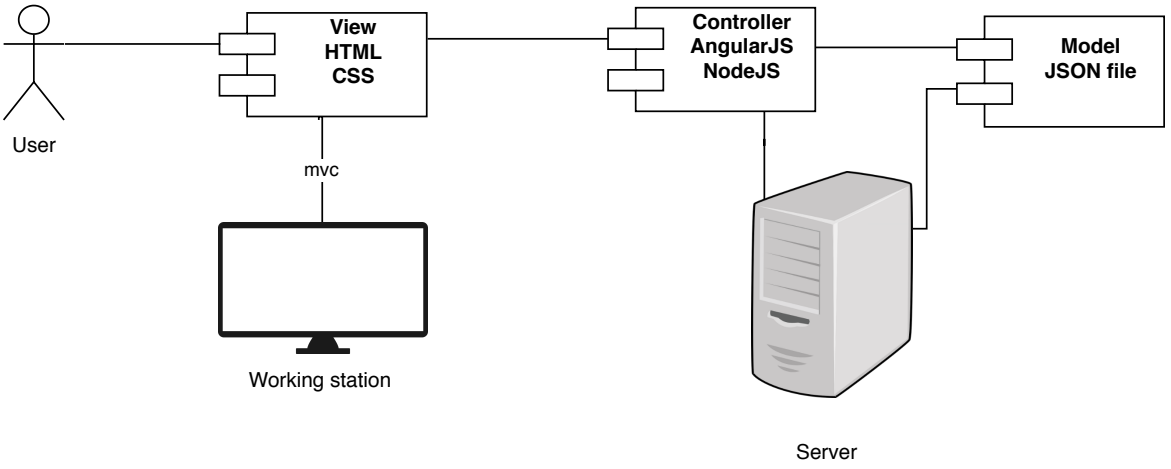
Do implementacji biblioteki w *NodeJS* używamy modułów, które odpowiadają bibliotekom w JavaScript. Tworzymy zbiór funkcji, które następnie użyjemy w naszej aplikacji. Aby udostępnić na zewnątrz funkcje, należy stworzyć moduł oraz użyć słowa kluczowego *exports*. Aby wykorzystać stworzony moduł w innym pliku, należy użyć słowa kluczowego *require*.

Biblioteka została zaimplementowana w pliku *piggyback.js*. Wyeksportowane zostały odpowiednie funkcje umożliwiające: stworzenie serwera na wybranym numerze portu, obsługę zapytań i odpowiedzi zgodnie z zasadą piggyback, emitowanie dodatkowych informacji przychodzących na serwer, w celu sprawdzenia poprawności metody.

4 Wykorzystane wzorce projektowe

Oprócz poniższych wzorców projektowych użytych w bibliotece zastosowałyśmy wzorzec architektoniczny **MVC** (Model View Controler), który pozwala na trójwarstwową budowę

aplikacji. W naszym przypadku *NodeJS* będzie odpowiedzialny za kontrolowanie systemu od strony serwera, *AngularJS* odpowiada za stronę kliencką aplikacji, a widok umożliwi *HTML, CSS*.



Rysunek 2: Schemat architektury na podstawie wzorca MVC

4.1 Iterator

Zrealizowany jest wzorzec operacyjny **Iterator**, który zapewnia sekwencyjny dostęp do obiektów bez ujawniania ich reprezentacji wewnętrznej. W naszym przypadku iteruje po obiekcie json (a dokładniej po jego kluczach, zwracając wartości im odpowiadające) przechowującym eventy, które są wysyłane przez klientów.

Zaletą tego wzorca jest umożliwienie jednoczesnego działania wielu procesów przechodzenia po obiekcie. Ułatwia również pracę ze złożonymi strukturami, takimi jak obiekty json, i zapewnia funkcje, które można użyć wielokrotnie.

W projekcie realizowany jest przez obiekt *JsonIterator*.

4.2 DAO (Data Access Object)

Używane do wykonywania operacji na źródle danych (np. bazach danych), w naszym przypadku - pliki ze zdarzeniami. Dostarcza jednolity interfejs do komunikacji między aplikacją a plikiem z danymi. Aplikacja nie musi znać miejsca i sposobu składowania swoich danych.

Uniwersalność i niezależność to główna zaleta tego wzorca. Dla programowania biblioteki będzie dobrym rozwiązaniem.

W projekcie realizowany jest przez obiekt *LogDao*.

4.3 Factory Method

Wzorzec wykorzystany jest do utworzenia obsługi dwóch zapytań - GET oraz POST. Aby ułatwić pracę, zostały one zaimplementowane przez jedną klasę, a zmieniona została jedynie zmienna o rodzaju metody. Dzięki temu zapytania są uporządkowane w jednym miejscu, a przygotowywanie odpowiedzi na którekolwiek z tych zapytań przebiega tak samo dzięki *factory method*.

Zaletą tego wzorca jest przejrzystość i łatwa dostępność do elementów. W przypadku posiadania dwóch obiektów Factory Method było łatwiejsze do zaimplementowania od Fabryki abstrakcyjnej, która utworzyłaby wiele skomplikowanego kodu, zupełnie niepotrzebnego w aktualnej sytuacji.

W projekcie został zrealizowany poprzez funckję *takeMethod* obiektu Factory.
Obiekt *Factory* działa jako klasa **Creator**, a *getMethod i postMethod*, jako klasy **Concrete Products**.

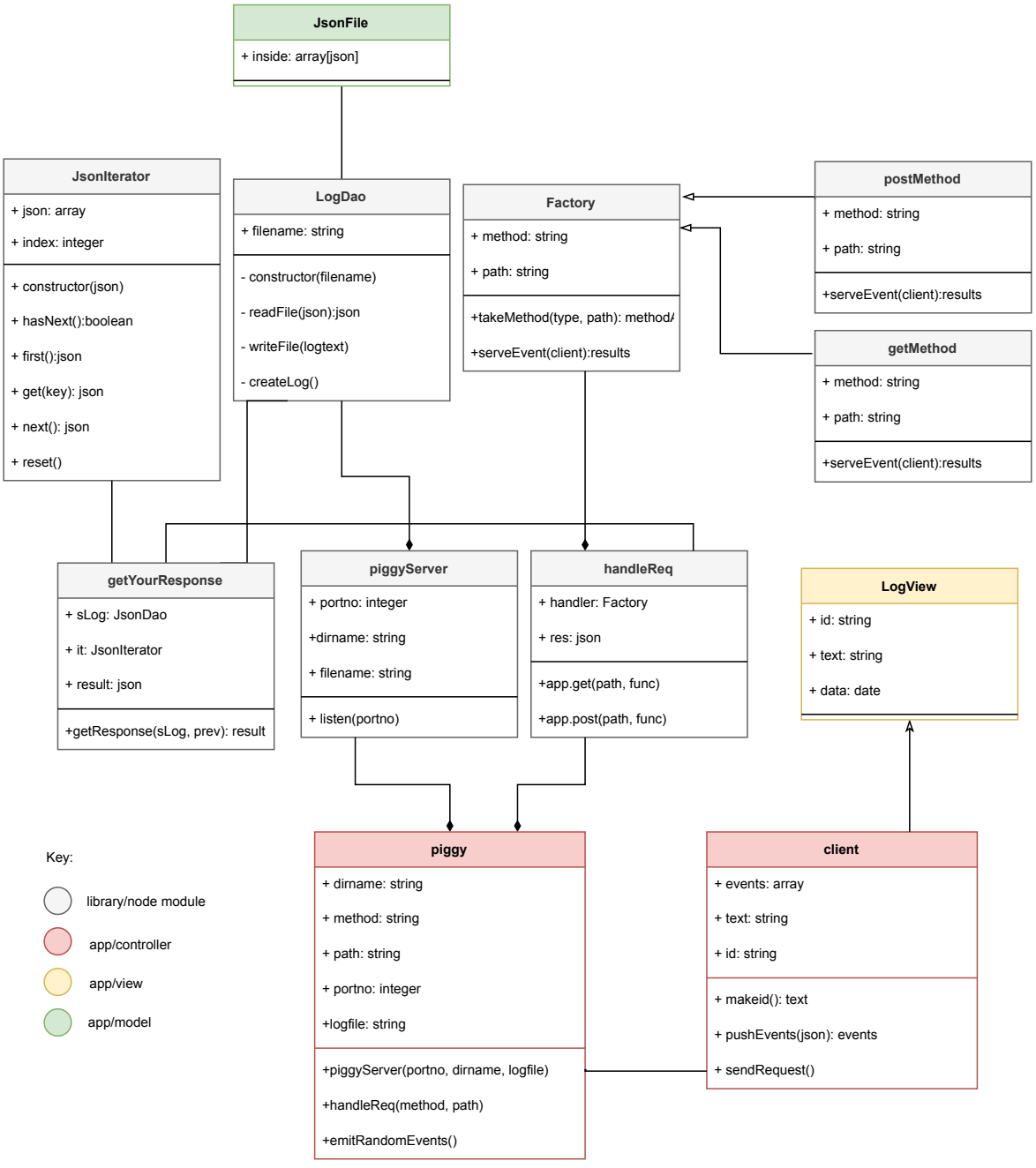
5 Opis aplikacji

Do zilustrowania działania metody piggyback zdecydowałyśmy się na stworzenie aplikacji webowej pozwalającej na czytanie i tworzenie logów przez użytkowników na stronie. Kilku klientów może jednocześnie korzystać z serwera, co pozwala na zauważenie działania piggyback. Kiedy klient tworzy nowy log i naciska przycisk *Click me!*, oczekuje od serwera odpowiedzi - wyświetlenie stworzonego loga. Serwer odsyła odpowiedź uzupełnioną o logi, które zostały dodane przez innych użytkowników od czasu ostatniego loga danego klienta. Możliwe jest ustawienie pojawiających się losowo zdarzeń na serwerze, które również zostają dołączone do odpowiedzi.

Zgodnie z paradygmatami MVC, podzieliłyśmy program na trzy części. W kontrolerach sprawdzona zostaje zawartość i poprawność danych. Model służy jako połączenie z plikiem z danymi. W widoku użytkownik używa przycisku, aby stworzyć nowy log. Kontroler przekazuje informacje do modelu, który następnie pobiera odpowiednie dane z pliku. Odczytywana jest ostatnia aktywność użytkownika i na jej podstawie wysyłane są informacje, które pojawiły się od ostatniej aktywności danego użytkownika oraz jego nowy log. Kontroler następnie przekazuje te wiadomości do widoku, który wyświetla je na interfejsie użytkownika.

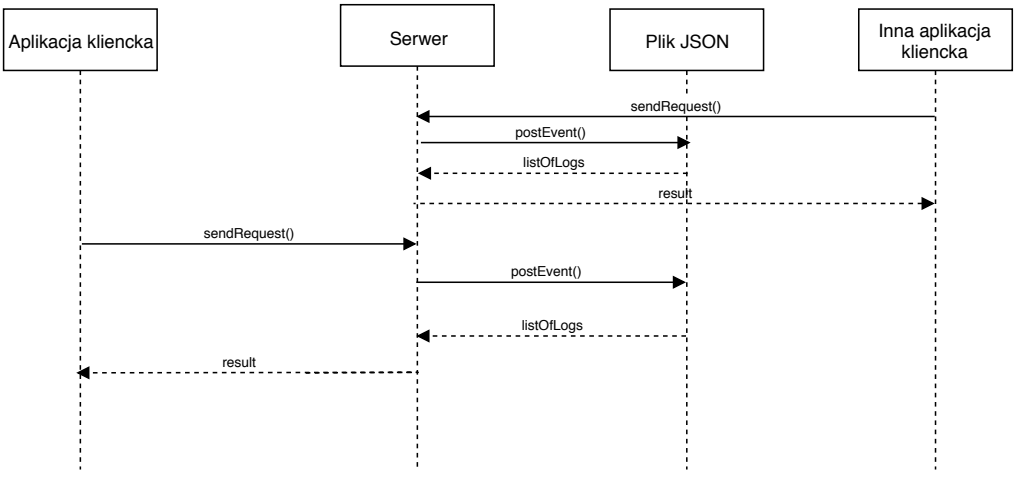
6 Struktura aplikacji

6.1 Diagram klas



6.2 Diagram przypadków użycia

6.3 Diagram sekwencji

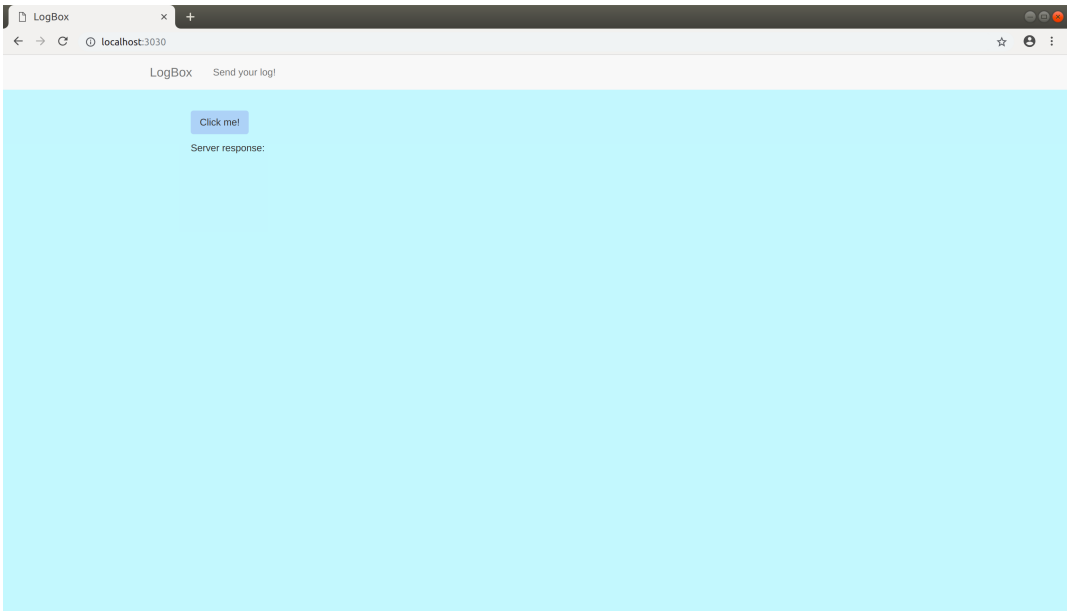


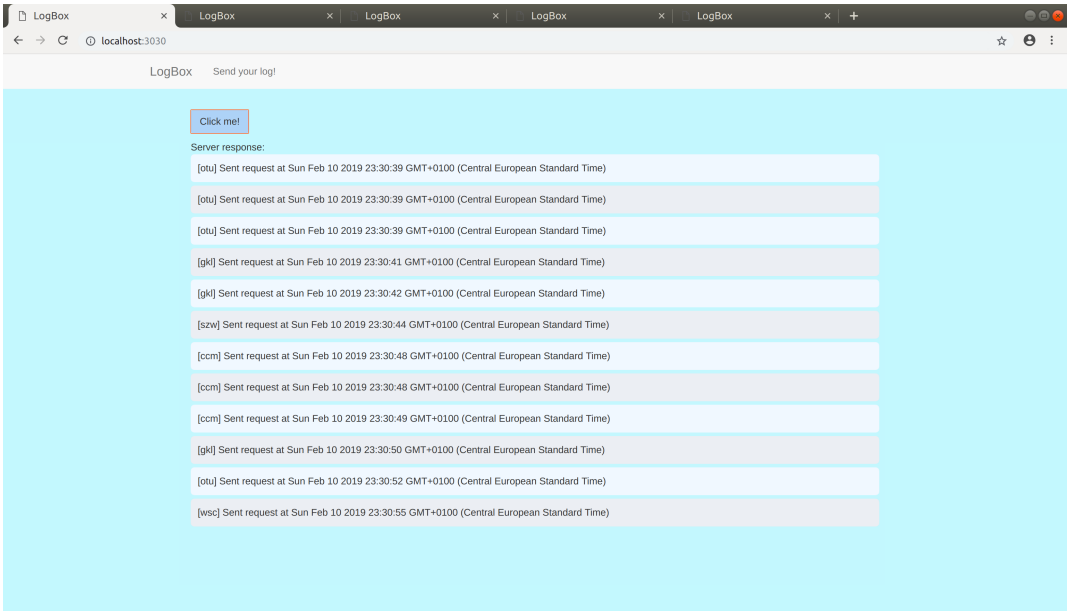
7 Uruchamianie i wymagania

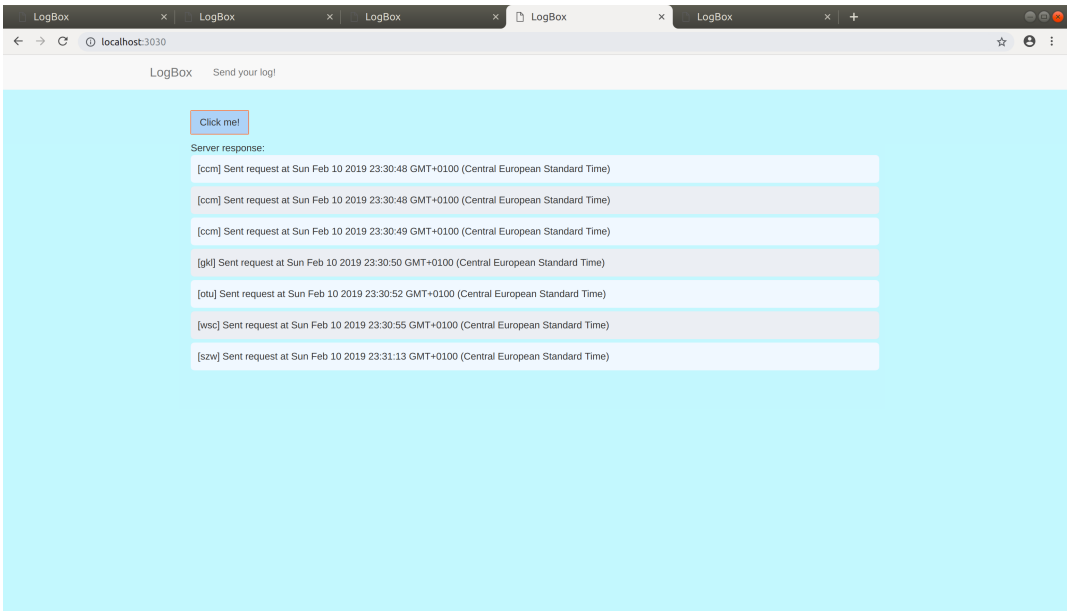
Biblioteka
W celu użycia biblioteki, należy wymusić jej użycie poprzez:
`var piggy = require('./piggyback.js');`
Stworzenie serwera:
`piggy.piggyServer(3030, 'public', 'myeventlog.json');`
Obsługa żądań *post*:
`piggy.handleReq('post','/');`
Obsługa żądań *get*:
`piggy.handleReq('get','/add/:id/:msg/');`
Emitowanie dodatkowych eventow:
`piggy.emitRandomEvents();`

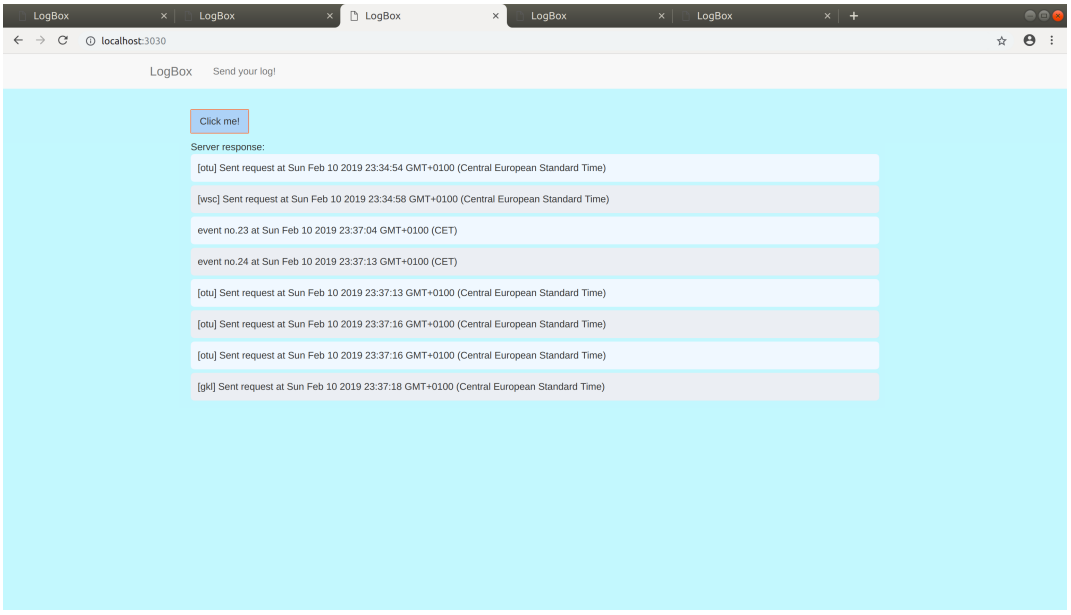
Aplikacja
Aby uruchomić demo aplikacji należy wpisać w terminalu polecenie:
`node demo.js`
Następnie w przeglądarce uruchomić:
`http://localhost:3030`
Stworzony serwer nasłuchuje na tym porcie.

8 Wynik aplikacji






LogBox
LogBox
LogBox
LogBox
LogBox
localhost:3030
LogBox
Send your log!
Click me!
Server response:
[ccm] Sent request at Sun Feb 10 2019 23:30:48 GMT+0100 (Central European Standard Time)
[ccm] Sent request at Sun Feb 10 2019 23:30:48 GMT+0100 (Central European Standard Time)
[ccm] Sent request at Sun Feb 10 2019 23:30:49 GMT+0100 (Central European Standard Time)
[gkl] Sent request at Sun Feb 10 2019 23:30:50 GMT+0100 (Central European Standard Time)
[otu] Sent request at Sun Feb 10 2019 23:30:52 GMT+0100 (Central European Standard Time)
[wsc] Sent request at Sun Feb 10 2019 23:30:55 GMT+0100 (Central European Standard Time)
[szw] Sent request at Sun Feb 10 2019 23:31:13 GMT+0100 (Central European Standard Time)


LogBox
LogBox
LogBox
LogBox
LogBox
localhost:3030
LogBox
Send your log!
Click me!
Server response:
[otu] Sent request at Sun Feb 10 2019 23:34:54 GMT+0100 (Central European Standard Time)
[wsc] Sent request at Sun Feb 10 2019 23:34:58 GMT+0100 (Central European Standard Time)
event no.23 at Sun Feb 10 2019 23:37:04 GMT+0100 (CET)
event no.24 at Sun Feb 10 2019 23:37:13 GMT+0100 (CET)
[otu] Sent request at Sun Feb 10 2019 23:37:13 GMT+0100 (Central European Standard Time)
[otu] Sent request at Sun Feb 10 2019 23:37:16 GMT+0100 (Central European Standard Time)
[otu] Sent request at Sun Feb 10 2019 23:37:16 GMT+0100 (Central European Standard Time)
[gkl] Sent request at Sun Feb 10 2019 23:37:18 GMT+0100 (Central European Standard Time)