JSON-RPC

Implementacja w Node.js z wykorzystaniem biblioteki Jayson

Narzędzia:

- Środowisko Node.js - najlepiej w wersji najnowszej stabilnej (w chwili przygotowywania: v8.9.x).
- Edytor kodu lub IDE - do wyboru: Visual Studio Code, Notepad, Webstorm, Atom, Notatnik, etc.
- Program Postman lub wtyczka Postman do przeglądarki Chrome.

Plan ćwiczeń

1. Utworzenie nowego folderu “json-rpc”.
2. Utworzenie nowego pliku “server.js” we wcześniej utworzonym folderze.
3. Instalacja biblioteki “jayson” poprzez wywołanie komendy “npm install jayson”
4. Import modułu “jayson” do pliku “server.js” i utworzenie szkieletu usługi oferującej zdalne wywoływanie procedur, wykorzystującej protokół HTTP i nasłuchującej na porcie 3000:

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

const server = jayson.server(
   // object with methods
);

server.http().listen(3000);
```

5. Implementacja metody “sayHello” na serwerze, która będzie zwracać wartość “Hello”:

```
sayHello: (args, callback) => callback(null, 'Hello')
```

6. * Wywołanie metody “sayHello” w programie Postman w celu prześledzenia dokładnej komunikacji z serwerem (należy zaimportować kolekcję ”JSON-RPC” dostępną w repozytorium do programu Postman). Serwer uruchamiamy komendą ”node server.js” będąc w katalogu “json-rpc”.
7. Utworzenie nowego pliku “client.js” w folderze “json-rpc”.

8. Import modułu “jayson” do pliku “client.js” i utworzenie szkieletu klienta zdalnego wywoływania procedur wykorzystującego HTTP i wykonującego zapytania na porcie 3000:

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

const client = jayson.client.http({
   port: 3000
});
```

9. Implementacja zdalnego wywołania metody “sayHello” po stronie klienta:

```
client.request('sayHello', {}, (err, response) => {
   if(err) throw err;
   console.log(response.result);
});
```

10. Przetestowanie działania: (będąc w katalogu “json-rpc”) uruchomienie serwera w terminalu komendą ”node server.js” oraz uruchomienie aplikacji klienckiej w drugim oknie terminala komendą ”node client.js” - obserwacja wyjścia na konsoli, w której działa aplikacja klienta.
11. Implementacja metody “add” na serwerze przyjmującej dwa argumenty i zwracającej ich sumę:

```
add: (args, callback) => callback(null, args[0] + args[1])
```

12. Implementacja zdalnego wywołania metody “add” po stronie klienta:

```
client.request('add', [ { arguments } ], (err, response) => {
   if(err) throw err;
   console.log(response.result);
});
```

13. Przetestowanie działania: patrz pkt. 10.
14. Spreparowanie danych służących jako atrapa bazy danych:

```
const usersData = [
   {
       name: { string },
       email: { string }
   }
]
```

15. Implementacja metody “getUser” zwracającej dane użytkownika o zadanej nazwie:

```
getUser: (args, callback) => callback(null, usersData.find((object) =>
object.name === args.name))
```

16. Implementacja zdalnego wywołania metody “getUser” po stronie klienta:

```
client.request('getUser', {name or/and email field with string value},
(err, response) => {
   if(err) throw err;
   console.log(response.result);
});
```

17. Przetestowanie działania: patrz pkt. 10.