Politechnika Śląska

Wydział Matematyk Stosowanej

Kierunek Informatyka

Gliwice, 01.10.2020

Programowanie I

**projekt zaliczeniowy**

**"*Serwer HTTP*"**

**Krzysztof Czuba gr. lab. 7**

**1. Opis projektu.**

Prosty Serwer HTTP

<https://github.com/kriskros341/SocketHTTPServer>

**2. Wymagania**

- Klasa obsługująca połączenia pomiędzy socketami oraz rozwijana przez nią klasa serwera HTTP

- Serwowanie plików lokalnych lub funkcji jako endpointów.

- Kontekst jako globalny stan serwera (np. baza danych) i stan prywatny poszczególnych endpointów stworzonych klasowo.

- w pełni asynchroniczna obsługa zapytań.

- Funkcja mapująca rekursywnie zawartość folderu na endpointy.

- Obsługa aplikacji tradycyjnych ze stanem po stronie serwera oraz renderowanych po stronie klienta z serwerem jako REST API

- minimalistyczne API nie wymaga żadnego wcześniejszego doświadczenia z HTTP, które jednocześnie daje kontrolę nad każdym aspektem wymiany danych w granicach zasad protokołu.

- zapytania i odpowiedzi reprezentowane są przez struktury. Posiadają metodę zwracającą dane gotowe do przesłania.

- wektor obiektów middleware do implementacji powtarzającej się logiki.

**3. Przebieg realizacji**

Przeczytałem książkę o komunikacji przez sockety w C.

Dostosowałem kod do bardziej obiektowego C++.

Zaimplementowałem na nich HTTP podążając za dokumentacją MDN.

Dodałem kontekst jako sposób na dzielenie danych globalnych pomiędzy endpointami.

Utwożyłem demo

Zamieniłem podprocesy na wątki przy obsłudze zapytań.

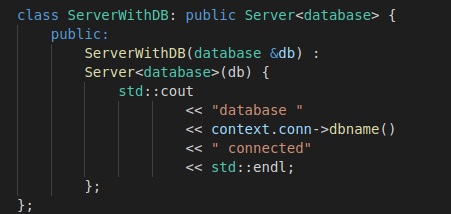
Dodałem warstwę obiektów pośredniczących middleware dla powtażającej się logiki.

Projekt składa się z pliku server.h oraz server.cpp w folderze include. Nie wykorzystuje żadnych bibliotek spoza STL.

Demo zawiera main.cpp, skrypt compile.sh, bibliotekę odpowiadającą za JSON oraz folder build z aplikacją ReactJS. Wymaga działania bazy danych na hoście w sieci lokalnej (W przypadku dema jest to Postgres, wirtualny Debian)

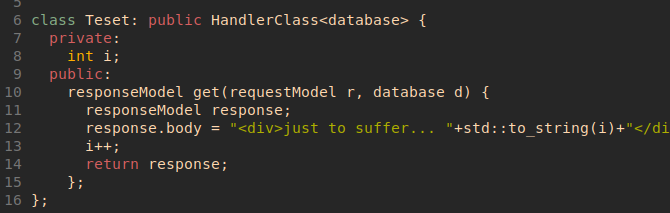
**4. Instrukcja użytkownika**

1. Użytkownik tworzy instancję serwera, wypełniając template oraz przekazuje do konstruktora obiekt tego samego typu. Jest to stan globalny dostępny w każdej funkcji. Przykładowo:

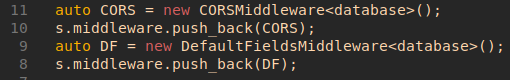




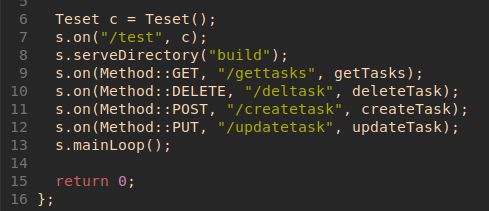
2. Użytkownik definiuje funkcje lub obiekty endpointowe. Funkcje odpowiadają jednemu z typów handlerFunction<>, a obiekty rozszerzają klasę HandlerClass<>. Template powiela ten z serwera. Przykładowo:



3. Opcjonalnie określa middleware. Własne rozszerzające klasę MiddlewareFunctor, lub predefiniowane



4. Określa endpointy poprzez powiązanie metody i ścieżki z funkcją, lub ścieżki z obiektem rozszerzającym ClassEndpoint<>



Gotowe!

**5. Podsumowanie i wnioski.**

Pierwszym problemem na jaki natrafiłem było przechowywanie lambd, metod i funkcji o tej samej sygnaturze w pojedyńczym typie. std::variant, std::any ani Union nie wchodziły w grę.

Rozwiązaniem tego problemu był struktura std::function dostępna w module <functional>. Bardzo ułatwiło tworzenie domknięć, callbacków i bindów przy pomocy funkcji anonimowych.

Nie udało się dodać szyfrowania TLS. Aktualnie najłatwiej użyć reverse proxy z szyfrowaniem i przekierować ruch do serwera bez szyfrowania.

Planowałem dodać pool wątków tak, aby nie tworzyć nowego przy każdym zapytaniu, ale nie.