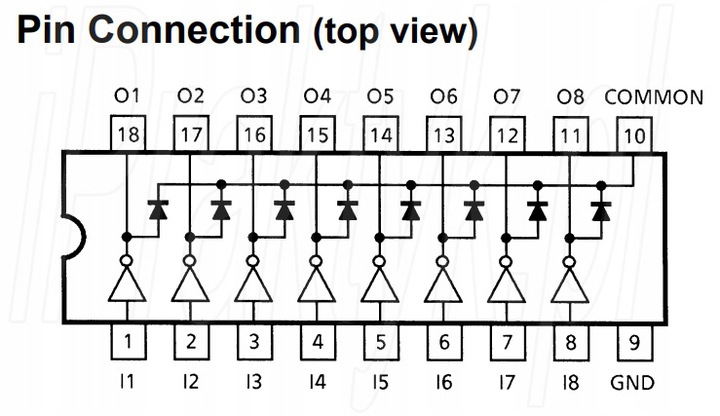
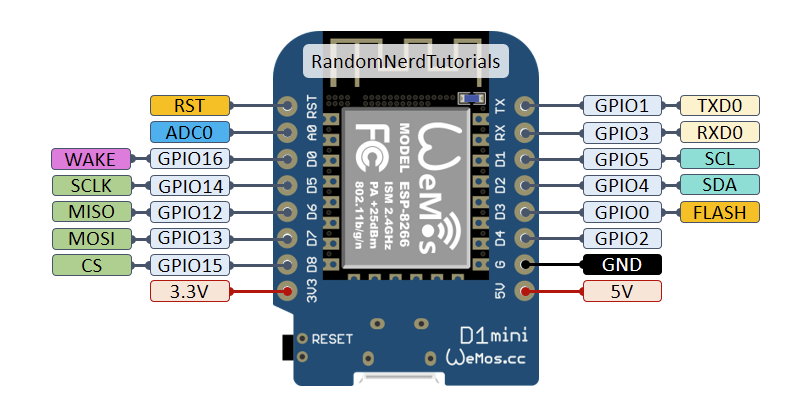
Do realizacji projektu została użyta listwa elektryczna w której każde z gniazd sterowane jest odrębnym przekaźnikiem. Model S14-1C-0516 z cewką 5V umożliwia przepływ prądu do 16A po stronie uzwojenia wtórnego, co w znacznej mierze będzie wystarczające aby spełnić wcześniej wymienione założenia.

Rolę mikrokontrolera spełnia WeMOS D1 mini NodeMCU zintegrowany z modułem ESP8266 do komunikacji bezprzewodowej w standardzie 802.11 b/g/n. Został on połączony z układem TOSHIBA ULN2803APG, to 8-kanałowy sterownik zbudowany w oparciu o 8 bardzo wydajnych układów Darlingtona, czyli wzmacniaczy złożonych z tranzystorów bipolarnych o szczególnie dużym wzmocnieniu. Przekaźnik listwy zostaje wysterowany impulsem stanu wysokiego ze wspólnego wejscia COMMON za pomocą bramki logocznej tego drivera.



Sam moduł Ardiuno umożliwia podanie stanu wysokiego lub niskiego na dowolny z pinów GPIO. Należy zwrócić szczególna uwagę, aby nie obciążać zbyt bardzo wejść\wyjść cyfrowych gdyż mają dość duże ograniczenia prądowe, co jest związane z budową samego układu. W takiej sytuacji kontroler nie uruchomi się wcale lub jego praca będzie niestabilna.



Rolę serwera odpowiedzialnego za przekazywanie komunikatów między przeglądarką użytkownika a modułem wykonawczym spełni tutaj aplikacja ASP.NET Core opracowana w .NET6. Ze względu na swoją szybkość oraz możliwość uruchomienia na systemach Linux jest tutaj odpowiednim wyborem. Komunikacja odbywa się za pośrednictwem protokołu http oraz WebSocket. Każde połączenie jest nawiązywane za pomocą protokołu HTTP. Jeśli serwer obsługuje protokół WebSocket, zgadza się zaktualizować połączenie. To tzw. “handshake”. Po akceptacji początkowe połączenie HTTP zostaje zastąpione połączeniem WebSocket, które korzysta z tego samego protokołu TCP/IP. W tym momencie dane mogą swobodnie przepływać między klientem a serwerem. Właśnie dlatego jest to dobre rozwiązanie ponieważ umożliwi komunikację w obie strony w dowolnej jednostce czasu. Korzystając wyłącznie z protokołu http nie było by możliwości przesłania komunikatu bezpośrednio do kontrolera.

Jak wygląda komunikacja?

Komunikacja między urządzeniami IoT, serwerem a przeglądarką odbywa się w oparciu o prosty obiekt zawierający nazwę polecenia oraz argumenty.

public class RequestModel

{

[JsonConverter(typeof(StringEnumConverter))]

public Command Command { get; set; }

public string? Args { get; set; }

}

public enum Command

{

SWITCH\_LED\_DONE,

GET\_GPIO\_LIST,

REGISTER\_NODE\_MCU,

SWITCH\_LED,

UNREGISTER\_NODE\_MCU,

REGISTER\_WEB\_BROWSER,

GET\_GPIO\_LIST\_DONE,

SWITCH\_GPIO\_DONE,

SWITCH\_GPIO,

OFF\_GPIO,

ON\_GPIO,

OFF\_GPIO\_DONE,

ON\_GPIO\_DONE

}

Do poprawnego prasowania tej struktury po stronie kontrolera została użyta zewnętrzna biblioteka ArduinoJson. Format ten jest bardzo popularny w wielu systemach informatycznych ze względu na swoja prostotę oraz oszczędność wielkości wysyłanych paczek. Wszystkie komunikaty są obsługiwane przez moduł routingu. Przekazuje on każde z wysłanych poleceń do odpowiednich odbiorców. Dodatkowo takie rozwiązanie daje możliwość bardzo szybkiej implementacji dodatkowych funkcjonalności w późniejszym czasie. Odczyt z czujników czy obsługa dodatkowych urządzeń podłączonych do kontrolera sprowadzi się do napisania dodatkowego polecenia dla istniejącego już kontraktu. Aby ułatwić ten proces aplikacja serwerowa korzysta z wzorca projektowego jakim jest dependency injection. Wstrzykiwanie zależności pozwala na łatwiejsze zarządzanie projektem w późniejszym czasie.

Użytkownik przechodzi na witrynę obsługującą nasze urządzenie IoT. Powoduje to nawiązanie połączenia do serwera który zwraca stronę html. Po jej załadowaniu z wykorzystaniem JavaScriptu zostanie nawiązane połączenie do endpointa obsługującego WebSockets. Jeśli ten proces został zakończony powodzeniem klient jakim jest przeglądarka wysyła polecenie REGISTER\_WEB\_BROWSER. Serwer w reakcji zapisuje sobie referencję do tego połączenia. Od tej chwili przeglądarka czeka na podłączenie się kontrolera. Adruino po dostarczeniu zasiania, ustawia wybrane wcześniej piny GPIO w tryb „wyjścia”, co umożliwi na zmianę ich stanu logicznego. W dalszej kolejności nawiązywana jest próba połączenia z siecią wifi z użyciem poświadczeń podanych w konfiguracji. Po poprawnym uzyskaniu adresu IP kontroler przechodzi do nawiązania komunikacji z endpointem WebSocket za pośrednictwem bezpiecznego kanału SSL. Kiedy serwer potwierdzi zestawienie połączenie wykonywana jest inicjalizacja urządzenia. Polega to na wysłaniu komunikatu REGISTER\_NODE\_MCU, co umożliwia serwerowi przechowanie referencji tego połączenia w celu dalszego procesowania komunikacji między klientami. W chwili odebrania tego zgłoszenia przez moduł routingu wysyła on zapytanie do modułu wykonawczego o podanie listy interfejsów GPIO (GET\_GPIO\_LIST) jakie mogą zostać udostępnione użytkownikowi poprzez przeglądarkę. NodeMCU po odebraniu komunikatu zwraca wypełnioną zmienną argumentów o listę dostępnych numerów GPIO potwierdzając to poleceniem GET\_GPIO\_LIST\_DONE. Serwer przekazuje ten pakiet bezpośrednio do przeglądarki, która na tej podstawie renderująca odpowiednią ilość przycisków. Kliknięcie w dowolny przycisk wywołuje SWITCH\_GPIO której argument jest numerem wyjścia GPIO kontrolera. Pakiet jest kierowany do naszego urządzenia IoT. Moduł ten wykonuje swoje polecenie odczytując aktualny stan wyjścia i zmieniając jego stan na odwrotny. Potwierdza to wysłaniem komendy SWITCH\_GPIO\_DONE. Umożliwia to oznaczenie przycisku w przeglądarce jako aktywny (podświetlony). Sytuacja wygląda analogicznie jeśli chodzi o wyłączenie i włączenie wszystkich wyjść. Przy czym kontroler nie przełącza stanu na odwrotny tylko ustawia go na taki który jest zgodny z treścią komunikatu, odpowiednio OFF\_GPIO lub ON\_GPIO. W związku z tym że otwarte połączenia z WebSocketami utrzymywane są przez dłuższy czas może dojść do sytuacji kiedy użytkownik wyśle komunikat do urządzenia IoT w chwili kiedy to będzie już niedostępne. Zaimplementowany został zatem asynchroniczny mechanizm usuwania zamkniętych już połączeń. Lista przechowująca referencje połączeń jest stale aktualizowana przez nowo dołączone urządzenia, które są do niej sukcesywnie dodawane. Jednocześnie należy z niej usuwać połączenia już zamknięte aby oszczędzać pamięć serwera. Sytuacja podczas wykonywania dwóch operacji asynchronicznych w jednym momencie na pewno doprowadzi do nieoczekiwanego błędu powodując problem z obsłudze klientów. Fragmenty kodu zawierające zmienne które nie mogą być jednocześnie odczytywane i zapisywane określany jest sekcją krytyczną, do której dostęp musi zostać zsynchronizowany poprzez zastosowanie semafora.

private void RemoveClosedConnections()

{

Task.Factory.StartNew(() =>

{

while (true)

{

lock (\_syncObject)

{

Sockets.RemoveAll(x => x.Socket.State == WebSocketState.Closed);

}

Thread.Sleep(5000);

}

});

}