Docelowy Spis treści :

1. Wstęp
2. Cel i zakres pracy
3. Teoria
   1. Programy
      1. Visual Studio Code
      2. FileZilla
      3. Putty
      4. Postman
   2. Urządzenia
      1. Raspberry
      2. Przekaźnik
      3. Czujniki
      4. Aktorzy
   3. Technologie
      1. HTML
      2. Python
      3. Flask
      4. SQLAlchemy
      5. SQLite
      6. GIT
4. Implementacja systemu
   1. Hardware
      1. Spis komponentów / cennik
      2. Schemat układu
   2. Software
      1. Ustawienie serwera
      2. Połączenie z bazą danych
      3. API
      4. Algorytmy użycia aktorów
      5. Wykonywanie funkcji w zaplonowanym czasie
5. Testy
   1. Testy manualne
   2. Testy jednostkowe
   3. Testy API
6. Plany rozwoju
7. Podsumowanie
8. Bibliografia
   1. Publikacje książkowe
   2. Strony internetowe

**Wstęp**

Celem pracy jest stworzenie systemu, który automatyzuje proces dbania o roślinę oraz pozwala na dostosowanie parametrów przez interface sieciowy. System postawiony jest na platformie Raspberry Pi oraz miniframeworku Flask.

Podstawowe założenia dotyczące systemu :

1. System jest przeznaczony do dbania o rośliny doniczkowe co minimalizuje jego rozmiary oraz potrzebne źródła zasilania
2. System cyklicznie wykonuje algorytmy użycia urządzeń elektronicznych zapewniających roślinie odpowiednie warunki.
3. Parametry rośliny można wprowadzić przez stronę internetową
4. Udostępniona strona internetowa skaluje się na urządzenia mobilne
5. Inną drogą komunikacji z system jest Web API

Niniejszy dokument stanowi formę dokumentacji stworzonego systemu, zawiera on opis stworzonych komponentów i wszelkie informacje potrzebne do zrozumienia czy odtworzenia systemu.

**zakres pracy**

1. Określenie optymalnych wybranych warunków dla wzrostu wybranej rośliny
2. Przygotowanie elementów sterujących
3. Konfiguracja serwera
4. Utworzenie bazy danych
5. Implementacja API
6. Implementacja strony internetowej
7. Implementacja algorytmów użycia urządzeń elektronicznych
8. Przeprowadzenie testów

// To Do : dodac gdzies opis rozdziałów

**Teoria**

**Programy**

**Visual Studio Code**

Jest edytorem kodów źródłowych stworzonym przez firmę Microsoft. Bazowo zajmuje niewiele miejsca na dysku i jest prosty w użyciu, dzieje tak dlatego ponieważ edytor pozwala na dodanie rozszerzeń. Przez to przed przystąpieniem do projektu musimy poświęcić nieco uwagi rozszerzeniom których potrzebujemy ale wybór narzędzi jest przez to bardziej świadomy i mamy pewność że żadna funkcjonalność narzędzia nie uchodzi naszej uwadze.

**FileZilla Client**

Jest darmowym oprogramowaniem do połączeń z serwerami FTP/FTPS/SFTP które pozwala na wymianę plików. W pracy oprogramowanie to było wykorzystywane to przesyłania kodu źródłowego między stacją roboczą na której kod był pisany a serwerem na którym system był uruchamiany i testowany.

**Putty**

Jest oprogramowaniem pozwalającym na połącznia przez protokoły takie jak SCP/SSH/Telnet. Korzystałem z niego do podłączenia się do serwera na którym mieścił się system.

**Postman**

Oprogramowanie do testowania API, bardzo przydatne do testowania pojedynczych endpointów przy tworzeniu API ale też do tworzenia zautomatyzowanych testów API którego pomogą nam zapewnić poprawne działanie API nawet przy długotrwałym rozwoju oprogramowania przez kilka osób.

* + 1. **HTML**

HTML to skrót od HyperText Markup Language jest prostym językiem do budowania struktury stron internetowych. Stronę taką buduje się przez odpowiednie wykorzystanie tzw. Znaczników. Każda ze stron internetowych powinna być podzieloną na dwie części zawierające metadane i drugą zwierająca strukturę strony. podział ten oznacza że każda strona powinna być bazowana na odpowiednim szablonie który zawiera znaczniki takie jak html/head/body

* + 1. **Python**

Python jest popularnym otwartoźrudłowym językiem programowania używanym zarówno do tworzenia samodzielnych programów jak i samych skryptów. Jest prosty do nauki i można zastosować go na wielu platformach i do wielu zadań, posiada ogromną społeczność co sprawia że ilość bibliotek oraz materiałów do nauki jest znacznie większa niż byśmy byli w stanie się nauczyć

* + 1. **Flask**

Flask prostym frameworkiem który szybko pozwala nam na utworzenie API aplikacji napisanie w Pythonie. Flask jest microframeworkiem co znaczy że nie potrzebuje żadnych zewnętrznych narzędzi ani bibliotek. Nie ma on walidacji formularzy weryfikacji użytkownika, ORMów itd. Daje nam tylko bazową funkcjonalność co pozwala nam na napisanie tych funkcjonalności samemu bądź na skorzystanie z wybranych przez siebie bibliotek. Dla zainteresowanych bardziej kompleksowym frameworkiem do API w pythonie zachęcam do poczytania o Django. Gdybym użył Django do systemu który stworzyłem, nie skorzystał bym z wielu funkcjonalności które oferuje a Flask pozwolił mi na szybsze dobranie wymaganych komponentów co sprawie że system jest nie tylko bardziej przejrzysty ale też zajmuje mniej miejsca w pamięci.

* + 1. **SQLAlchemy**

SQLAlchemy jest ORMem (Object Relational Mapper) do Pythona, co znaczy że pozwala nam na przetworzenie obiektu z bazy danych na obiekt którym można posłużyć się w programie i na odwrót. Narzędzie to zapewnia nam zestaw wzorców zaprojektowanych do wydajnego dostępu do bazy danych, zaadaptowanych do języka jakim jest Python.

* + 1. **SQLite**

SQLite jest biblioteką która daje nam system zarządzania relacyjną bazą danych. Cechy SQLite:

* Wieloplatformowość : można użyć go do aplikacji Desktopwej, mobilnej, lub jak w tym przypadku systemu osadzonego którym jest Raspberry Pi
* Nie wymaga serwera : SQLite w przeciwieństwie do większości systemów zarządzania bazami danych jest zintegrowany z aplikacją i nie wymaga osobnego serwera po którym łączymy się protokołem TCP/IP jak w przypadku MySQL, PostgreSQL itd.
* Transakcyjny : wszystkie operacje na bazie są w pełni zgodne z ACID więc w przypadku wielu zapytań lub ich przerwania nie musimy się martwić o dane w bazie.

tym że nie trzeba go konfigurować, pozwala na transakcje, nie wymaga serwera oraz

* + 1. **GIT**

Narzędzie bez którego większość programistów nie wyobraża sobie pracy. GIT jest systemem kontroli wersji, co oznacza że śledzi wszelkie nasze zmiany w kodzie źródłowym od początku naszego projektu. Pozwala nam na cofnięcie się do poprzedniej wersji kodu w przypadku gdy coś w nim zepsujemy, bardzo przydatny w przypadku pracy zespołowej gdyż pozwala nam na zobaczenie kto, kiedy i co zmienił w kodzie.

**Spis komponentów :**

|  |  |
| --- | --- |
| Komponent | cena |
| Raspberry Pi 3 B+ | 200 |
| Moduł przekaźników 4 kanały 250VAC / cewka 5V | 25 |
| Konwerter ADC MCP3008 | 10 |
| fotorezystor GL5528 x2 | 4 |
| Czujnik wilgoci gleby | 8 |
| Taśma LED 300 GROW 1m | 50 |
| Pompa 5V | 20 |
| razem | 317 |

**Opis komponentów :**

Raspberry Pi 3 B+

Jest jednopłytkowym mikrokomputerem który posiada większość cech standardowego komputera takich jak złącza HDMI i USB, bezprzewodowe połączenie do sieci itd. System operacyjnym dedykowanym dla Raspberry jest Raspbian który jest bazowany na Debianie (dystrybucja linuxa). Dwie Główne wyróżniające cechy Raspberry Pi to jedynie 5V wymaganego zasilania oraz GPIO (general prupose input/output) które pozwalają na kontrole oraz odczyt komponentów elektrycznych co sprawia że Raspiberry Pi jest dobrym narzędziem to wszelkich projektów elektronicznych które potrzebują dostępu do internetu, większej ilości obliczeń lub komunikacji z innymi urządzeniami. Dlatego też możemy znaleźć Raspiberry w wielu projektach związanych z IoT oraz smart home.

Moduł przekaźników

Przekaźnik to w zasadzie przełącznik używający elektromagnesu. Elektromagnes potrzebuje niskiego napięcia by aktywować przełącznik co pozwala nam na włączenie/wyłączenie urządzenia które wymaga większego napięcia. Zapewniam nam to odseparowania źródeł zasilania co chroni nas przed przebiciem które może doprowadzić do uszkodzeń pozostałych komponentów które nie są przystosowane to takich wartości napięcia.

ADC (konwerter analogowo cyfrowy)

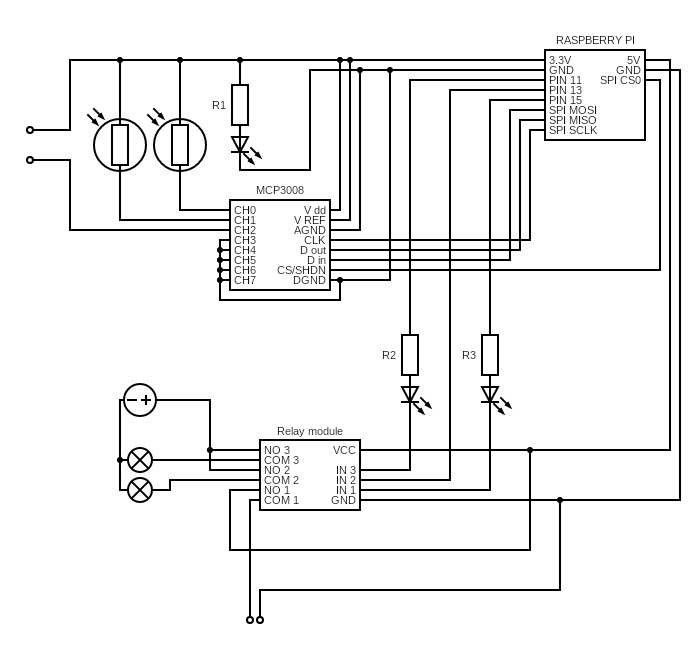
Jako że Raspiberry Pi nie jest w stanie sam odczytać wartości analogowych potrzebuje pomocy w postaci konwertera ADC. Konwerter analogowo cyfrowy to urządzenia które zmienia ciągły sygnał analogowy na przybliżony sygnał cyfrowy. Odbywa się to przez próbkowanie sygnału analogowego ze stała częstotliwością. Dokładność konwersji zależy głownie od 2 parametrów, czyli czasu konwersji oraz zakresu sygnału cyfrowego. W przypadku użytego ADC MCP3008 jest to 10us czasu konwersji oraz konwersja do 10 bitowej wartości czyli w zakresie 0 – 1024.

Fotorezystor

Czyli rezystor którego rezystancja jest zależna od siły nasłonecznienia. Pozwala nam on na pomiar wartości nasłonecznienia przez zliczenie różnicy napięć przed i za elementem, co pomoże nam określić czy roślina wymaga dodatkowego naświetlenia.

Czujnik wilgoci gleby  
  
Czujnik wilgotności gleby składa się z dwóch sond, które umożliwiają przepływ prądu przez wilgotną glebę. Można go bardzo łatwo wykorzystać, po prostu wkładając czujnik do gleby i odczytując wyniki za pomocą ADC, działa to ponieważ im większa wilgoć tym mniejsza rezystancja gleby.

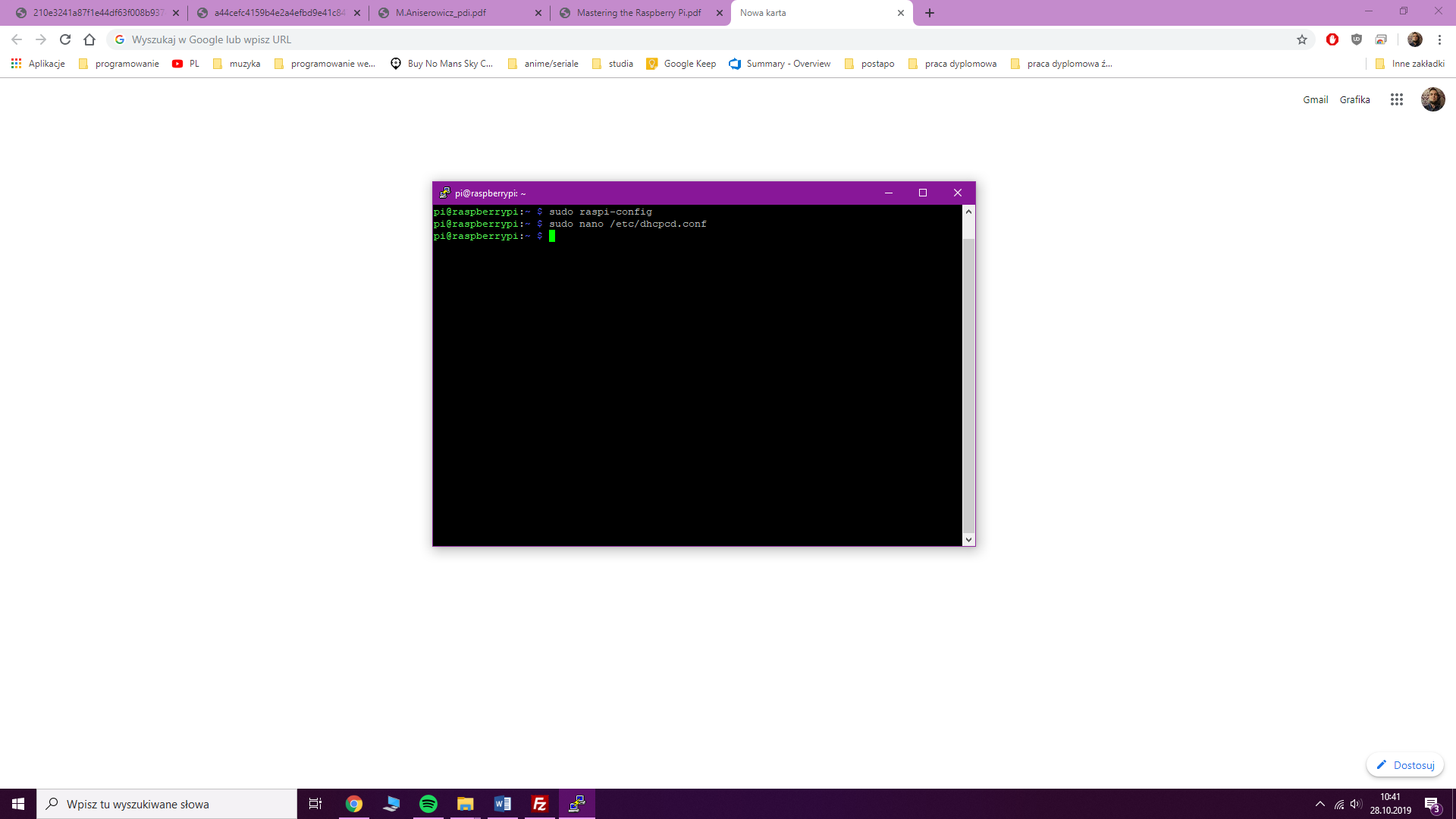
**Schemat układu :**

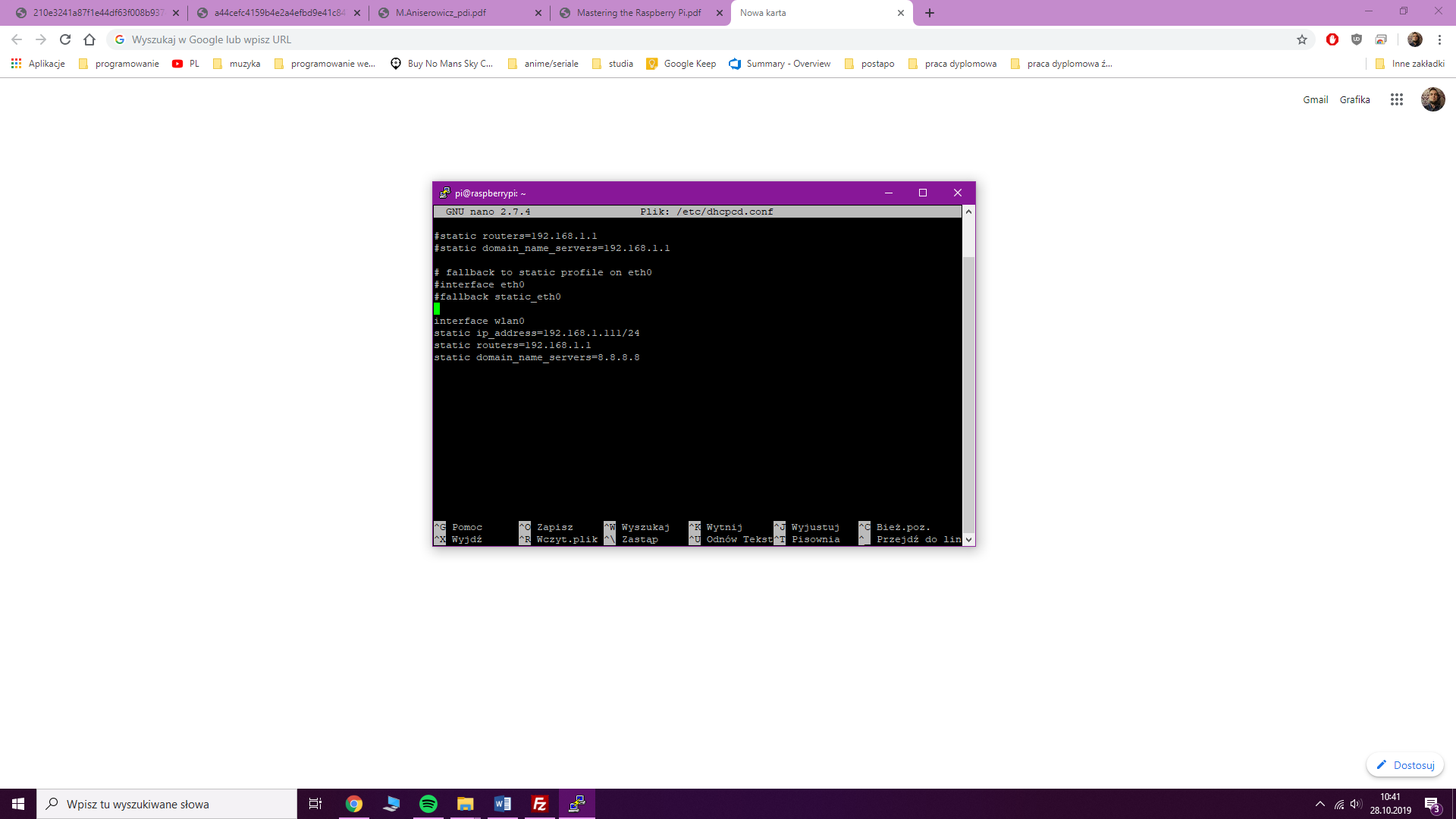


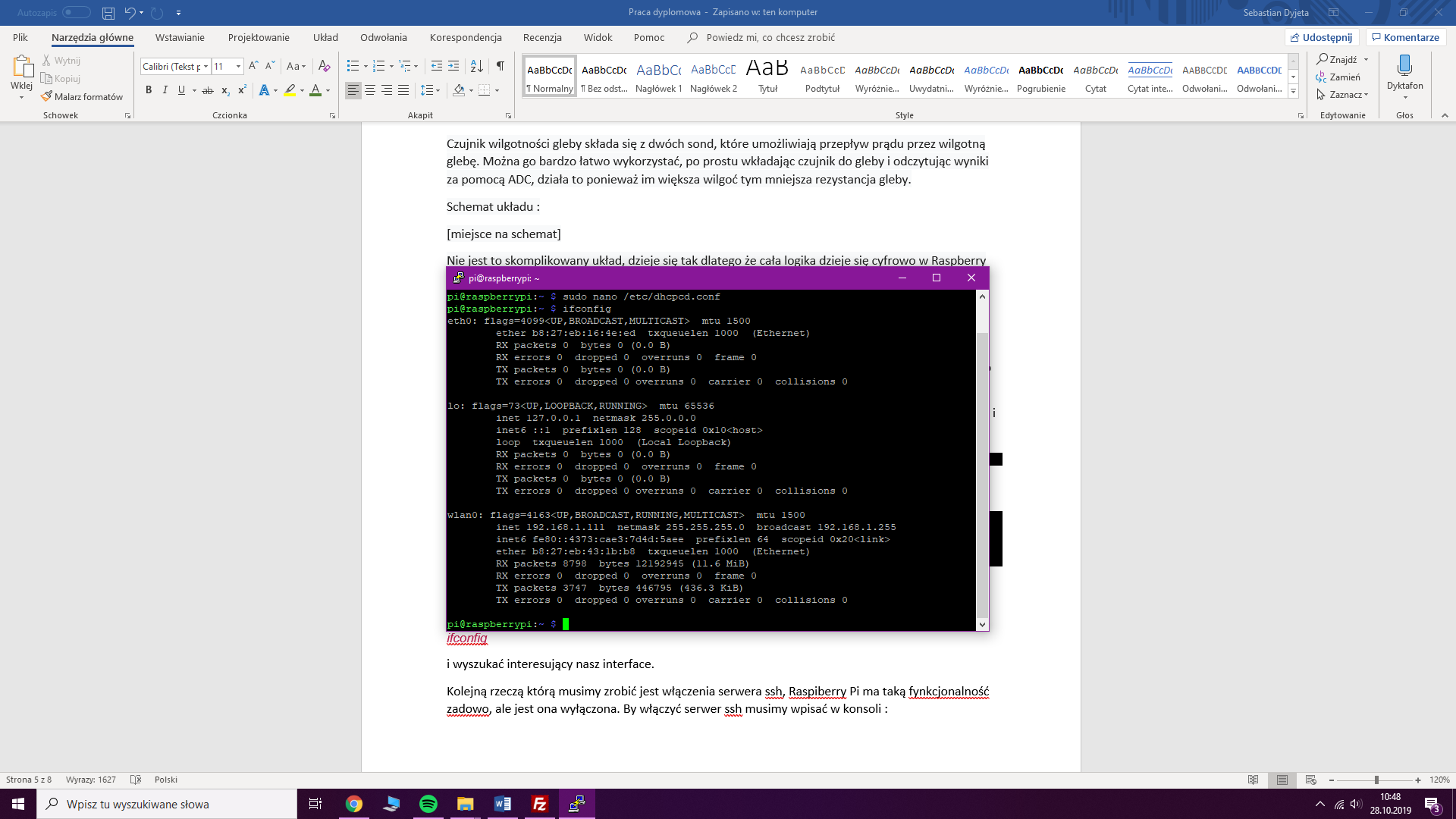
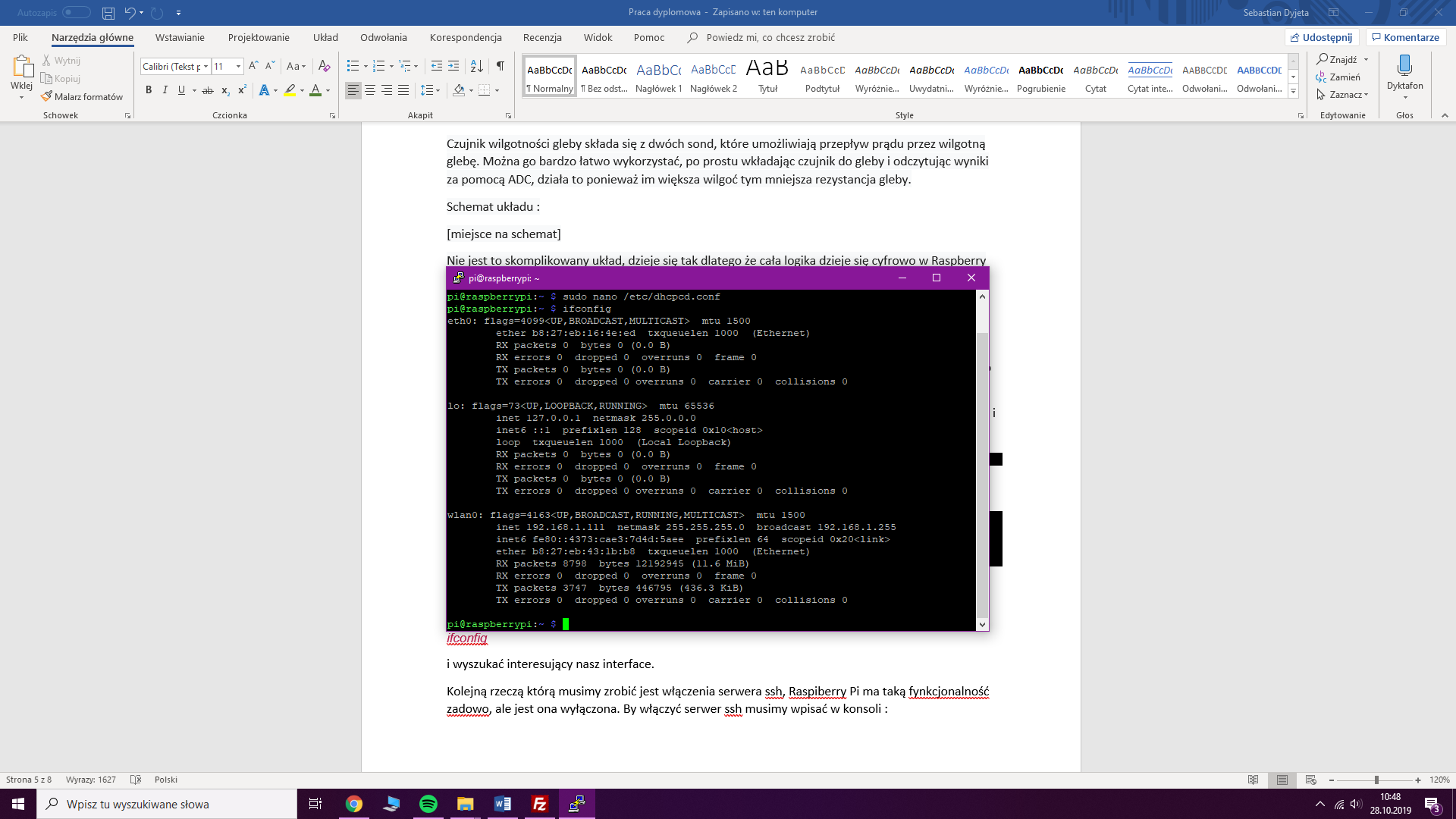
* + 1. **Konfiguracja serwera**

Na pączku pracy z Raspberry Pi musimy zadbać by udostępniało one odpowiednie porty do połączenia się z nim bez potrzeby podłączania peryferiów. Można co prawda napisać cały projekt bezpośrednio na mikrokomputerze ale jednak jego rozmiary świadczą też o jego wydajności co z poziomu działania systemu jest w pełni wystarczające, tak praca na nim może być nieco niekomfortowa gdy jesteśmy przyzwyczajenia do prędkości i funkcjonalności (np. kilka monitorów) które oferują standardowe stacje robocze.

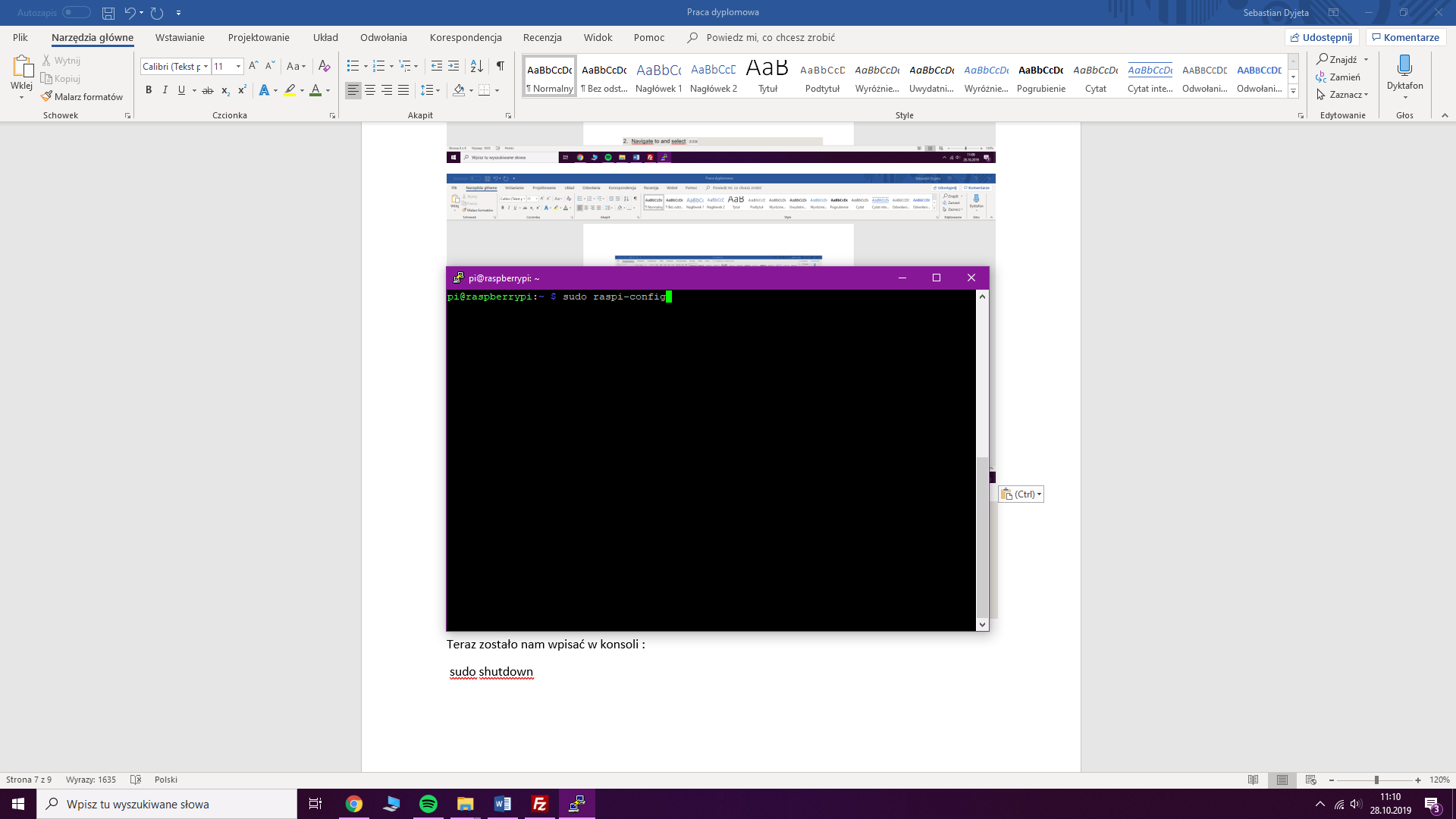
Zacznijmy od ustawienia statycznego adresu IP. Można to zrobić na kilka sposobów, osobiście preferuje zrobić to w pliku konfiguracyjnym na Raspberry Pi ponieważ wtedy naszą konfiguracje zabieramy wszędzie tam gdzie zabierzemy nasz system, co by nie miało miejsca gdybyśmy ustawili statyczny adres IP na routerze. Plik znajduje się pod ścieżką :

następnie schodzimy na dół pliku gdzie znajdziemy interface sieciowy wlan0, powinno to wyglądać tak :

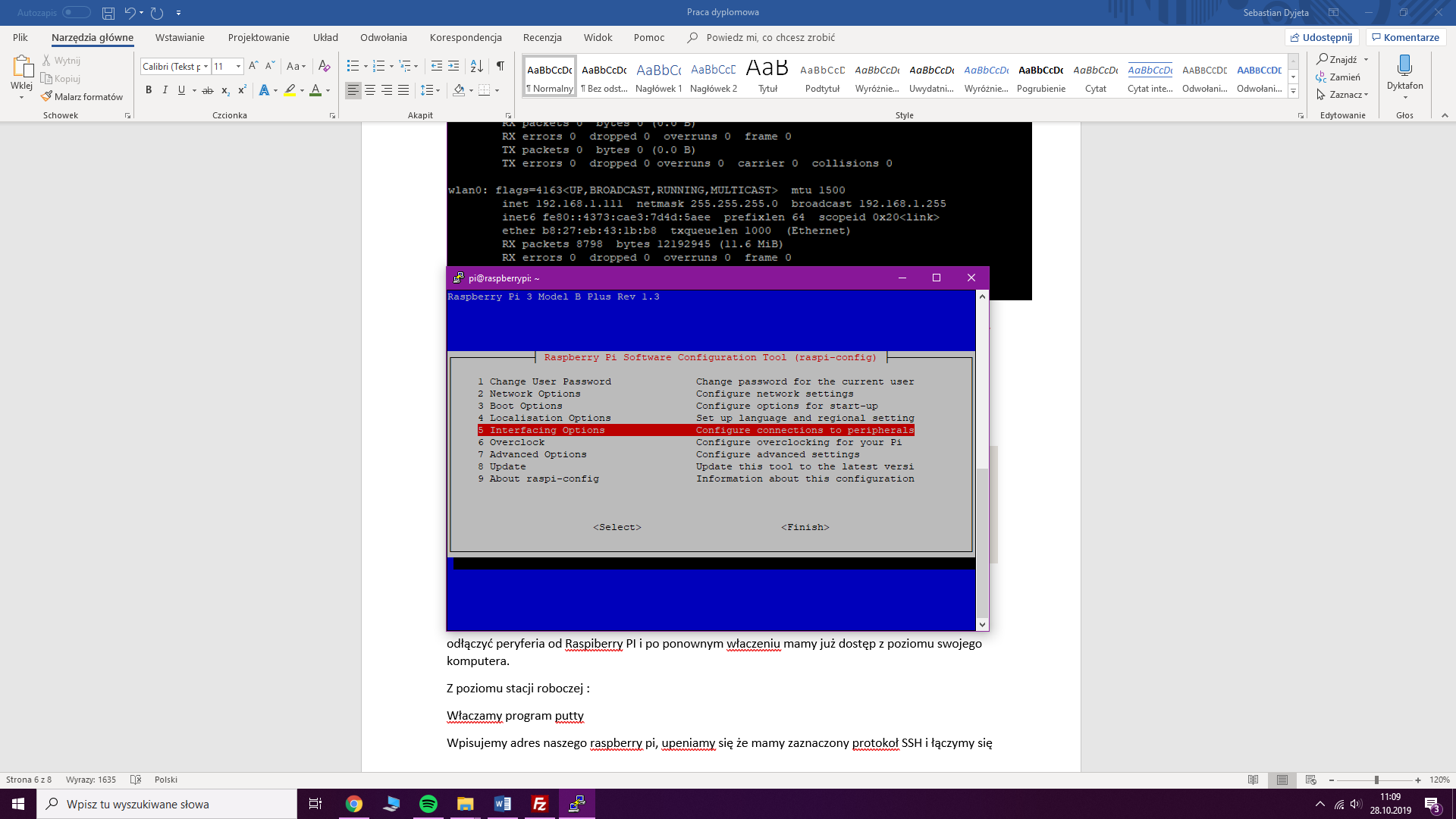
w miejsce ip\_address wpisujemy preferowany statyczny adres IP. Pamiętajcie o tym że adres musi znajdować się w tej samej podsieci co wasza stacja robota. Przypominam że by sprawdzić w jakiej podsieci jesteśmy powinniśmy w konsoli wpisać

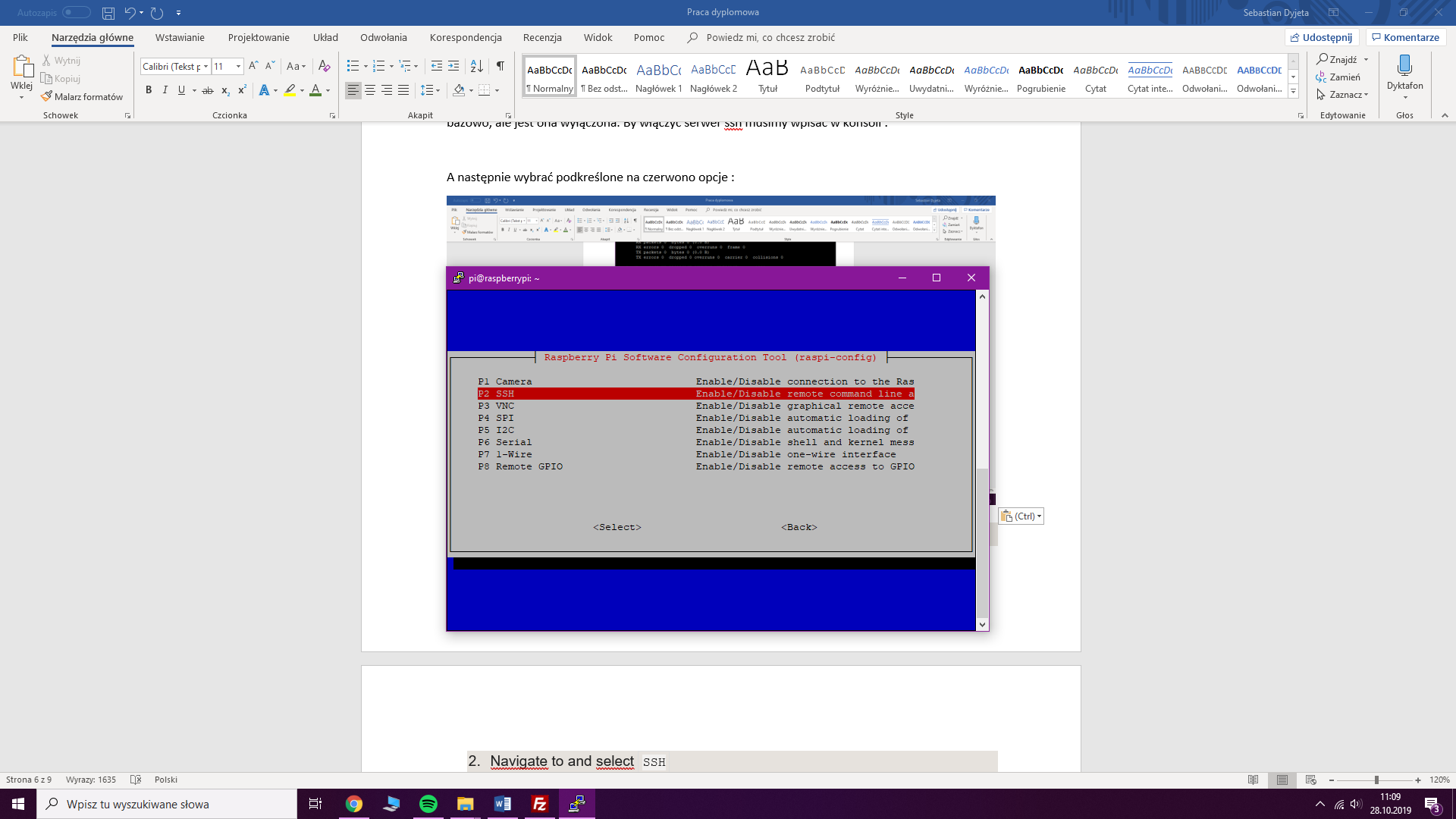
i wyszukać interesujący nasz interface. 

Kolejną rzeczą którą musimy zrobić jest włączenie serwera SSH, Raspberry Pi ma taką funkcjonalność bazowo, ale jest ona wyłączona. By włączyć serwer SSH musimy wejść do programu konfiguracyjnego

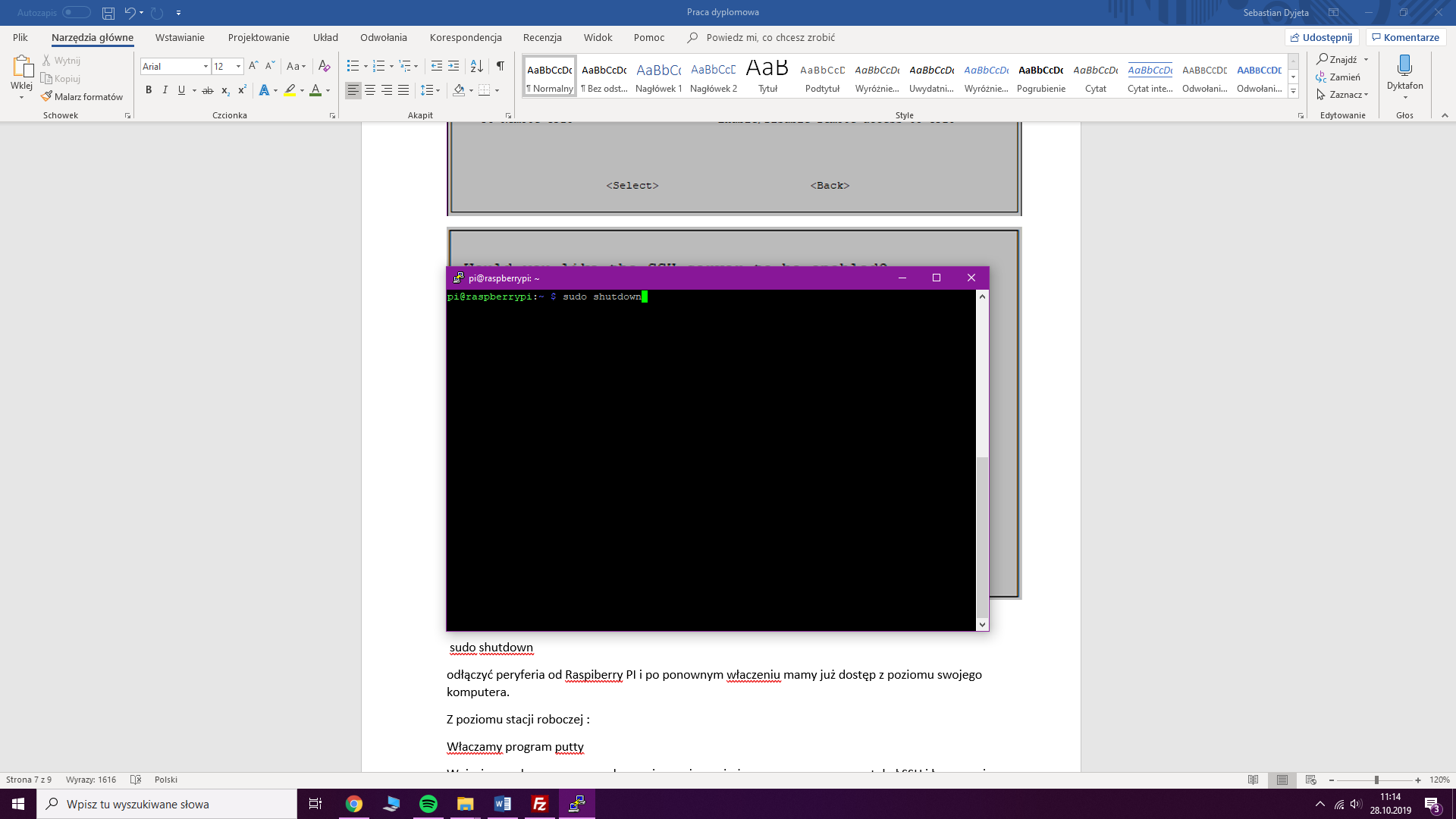


A następnie wybrać podkreślone na czerwono opcje :

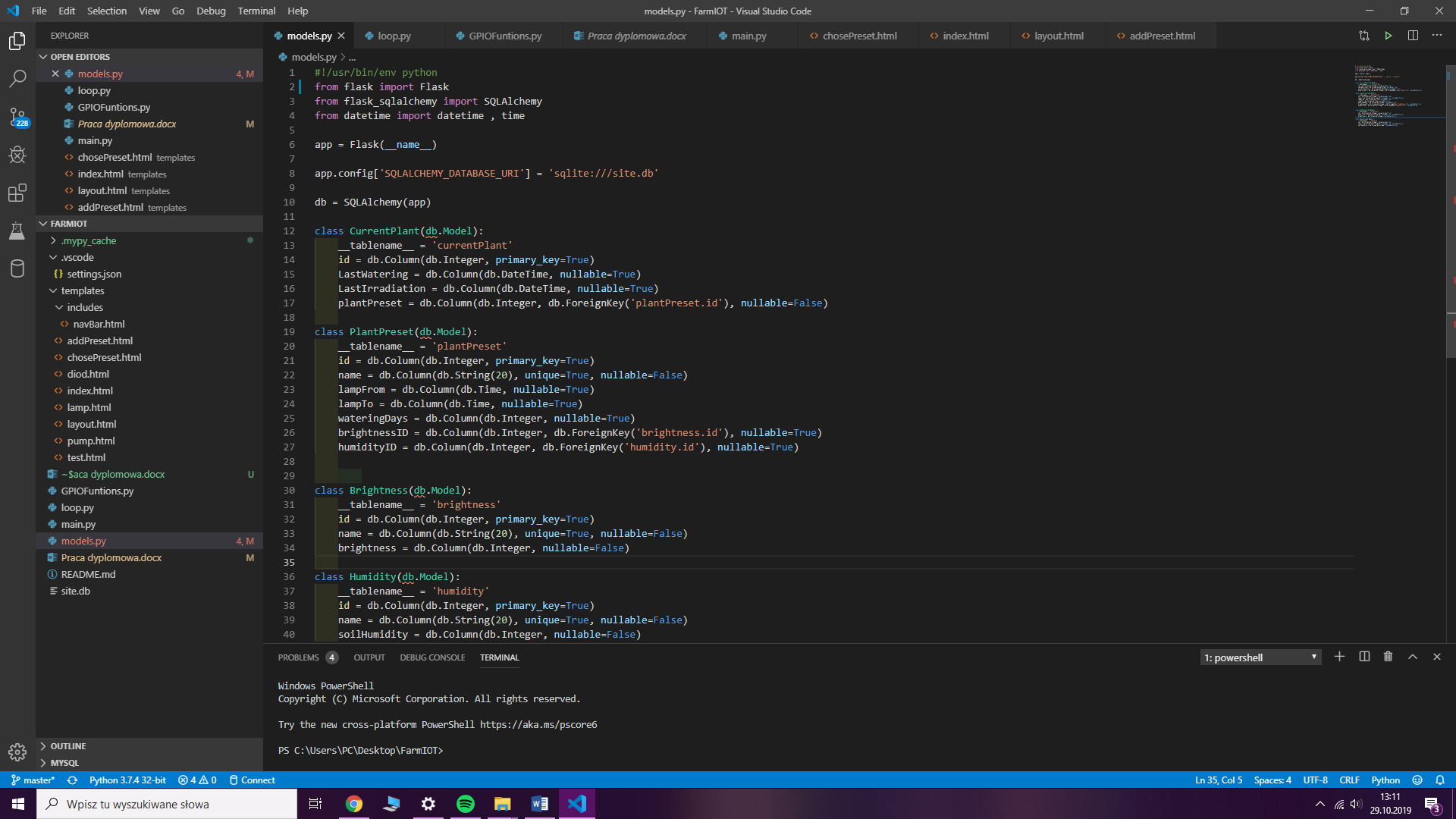




Teraz zostało nam wpisać w konsoli :

by wyłączyć Raspberry Pi. Teraz należy odłączyć peryferia od Raspberry Pi i po ponownym włączeniu mamy już dostęp do urządzenia z poziomu swojego komputera.

* + 1. **Połączenie z bazą danych**

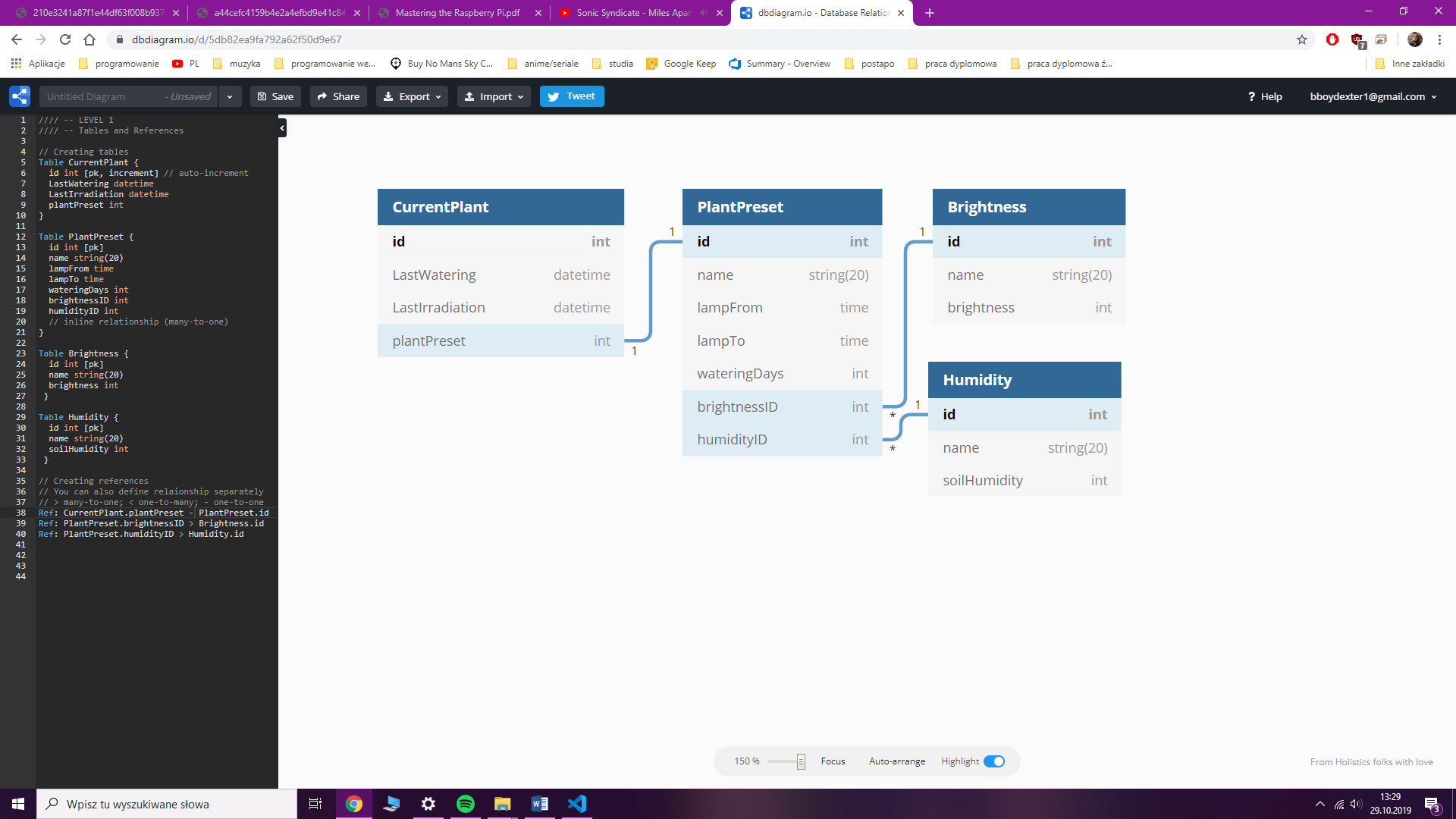


W tym przykładzie możemy zobaczyć dlaczego tak dużo osób lubi język Python. Kilka linii kodu i już nasza baza jest podłączona. Kodu co prawda jest mało ale integrujemy tutaj aż 3 komponenty czyli :

* Flask
* SQLAlchemy
* SQLite

**Utworzenie bazy danych :**

Schemat :



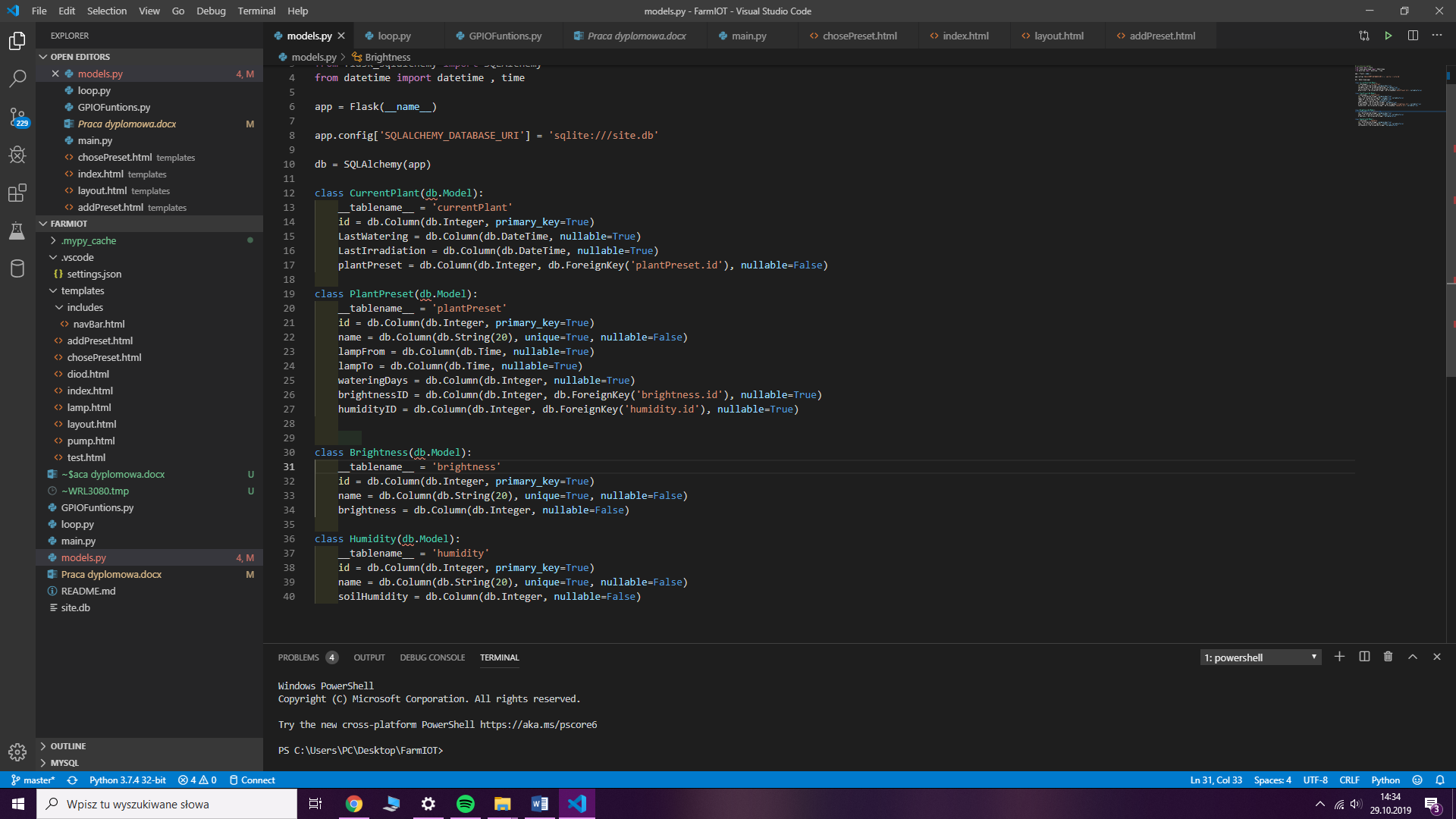
CurrentPlant – tabela która przechowuje informacje odniesienie aktualnie wybranych ustawień oraz ostatnie daty podlania oraz naświetlenia rośliny.

PlantPreset – ustawienia systemu. Mogą być dodawane przez użytkownika.

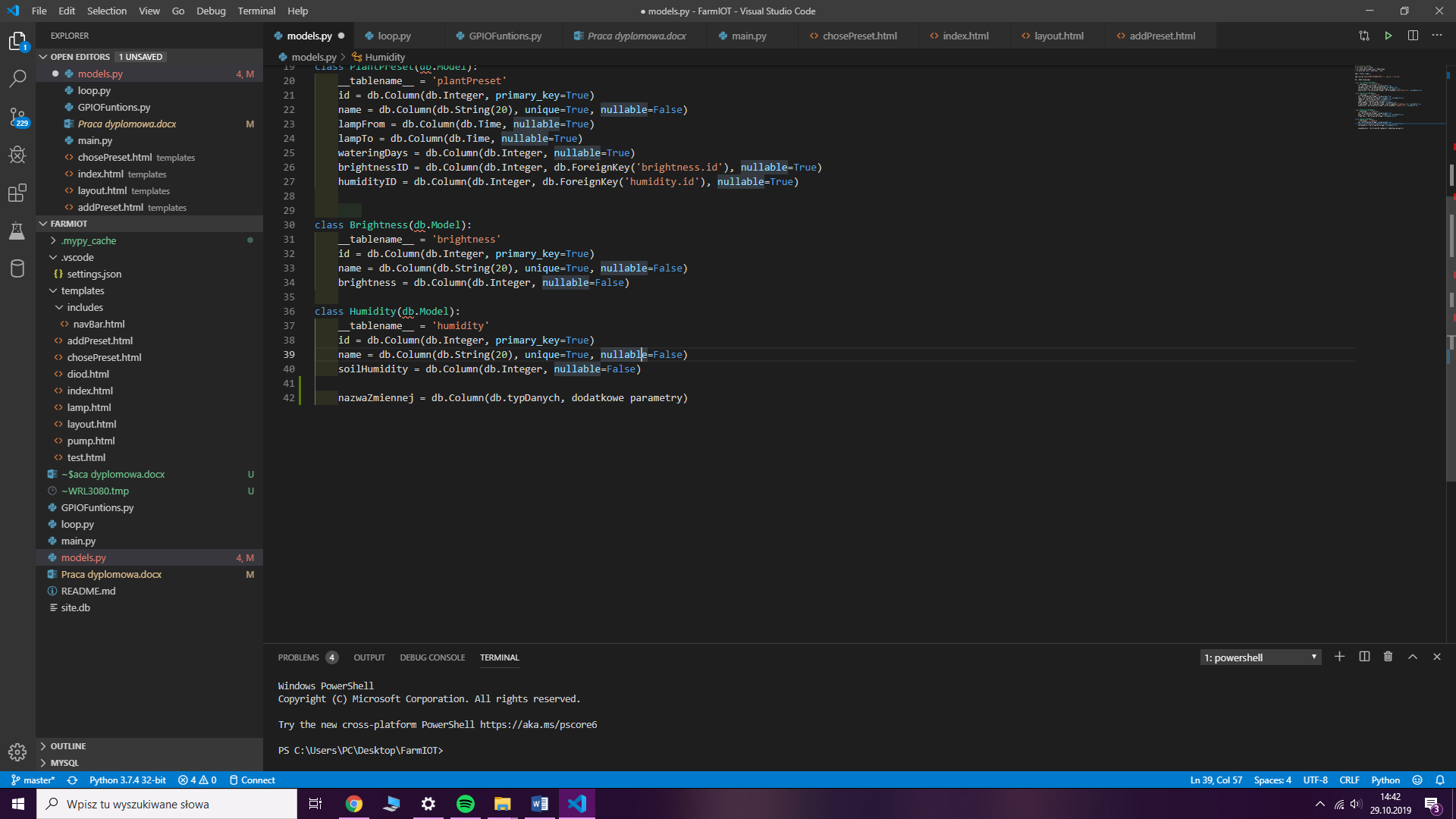
Brightness – tablica zawierająca docelowe wartości nasłonecznienia w formacie zrozumiałym dla systemu oraz przyjazną dla użytkownika nazwę (np. „niewielkie nasłonecznienie”).

Humidity – tablica zawierająca docelowe wartości wilgotności gleby w formacie zrozumiałym dla systemu oraz przyjazną dla użytkownika nazwę (np. „wysoka wilgotność”).

**Implementacja modeli**



Każdy model żeby być widziany przez SQLAlchemy musi dziedziczyć po klasie Model oraz posiadać kolumny opisane w formacie :



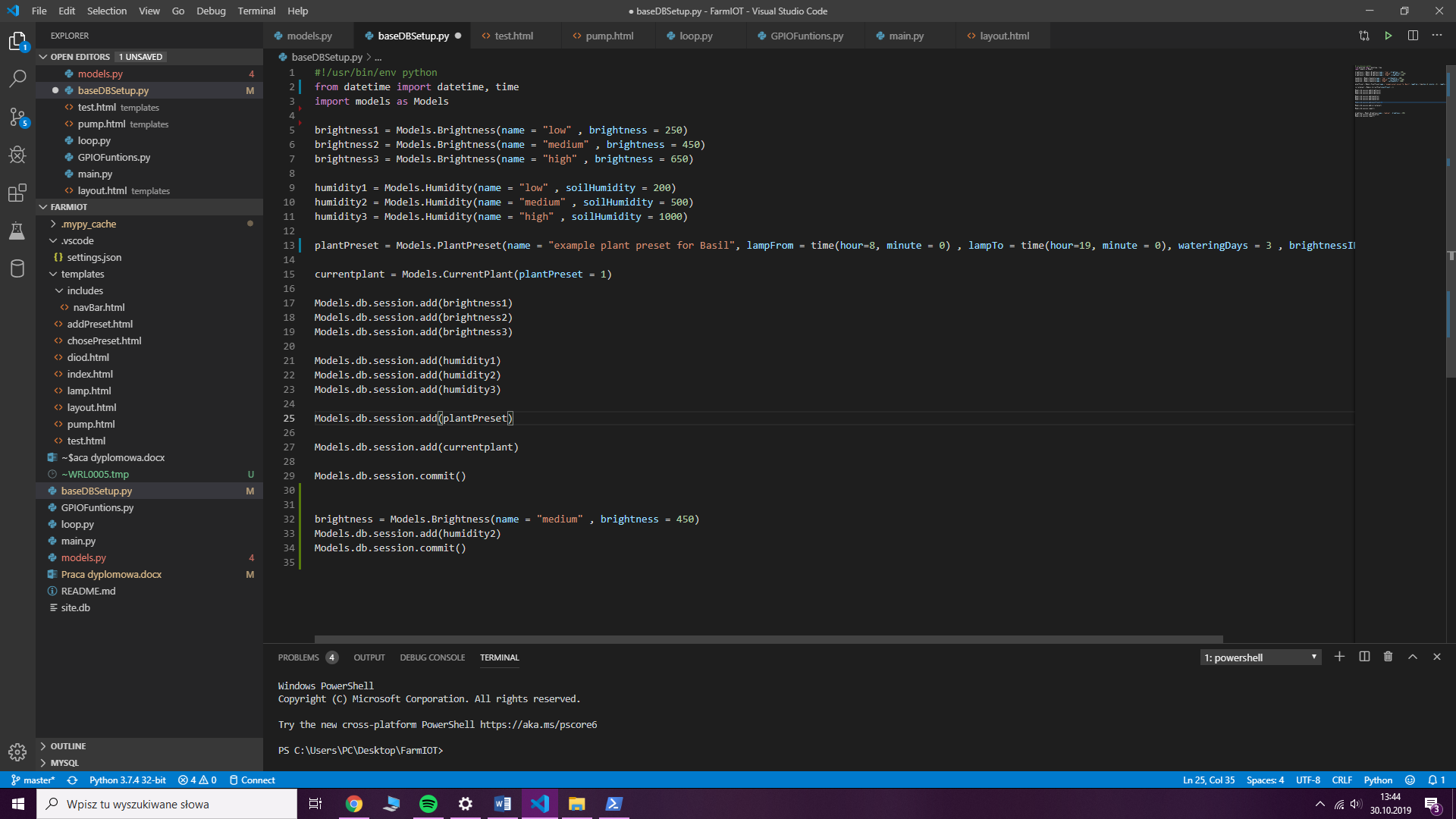
Przez dodatkowe parametry mam namyśli auto inkrementacje, indeksowanie itp.

Zmienna \_\_tablename\_\_ nie jest konieczna ale pozwala na nam nadanie dowolnej nazwy dla tabeli niezależnie od nazwy klasy modelu, domyślną nazwą dla PlantPreset była by nazwa plant\_preset.

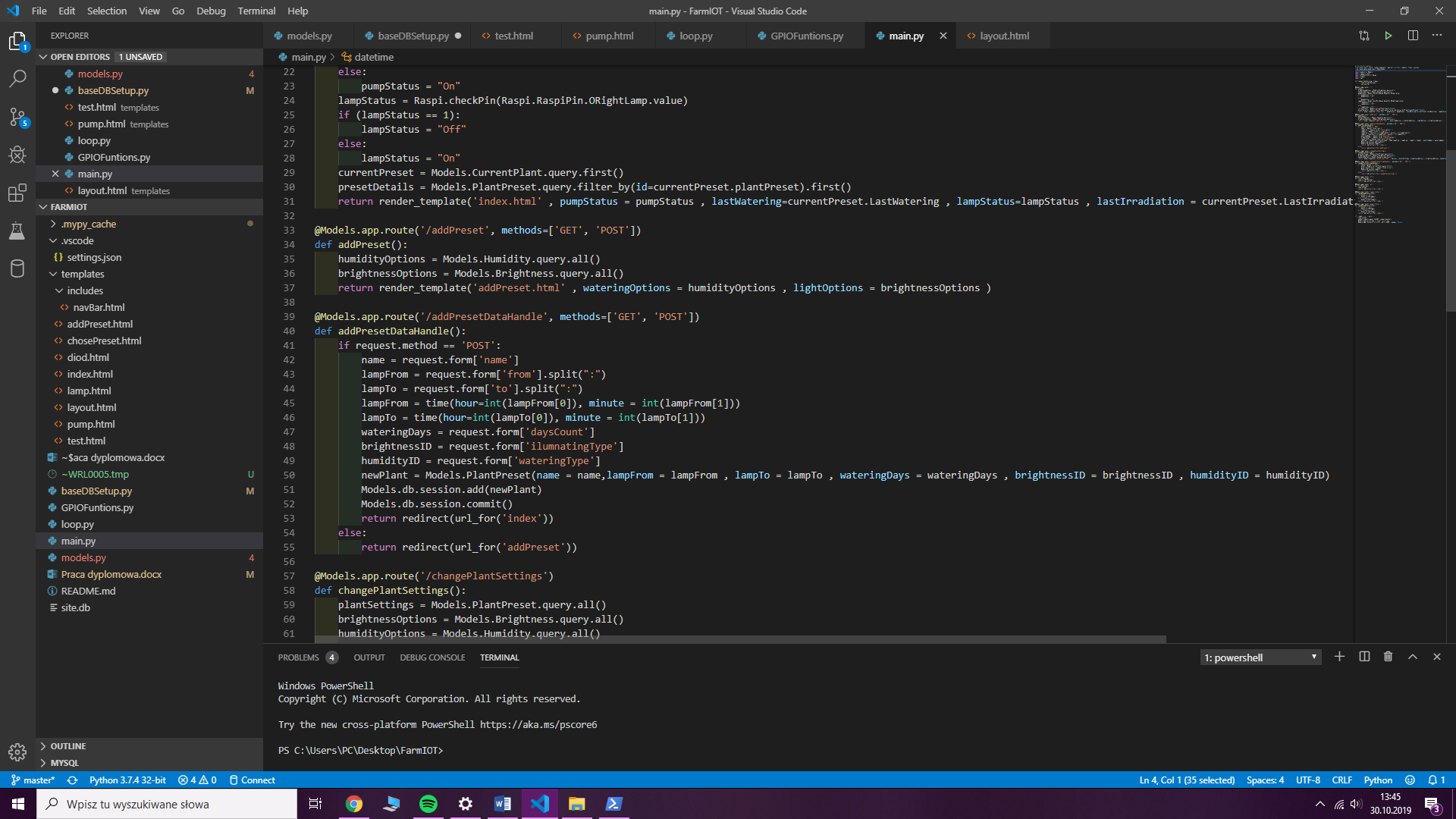
**CRUD (create,read,update,delete) w SQLAlchemy**

Każda baza danych udostępnia podstawowe 4 funkcjonalności znane jako CRUD czyli :

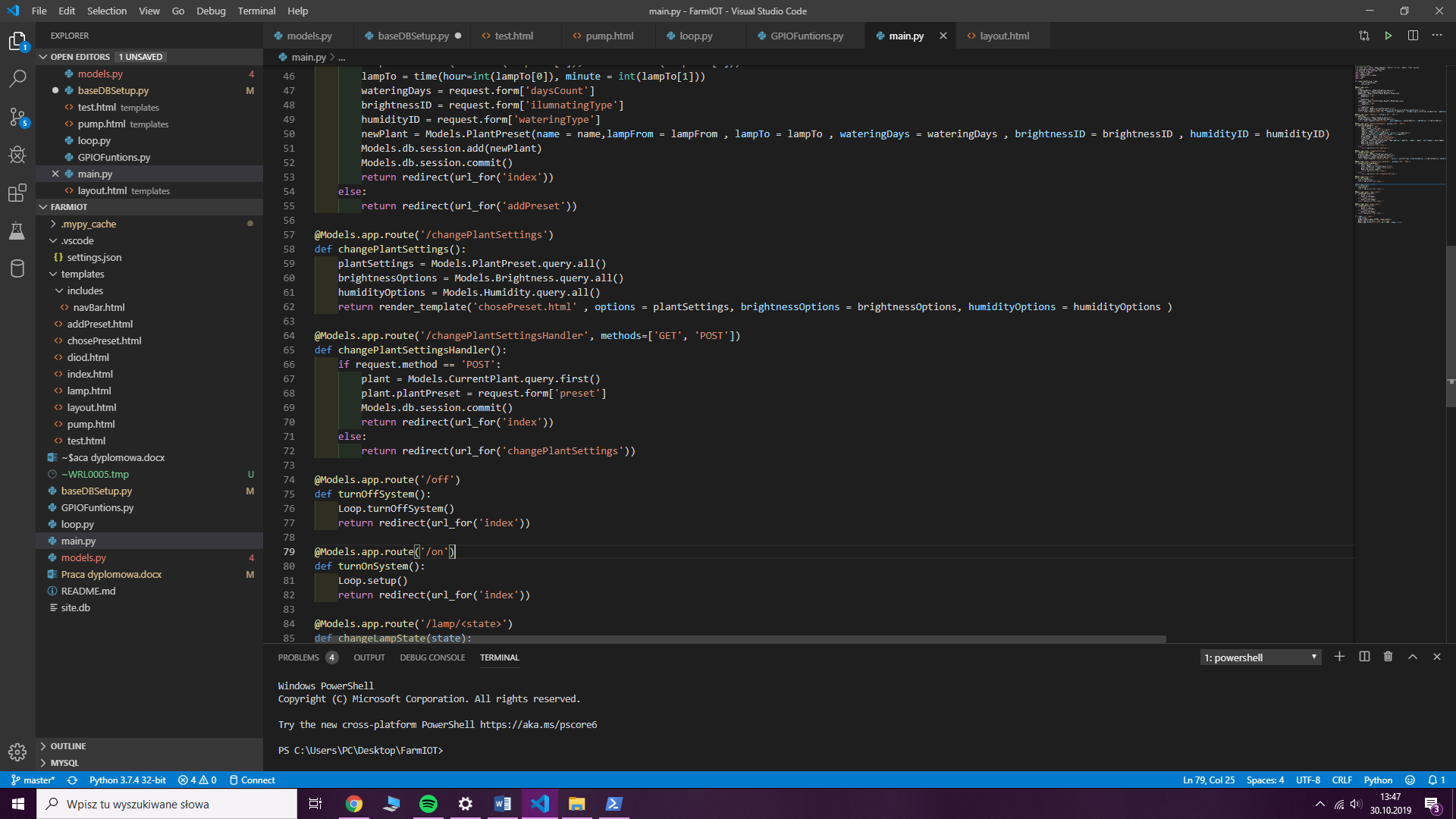
* Dodanie do bazy



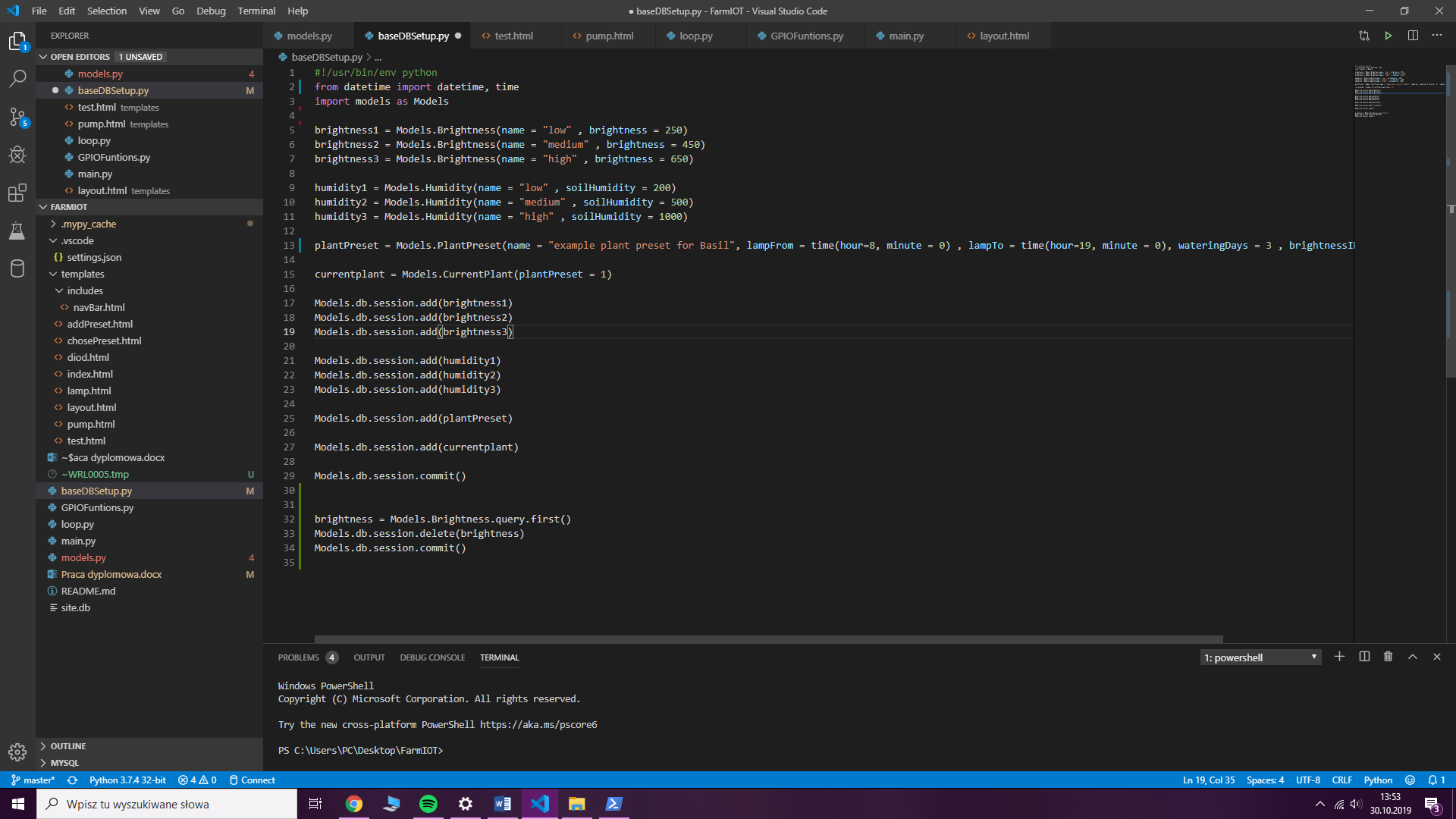
* Odczyt z bazy



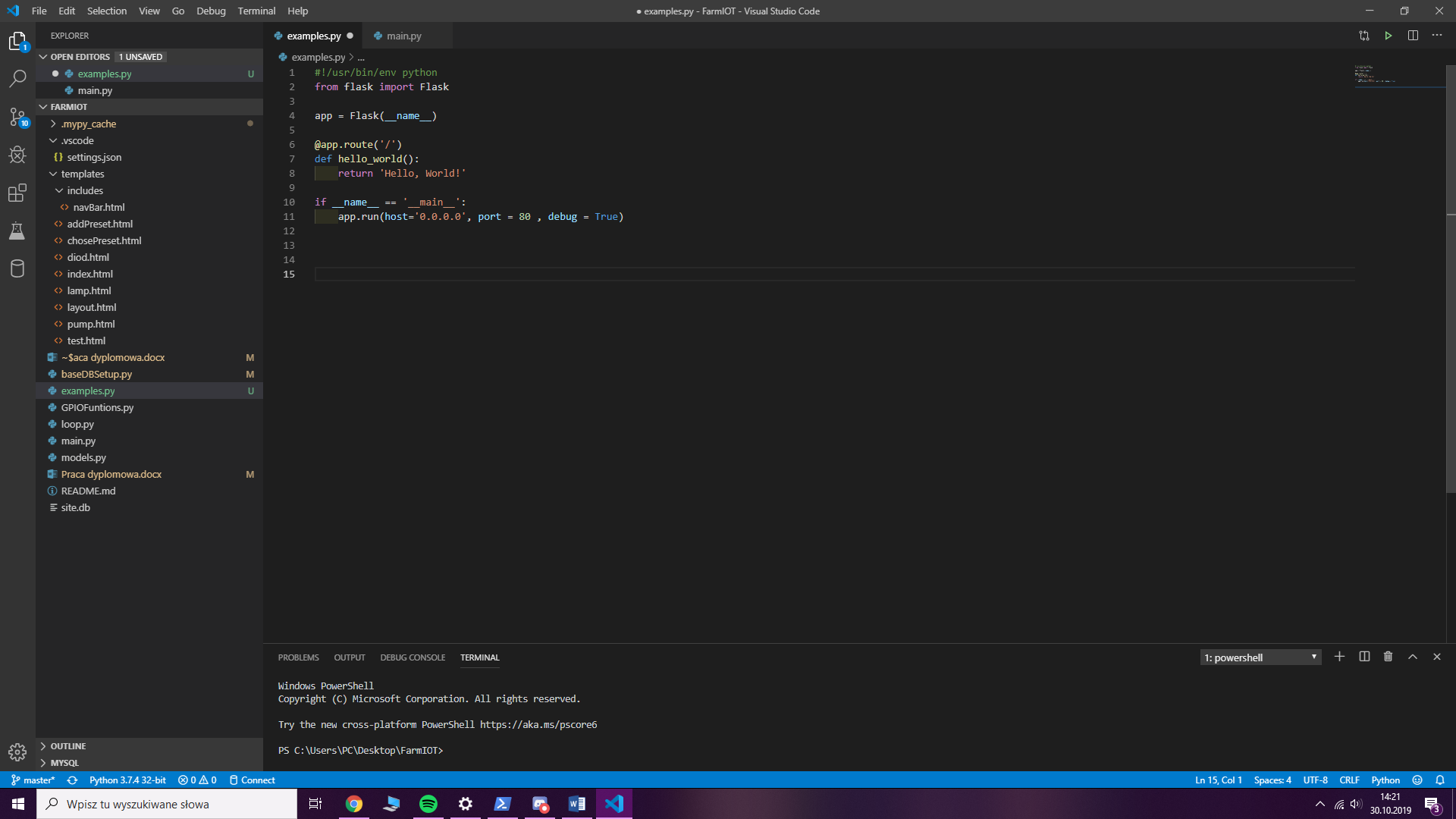
* Edytowanie danych w bazie



* Usuwanie z bazy

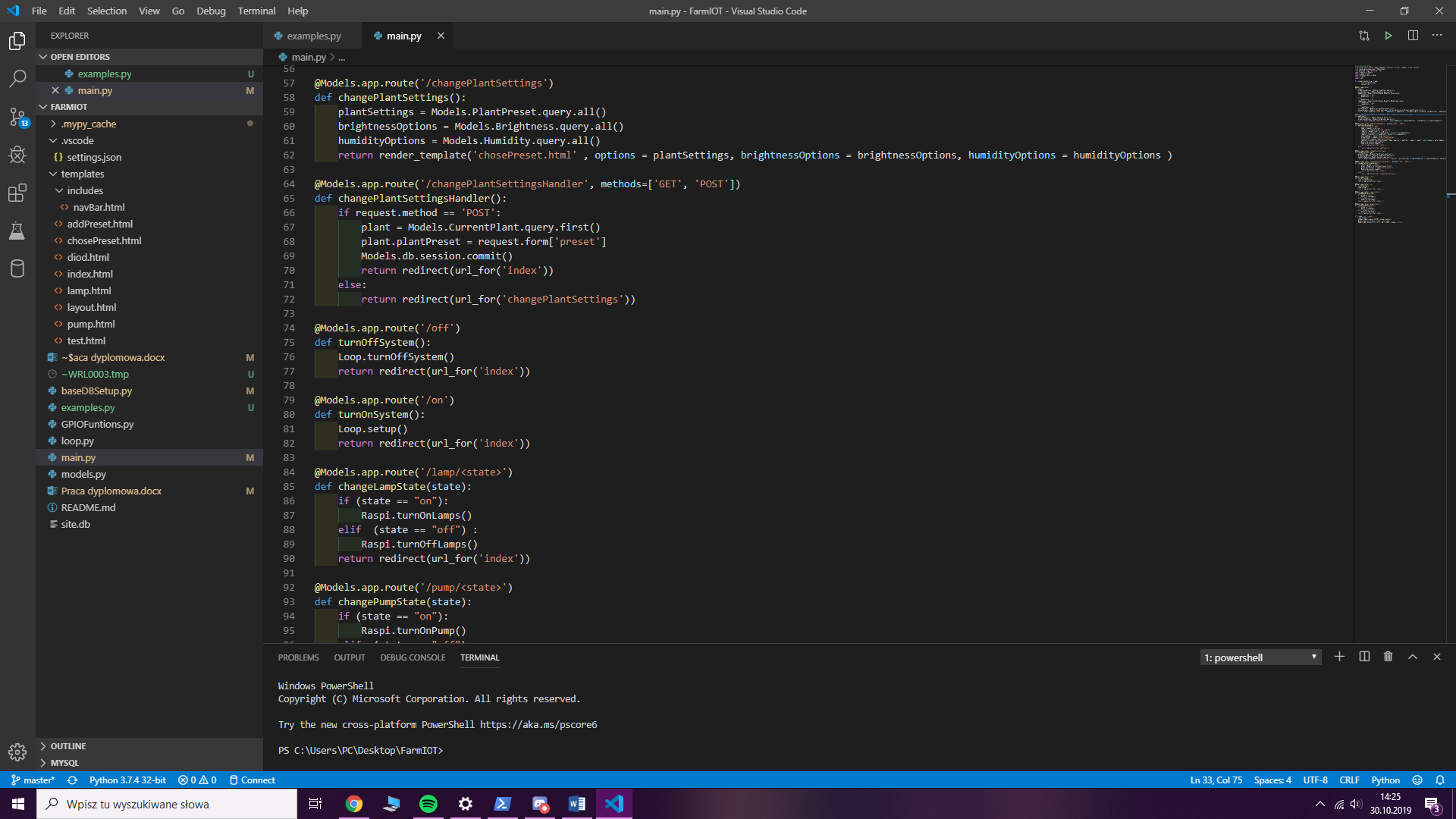


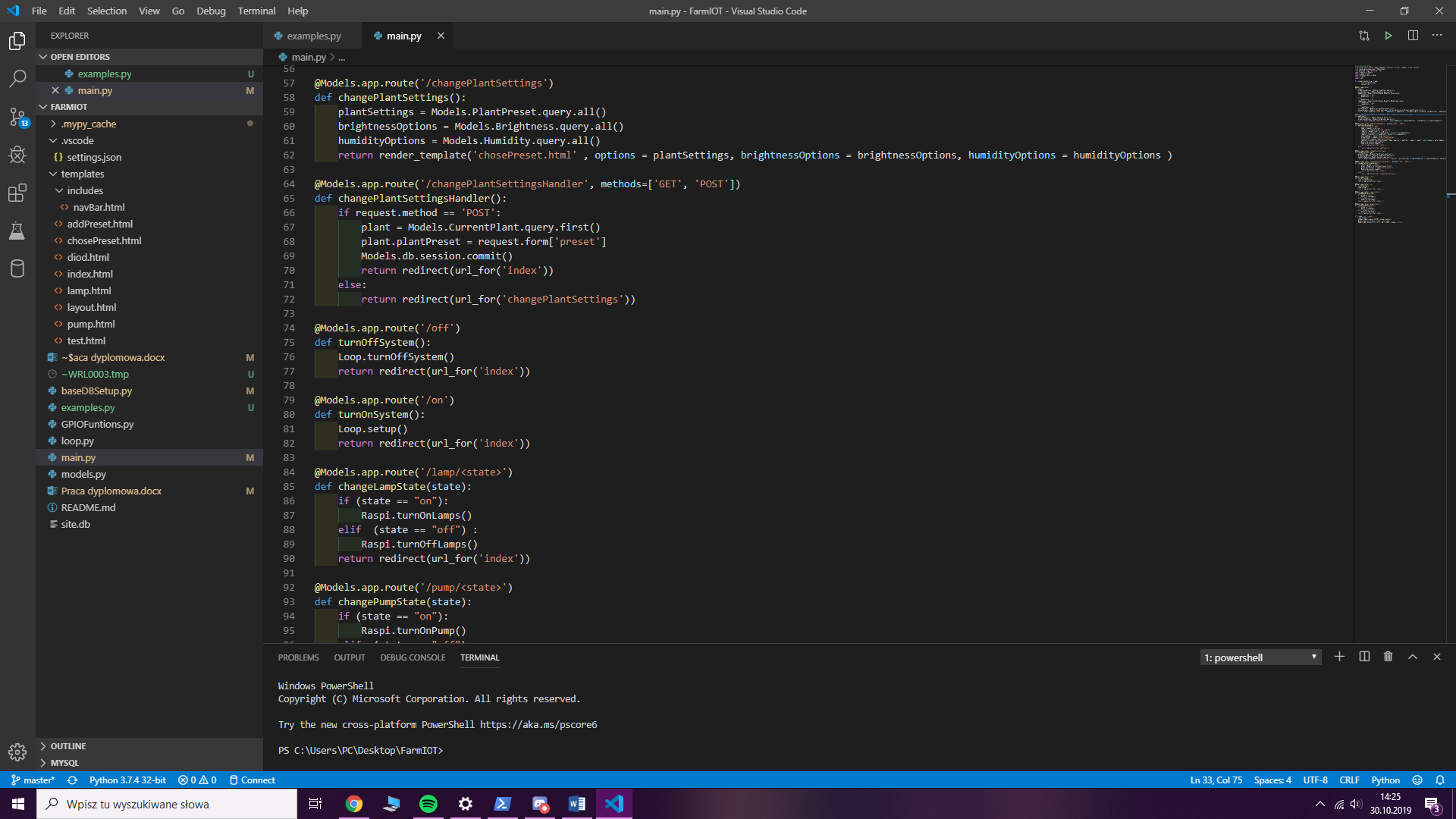
* + 1. **API**



Na przykładzie powyżej widzimy najprostsze API w frameworku Flask, linii 6-7 dodaje nam bazowy endpoint (nasz adres bez żadnej podstrony) który zwróci nam „Hello, World!”. Linia 11 mówi o tym że

* Host=’0.0.0.0’ – strona jest dostępna na adresie aktualnej maszyny
* Port = 80 – strona jest dostępna na porcie 80 czyli bazowym porcie http
* Debug = True – mówi że serwer jest włączony w trybie Debug co znaczy że z każdym żądaniem pliki są wgrywane od początku więc każda nasza zmiana w kodzie będzie widoczna bez resetu serwera. Należy pamiętać by finalnie opcja ta była ustawiona na False [co to daje]



W Flasku jak w przypadku każdej aplikacji o architekturze RESTowej tworzymy tak zwane endpointy do których wysyłane są żądania HTTP, endpoint składa się z adresu, parametrów oraz metod. Jeden URL może mieć do siebie przypisane kilka endpointów w zależności od użytej metody, warto pamiętać że domyślnie przeglądarka internetowa wysyła rządzenie GET

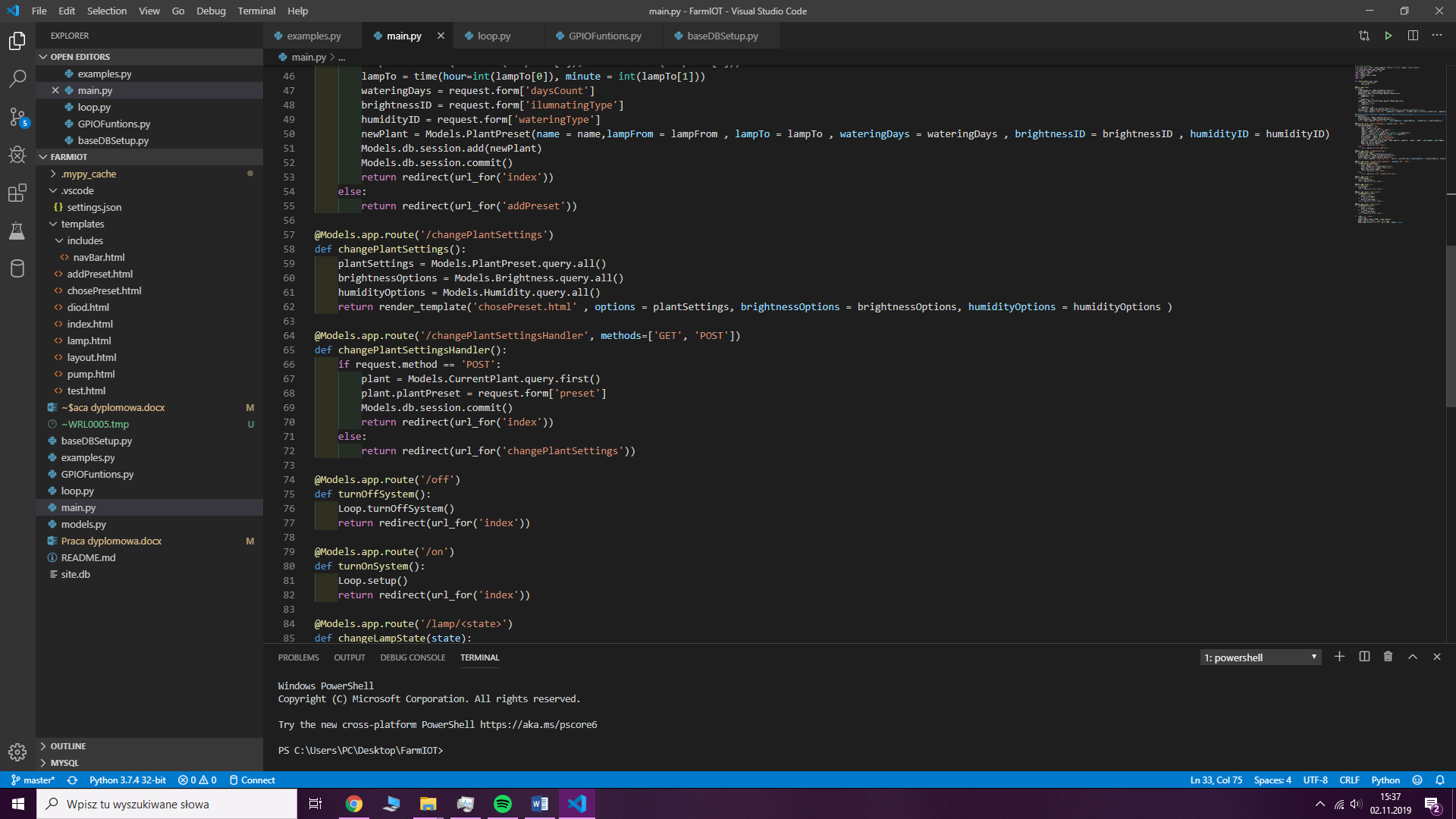
Flask pozwala nam też na obsługę zmiennych przesłanych przez użytkownika w URL w tym przypadku „lamp” jest czytane jako podstrona a następna rzecz po „/” będzie odczytana jako parametr. W ten sposób przesyła się proste oraz niewrażliwe dane. Bardziej złożone dane lub te wrażliwe wysyłamy w ciele zapytania.

Po pozytywnym rozpatrzeniu żądania wysyłamy zazwyczaj kod statusu 200 (kod sukcesywnego rozpatrzenia) oraz odpowiednią stronę.

Napisałem zazwyczaj ponieważ są odstępstwa od tych reguł nawet jeśli żądanie zakończone zostaje pozytywnie.

W Przypadku chęci usunięcia czegoś z bazy (metoda DELETE) standardową praktyką jest zwrócenie kodu 204 (brak zawartości).

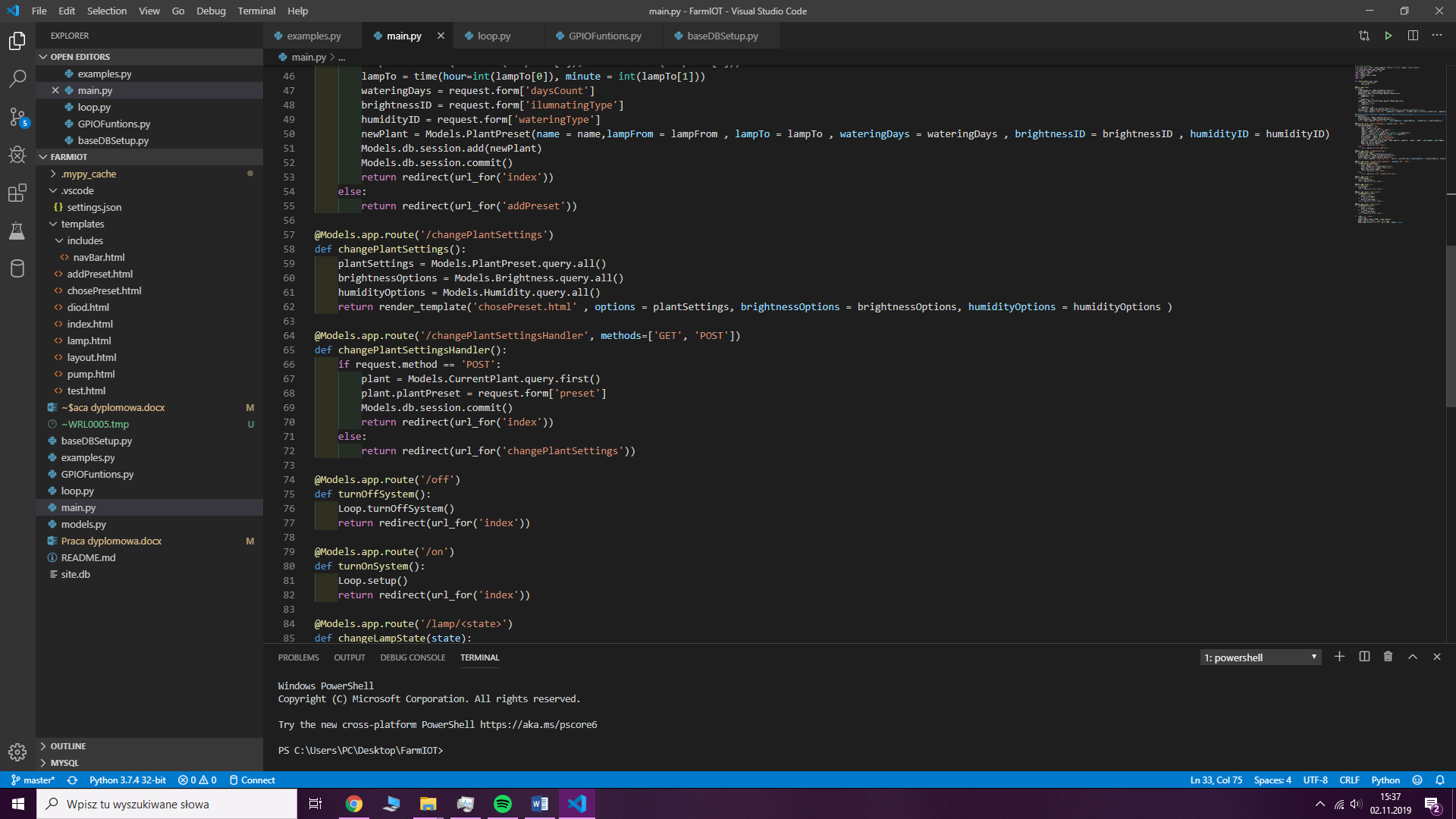
Zamiast wyświetlenia strony możemy też przekierować żądanie do innego endpointa



Co zwraca kod [sprawdzić jaki/ 300 ? ]

**Generowanie strony**

Gdy nasz endpoint kończy się w ten sposób



Wracamy użytkownikowi status 200 wraz z wygenerowaną stroną. Przebieg generowania strony jest następujący :

Endpoint odnosi się do pliku konkretnego pliku html który powinien zaczynać się od

{% extends 'layout.html' %}

Który dołącza do strony część wspólną dla wszystkich podstron

Możemy też doładować w dowolnym miejscu komponent

{% include 'includes/navBar.html' %}

* + 1. **Algorytmy użycia aktorów**

Nawadnianie

Zapisujemy dwa wydarzenia które będą odbywały się w okresie podanym przez użytkownika (co ile dni)

* Pierwsze wydarzenie odbywa się o 15:00 określonego dnia, wydarzenie to zapisuje kolejne wydarzenie które wykonuje się co sekundę i sprawdza ono wartość z czujnika wilgoci jeśli jest ona mniejsza niż optymalna to włączana jest pompa.
* Drugie wydarzenie odbywa się o 15:15 określonego dnia, wydarzenia to wyłącza pompę oraz wydarzenie zapisane w poprzednim kroku.

Oświetlenie

Tak samo jak w nawadnianiu zapisujemy dwa wydarzenia które działają podobnie.

* Pierwsze wydarzenie zapisywane jest na godzinę w której roślina powinna być wystawiona na działanie promieni słonecznych, wydarzenie to zapisuje kolejne wydarzenie które wykonywane jest co 5 minut i sprawdzana czy średnia wartość na fotorezystorach jest odpowiednia dla ustawionej w systemie rośliny jeśli nie włączane są lampy, jeśli mamy odpowiednią wartość na czujnikach wtedy lampy są wyłączane.
* Drugie wydarzenie zapisywane jest na godzinne o której roślina powinna przestać być wystawiona na działanie promieni słonecznych, wydarzenie to wyłącza lampy oraz wydarzenie nadzorujące zapisane w poprzednim punkcie.
  + 1. Wykonywanie funkcji w zaplonowanym czasie

Do wykonywania zadań w danym okresie czasu wykorzystałem bibliotekę schedule stworzoną przez

Daniela Badera. Biblioteka ta używa builder pattern do konfiguracji, co pozwala zapisywać nam do planera wszelkie funkcje, bądź inne wykonywalne obiekty (callable) by wkonywały się w wybranych prze znas okresach czasu.

[napisać coś o builder pattern / callable / problemi z tym że shuduler blokuje proces więc trzeba było odpalić wątek w jezyku skryptowym]

**Testy API**

Do stworzenia automatycznych testów API w Postamnie tworzymy tzw. kolekcje, kolekcja ma swoje zmienne oraz listę żądań. Każde żądanie zawiera :

* URL
* nagłówek
* ciało (może być puste)
* metodę
* testy

żądanie zawsze wywoływane są w tej samej kolejności ponieważ niektóre zmienne mogą być zmieniane przez odpowiedz na żądanie. Dobrze jest mieć testy każdego z endpointów z dwóch powodów

* po zmianie kodu od razu wiemy gdy coś nie działa
* jest to swoje rodzaju dokumentacja jako że poza samym spisem endpointów mamy też przykładowe dane wejściowe.

**Plany na rozwój**

Aktualnie kod nadzorujący jest integralną częścią serwera. W przyszłości zamierzam sprawić by program nadzorujący był niezależną częścią umieszczoną na platformie Arduino która będzie komunikować się z głównym serwerem który będzie zawierać ustawienia i z którym komunikować będzie się użytkownik. Pozwoli to na nadzorowanie większej ilości roślin przez jeden serwer, aktualnie serwer może obsłużyć do 5 roślin ze względu na ograniczoną liczbę pinów.

Chciałbym też dodać możliwości kontroli większej ilości parametrów takich jak : wilgotność powietrza, mieszanka powietrza. Ale wymaga to hermetycznego środowiska (co generuje koszty) oraz więcej wiedzy o danej roślinie.

**Publikacje**

**Strony** :

<https://www.sqlalchemy.org/>

[https://www.sqlitetutorial.net](https://www.sqlitetutorial.net/)

[https://www.techopedia.com](https://www.techopedia.com/)

<https://schedule.readthedocs.io/en/stable/>

**Książki** :

Samouczek HTML – Karol Wierzchołowski

Learning Python 5ed – Mark Lutz

Mastering the Raspberry Pi - Warren Gay