Spis treści

[Spis treści 1](#_Toc25069118)

[Wstęp 3](#_Toc25069119)

[Zakres pracy 3](#_Toc25069120)

[Teoria 3](#_Toc25069121)

[Programy 3](#_Toc25069122)

[Visual Studio Code 3](#_Toc25069123)

[FileZilla Client 3](#_Toc25069124)

[Putty 4](#_Toc25069125)

[Postman 4](#_Toc25069126)

[HTML 4](#_Toc25069127)

[Python 4](#_Toc25069128)

[Flask 4](#_Toc25069129)

[SQLAlchemy 4](#_Toc25069130)

[SQLite 5](#_Toc25069131)

[GIT 5](#_Toc25069132)

[Implementacja systemu 5](#_Toc25069133)

[Spis komponentów 5](#_Toc25069134)

[Opis komponentów 5](#_Toc25069135)

[Raspberry Pi 3 B+ 5](#_Toc25069136)

[Moduł przekaźników 6](#_Toc25069137)

[ADC (konwerter analogowo cyfrowy) 6](#_Toc25069138)

[Fotorezystor 6](#_Toc25069139)

[Czujnik wilgoci gleby 6](#_Toc25069140)

[Schemat układu 7](#_Toc25069141)

[SPI 7](#_Toc25069142)

[Konfiguracja serwera 8](#_Toc25069143)

[Konfiguracja dostępu 8](#_Toc25069144)

[Włączenie aplikacji przy starcie systemu. 9](#_Toc25069145)

[Utworzenie bazy danych 10](#_Toc25069146)

[Schemat bazy danych 10](#_Toc25069147)

[Dodanie bazy danych 10](#_Toc25069148)

[Połączenie z bazą danych 11](#_Toc25069149)

[CRUD (Create,Read,Update,Delete) w SQLAlchemy 11](#_Toc25069150)

[API 12](#_Toc25069151)

[Podstawy Web API 12](#_Toc25069152)

[Generowanie strony 12](#_Toc25069153)

[Widoki 14](#_Toc25069154)

[Choose preset – widok który wyświetla nam wszystkie wcześniej utworzone ustawienia i pozwala na wybranie aktualnego. Algorytmy użycia aktorów 15](#_Toc25069155)

[Naświetlanie 15](#_Toc25069156)

[Nawadnianie 16](#_Toc25069157)

[Wykonywanie funkcji w zaplonowanym czasie 16](#_Toc25069158)

[Wzorzec „Budowniczy” (builder pattern) 17](#_Toc25069159)

[Testy API 17](#_Toc25069160)

[Plany na rozwój 18](#_Toc25069161)

[Publikacje 18](#_Toc25069162)

[Strony 18](#_Toc25069163)

[Książki 19](#_Toc25069164)

[Artykuły 19](#_Toc25069165)

# Wstęp

Celem pracy jest stworzenie systemu, który automatyzuje proces dbania o roślinę oraz pozwala na dostosowanie parametrów przez stronę internetową lub Web API które pozwala zewnętrznej aplikacji na korzystanie z opisanego systemu. System nadzorujący ma za zadanie nawadniać roślinie oraz zapewnić jej naświetlenie w wyznaczonym czasie. Całość postawiona 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 projektu.

# 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 aplikacji 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. Co ważne edytor działa na wszystkich popularnych systemach operacyjnych więc możemy przenieść swoje ustawienia niezależnie od miejsca pracy.

### FileZilla Client

Jest darmowym oprogramowaniem do połączeń z serwerami FTP/FTPS/SFTP które pozwala na wymianę plików. W opisywanym projekcie 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.

### 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.

### Python

Python jest interpretowalnym, interaktywnym oraz obiektowo zorientowanym językiem programowania. Pozwala na korzystanie z wysoko poziomowych struktur danych takich jak listy, tablice słownikowe, moduły, klasy, wyjątki itd. Ma niesamowicie prostą i elegancką składnie lecz jest potężnym i uniwersalnym językiem programowania. Został zaprojektowany w 1990 roku przez Guido van Rossum. Jak wiele innych języków skryptowych jest darmowy nawet dla zastosowań komercyjnych i może być użyty na praktycznie każdym nowoczesnym komputerze. Kod Python`owy jest kompilowany automatycznie przez interpreter do nieżelaznego od platformy kodu bajtowego który jest wykonywany. Możemy wykonywać niemodyfikowane komponenty napisane w Python`nie w linux`się, Window`się, SunOS oraz OSF. Python z założenia jest modularny, kernel jest bardzo mały I może być rozrzedzony przez dodanie modułów rozszerzeń. Dystrybucja Python`a zawiera zróżnicowaną bibliotekę standardowych rozszerzeń (niektóre napisane w Python`ie inne w C lub C++) które pozwalają na rzeczy takie jak manipulacja string`ów, czy podobne do Perl`a wyrażenia regularne. [1]

Jako że Python jest językiem otwarto-źródłowym posiada dużą społeczność która dodaje nowe moduły, co często pozwala na budowanie dużych aplikacji w krótkim okresie czasu.

### 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. Django posiada bazowo duża ilość funkcjonalności co sprawia że używa więcej zasobów i wymaga znaczniej większej znajomości narzędzie ponieważ wiele rzeczy generuje automatycznie. Gdybym użył Django do opisywanego projektu, nie skorzystał bym z wielu funkcjonalności które oferuje, a Flask pozwolił mi na szybsze dobranie wymaganych komponentów co sprawia że system jest nie tylko bardziej przejrzysty, używa mniej zasobów oraz zajmuje mniej miejsca w pamięci.

### 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.

### SQLite

SQLite jest systemem do zarządzania bazami danych. Posiada on wbudowany silnik bazy danych i nie używa osobnego procesu serwera jak większość znanych baz danych. Niezależnie od złożoności bazy danych SQLite zapisuje wszystko w jednym pliku na dysku co nie tylko zapewnia łatwe przeniesienie systemu ale też ułatwia tworzenie kopi zapasowych.

Ważne cechy SQLite:

* Wieloplatformowość : można użyć go do aplikacji desktopwej, mobilnej lub w systemach osadzonych niezależnie od systemu operacyjnego.
* 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.

### 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 lub zobaczenie ostatnich zmian dodanych przez współpracownika. Każde pobrane repozytorium jest samodzielne i w razie awarii głównego, można w pełni odtworzyć z je z repozytoriów lokalnych.

# Implementacja systemu

## Spis komponentów

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

## 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)

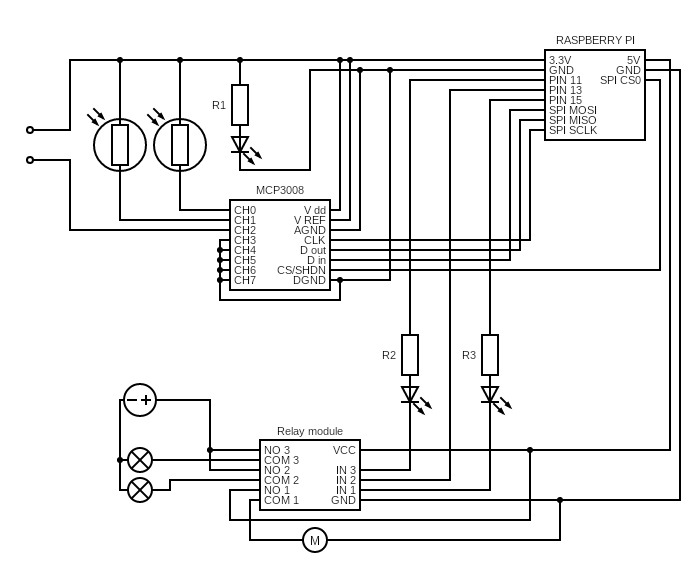
Jako że Raspiberry Pi nie jest w stanie sam odczytać wartości analogowych potrzebuje pomocy w postaci konwertera ADC. Konwerter analogowo cyfrowy to urządzenia które zmienia ciągły sygnał analogowy na przybliżony sygnał cyfrowy. Odbywa się to przez próbkowanie sygnału analogowego ze stała częstotliwością. 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



Bazowo Raspberry nie pozwala na odczyt danych analogowych, do tego potrzebujemy nie tylko konwertera ADC ale też połączenia przez konkretne piny protokołem SPI.

## SPI

Szeregowy interfejs urządzeń peryferyjnych (Serial Peripheral Interface) jest protokołem komunikacji używanym do przesyłania danych pomiędzy mikro-komputerami i peryferiami.   
SPI używa czterech osobnych połączeń do komunikacji z docelowym urządzeniem, te połączenia to :

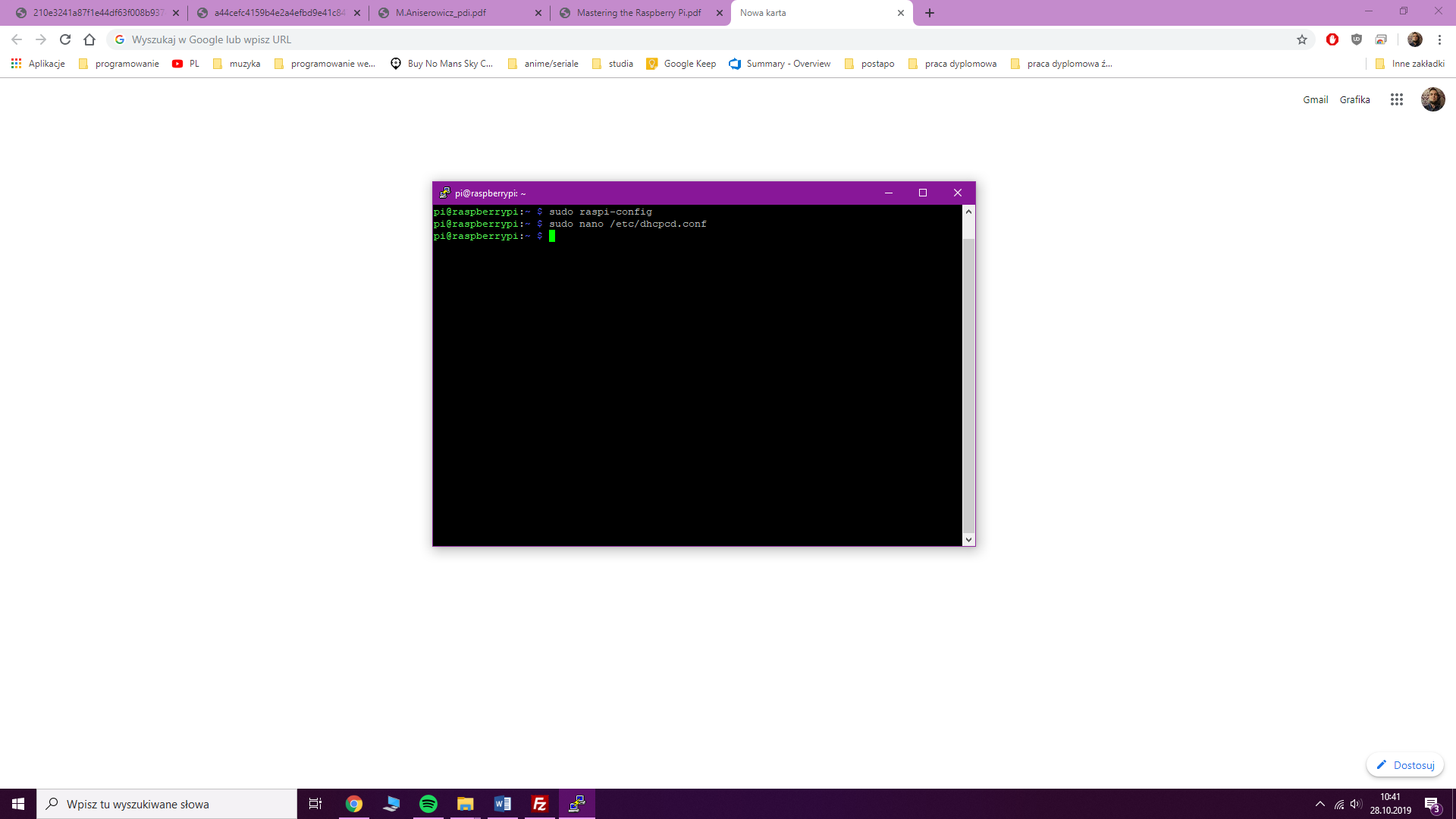
* Zegar (CLK – serial clock) – pin zegara zbiera impulsy o regularnej ustalonej częstotliwości. Dla ADC zegar wysyła impuls bazujący na wzroście krawędzi, czyli przejściu z niskiego do wysokiego napięcia.
* (MISO – Master Input Slave Output) – pin używany przez Raspberry PI do odbioru danych z urządzenia, dane są zbiera przy każdym impulsie zegara.
* (MOSI – Master Output Slave Input) – pin używany przez Raspberry PI do wysyłania danych. Dane również wysłane dopiero przy impulsie zegara.
* (CS – Chip Select) – wybór która urządzenie SPI jest w użyciu. Ponieważ kilka urządzeń może dzielić CLK, MOSI i MISO lecz tylko aktualnie aktywne urządzenie z sygnałem niskim jest brane pod uwagę.

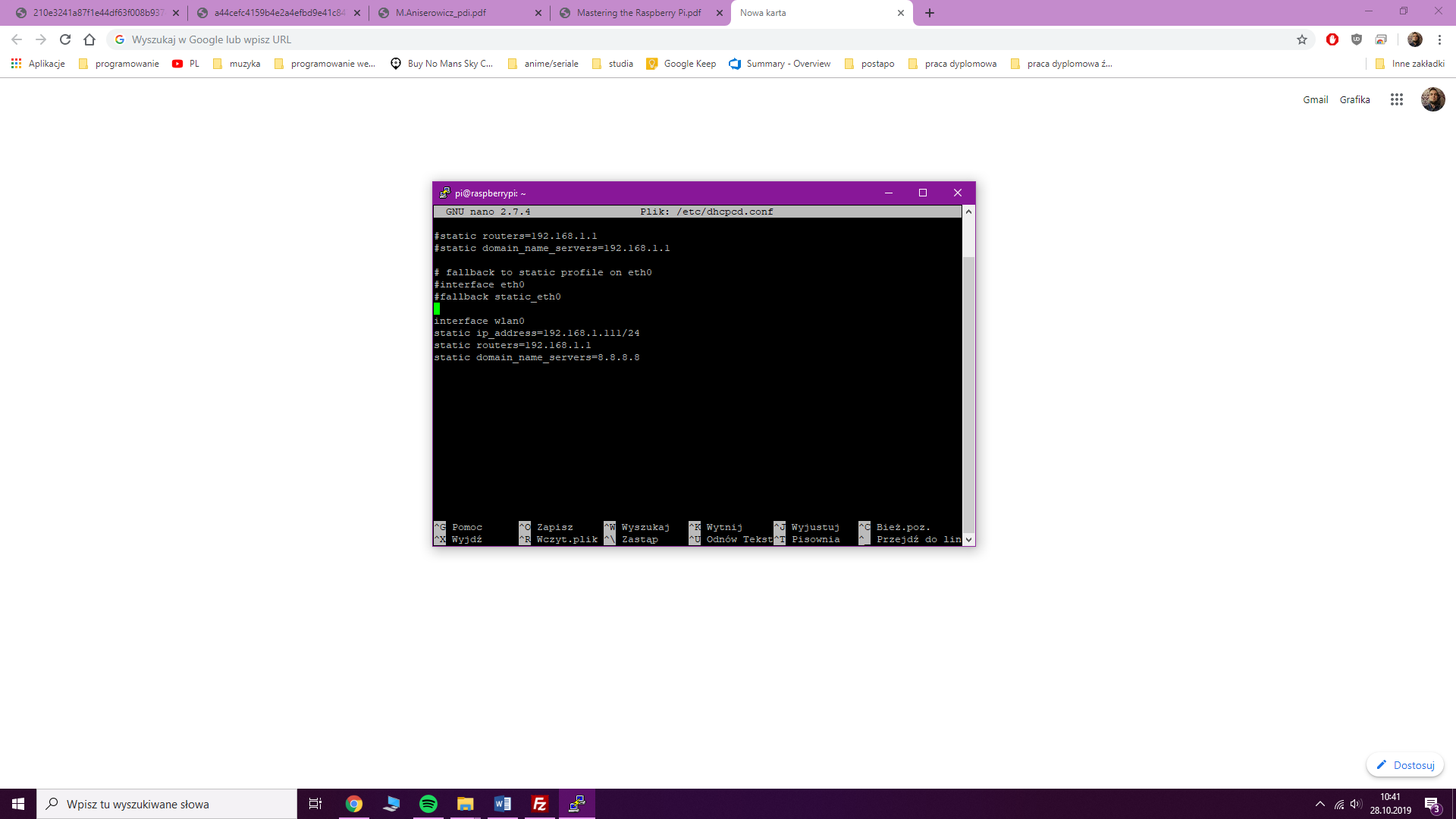
## Konfiguracja serwera

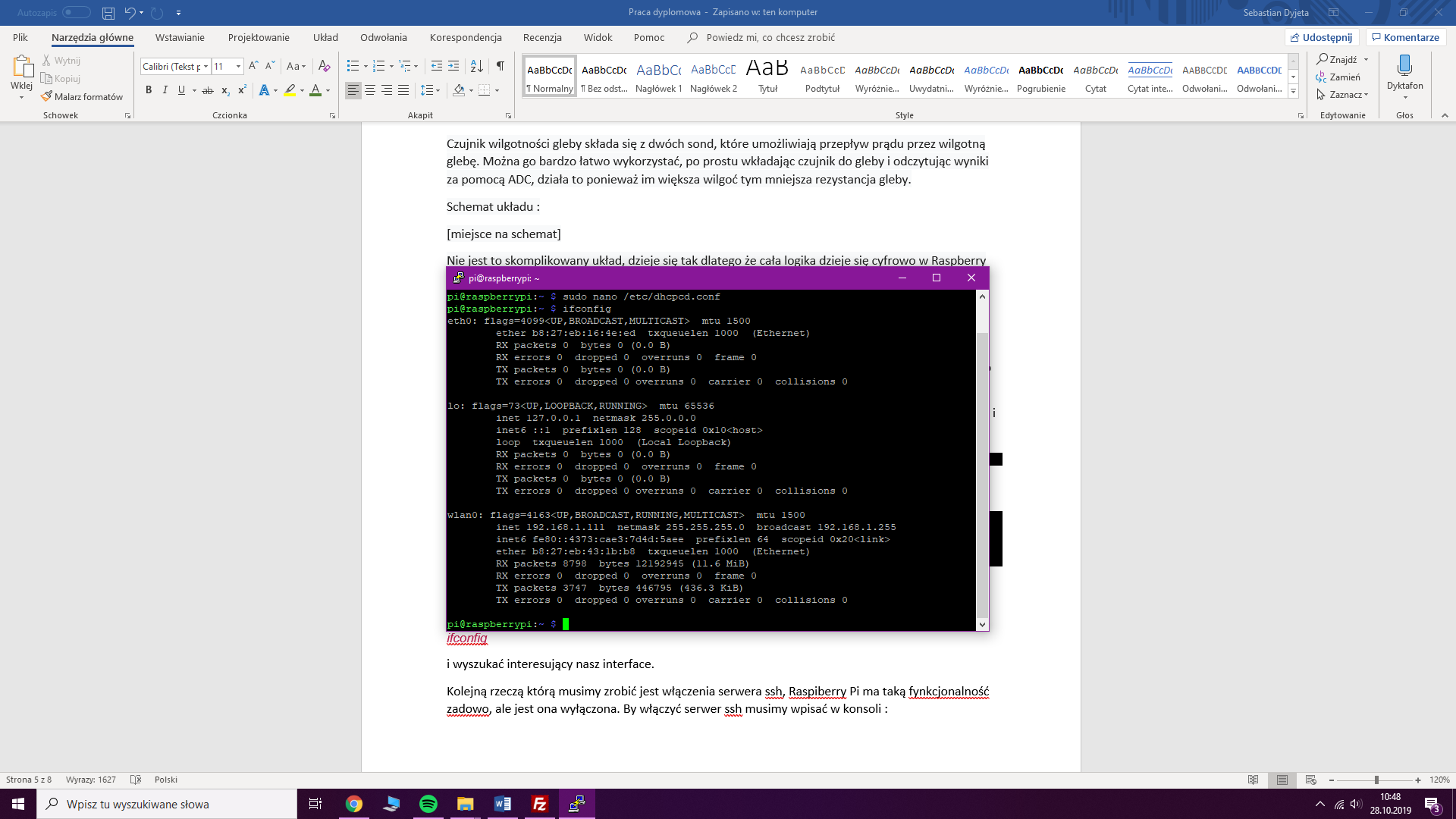
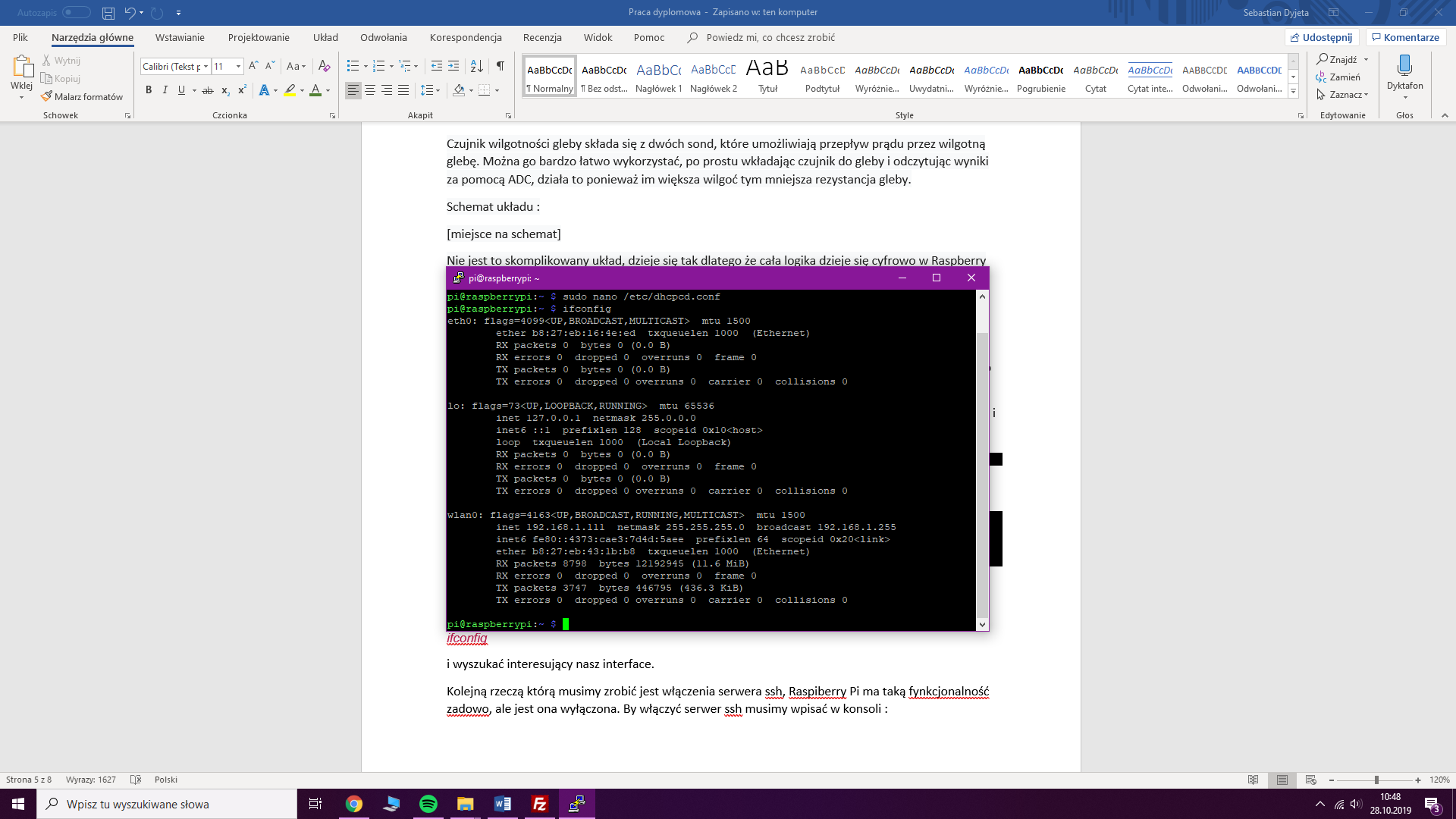
### Konfiguracja dostępu

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.

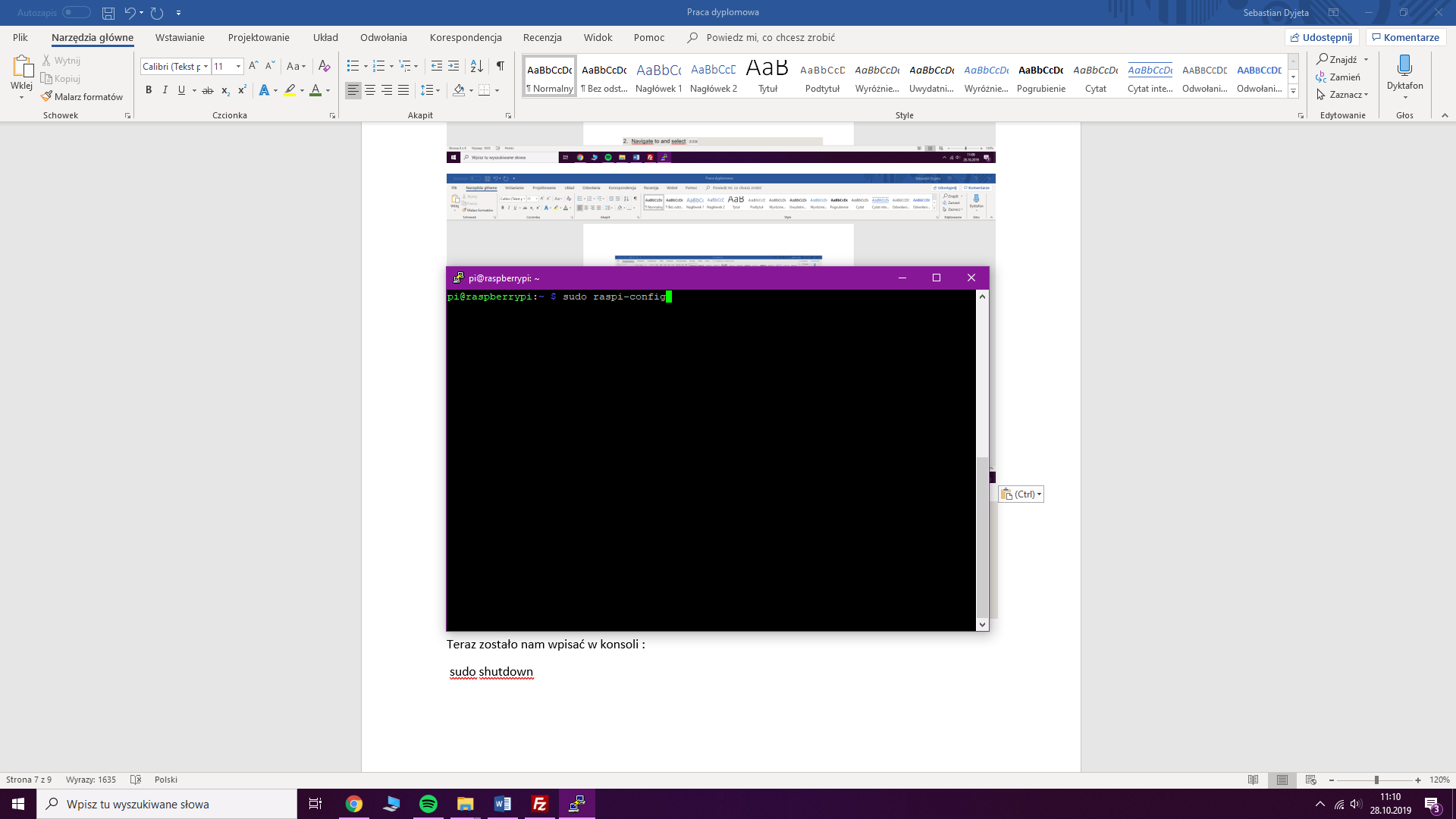
Zacznijmy od ustawienia statycznego adresu IP. Można to zrobić na kilka sposobów, osobiście preferuje zrobić to w pliku konfiguracyjnym na Raspberry Pi ponieważ wtedy naszą konfiguracje zabieramy wszędzie tam gdzie zabierzemy nasz system, co by nie miało miejsca gdybyśmy ustawili statyczny adres IP na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

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

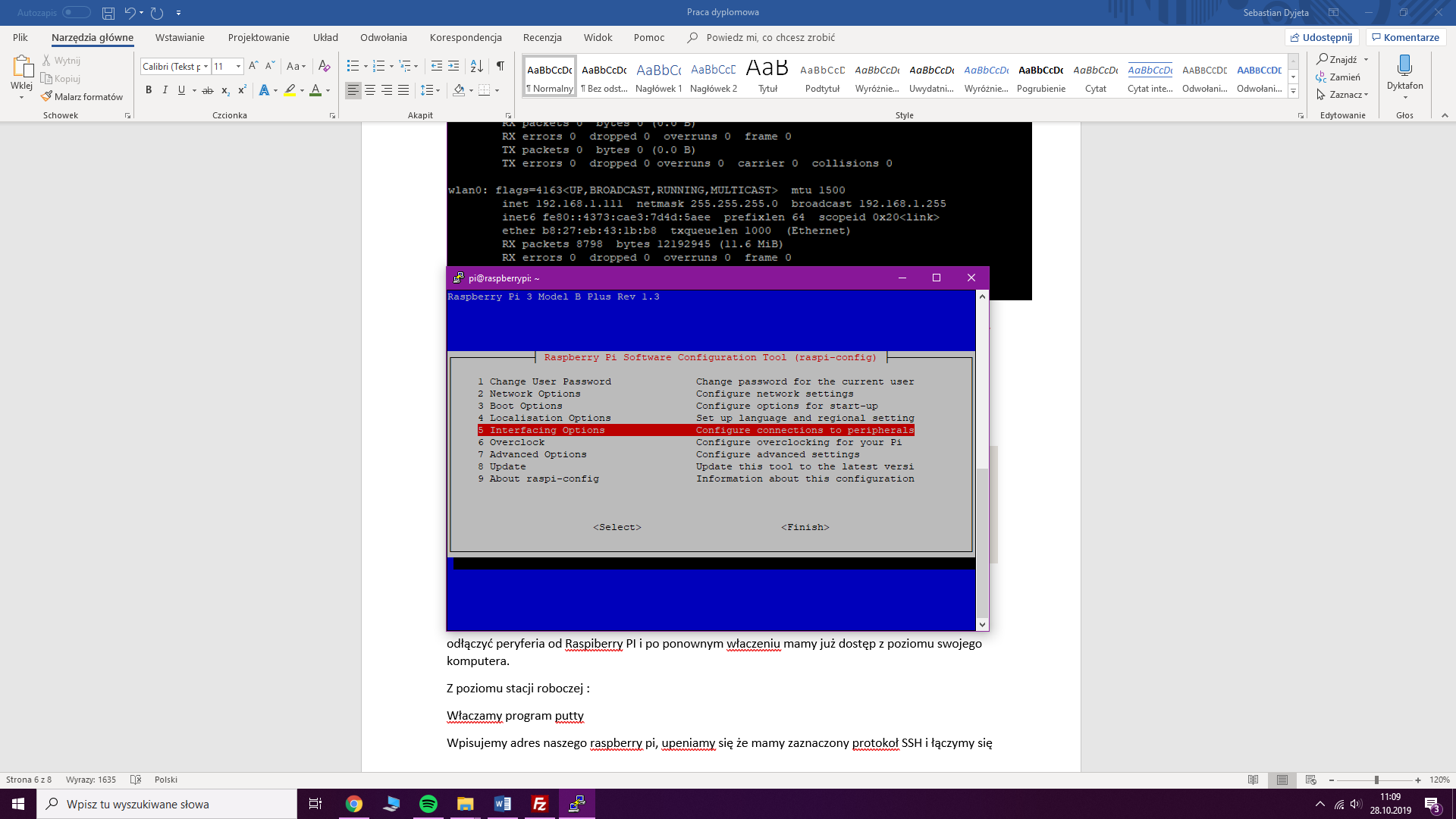
w miejsce ip\_address wpisujemy preferowany statyczny adres IP. Pamiętajcie o tym że adres musi znajdować się w tej samej podsieci co wasza stacja robota. Przypominam że by sprawdzić w jakiej podsieci jesteśmy powinniśmy w konsoli wpisać

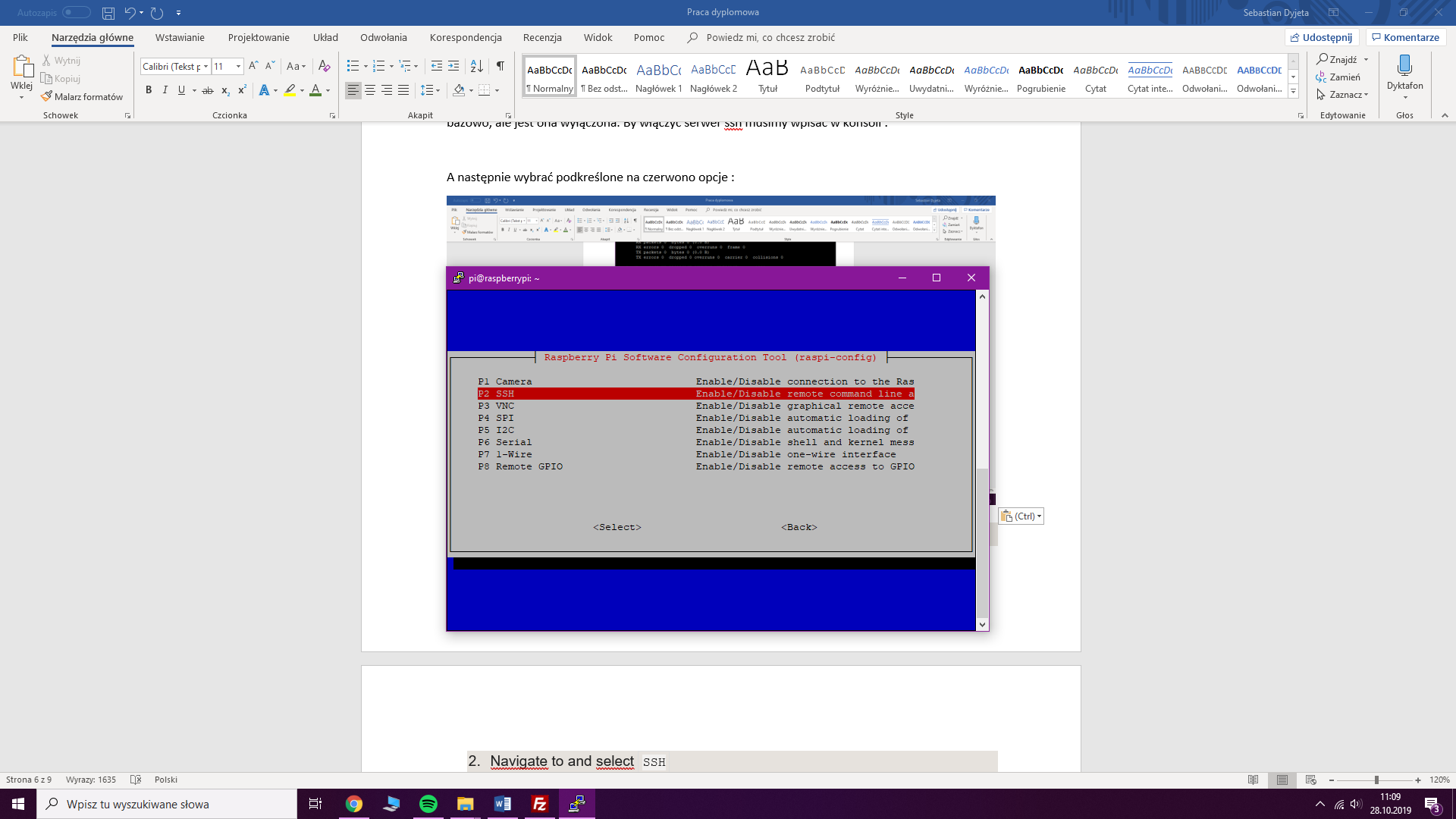
i wyszukać interesujący nasz interface. 

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

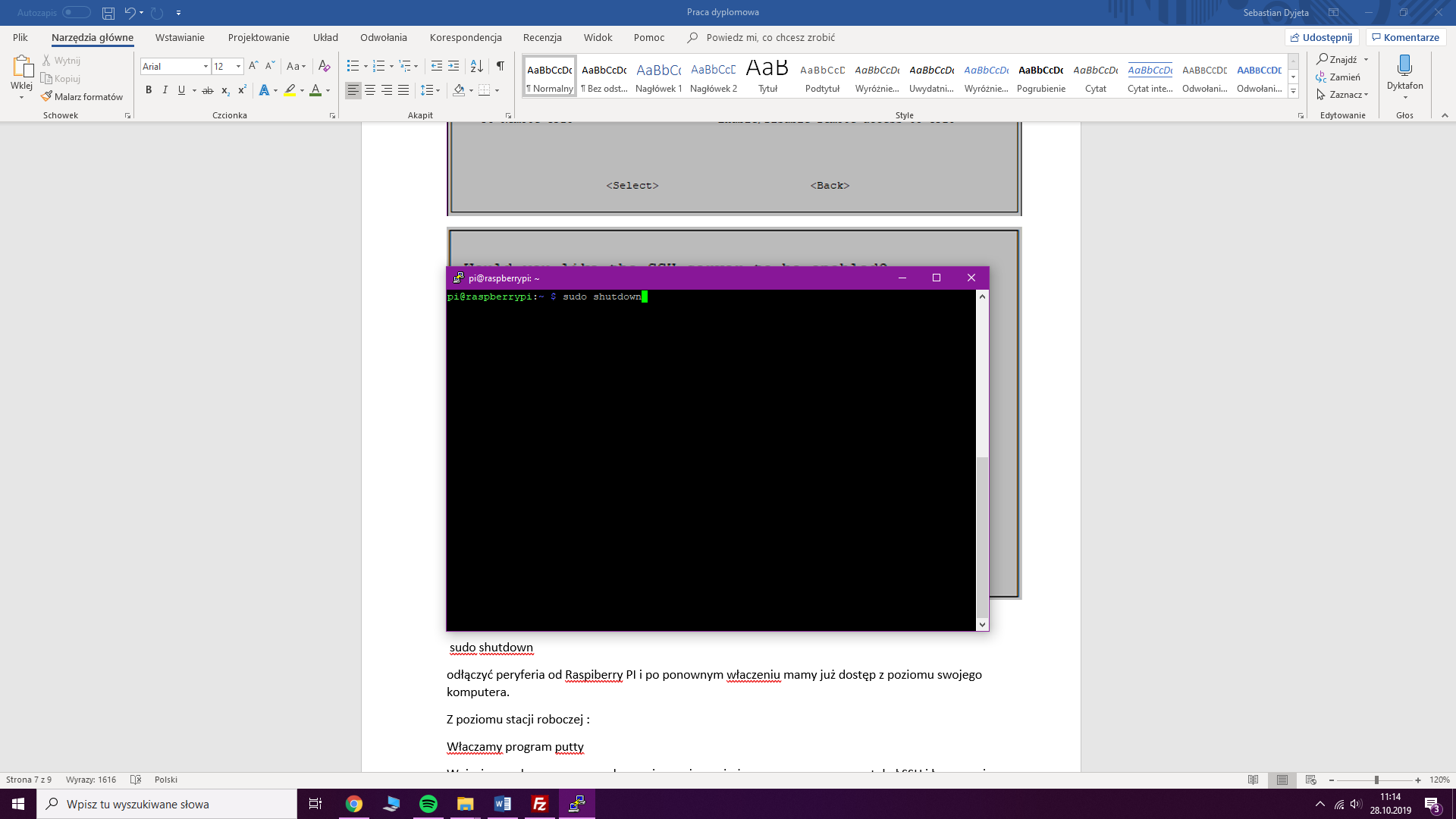


A następnie wybrać podkreślone na czerwono opcje :



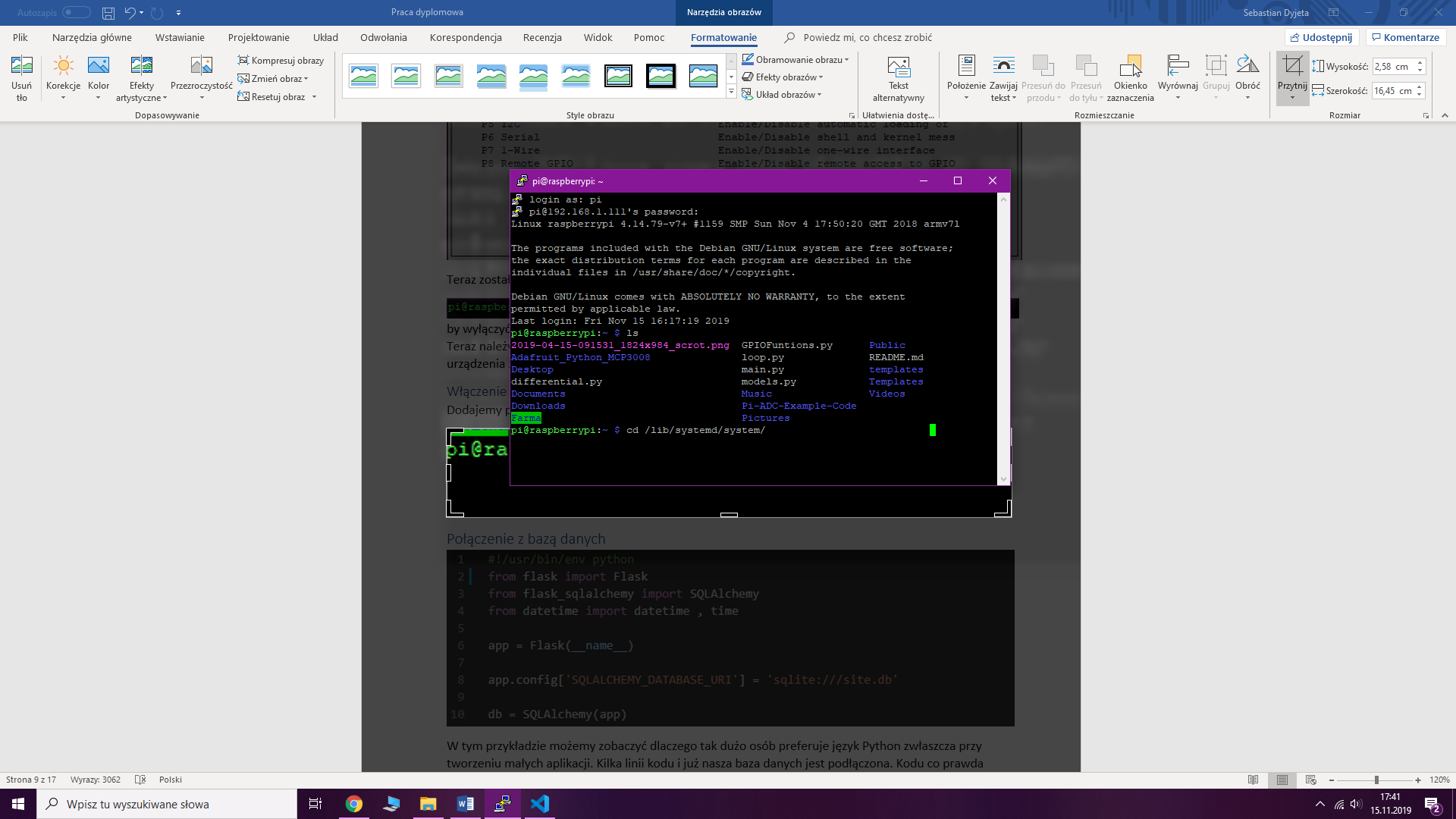


Teraz zostało nam wpisać w konsoli :

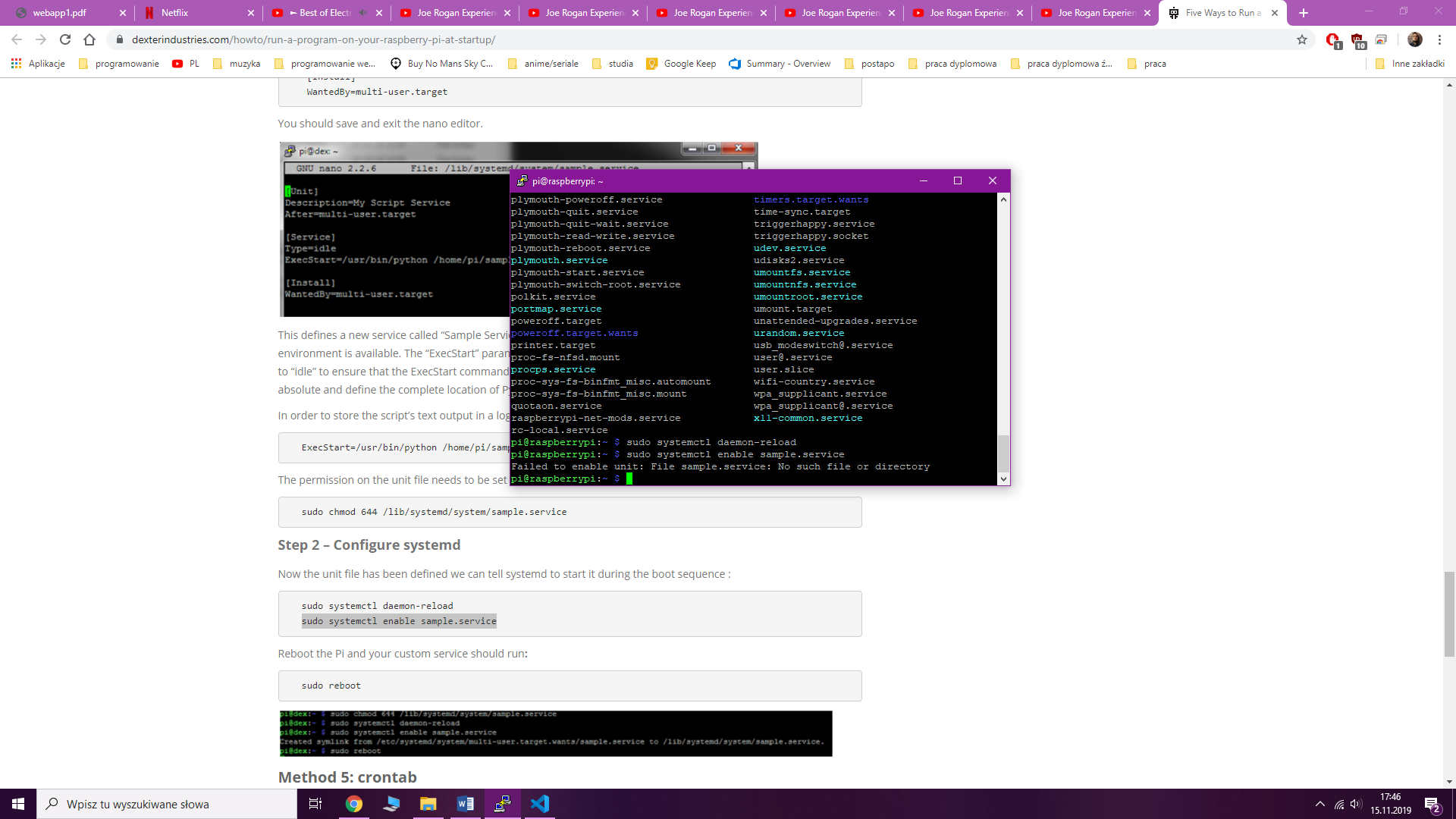
by wyłączyć Raspberry Pi.  
Teraz należy odłączyć peryferia od Raspberry Pi i po ponownym włączeniu mamy już dostęp do urządzenia z poziomu swojego komputera.

### Włączenie aplikacji przy starcie systemu.

Dodajemy plik konfiguracyjny zawierający skrypt uruchamiający naszą aplikację do ścieżki :



Następnie musimy użyć 2 komend które kolejno : załaduje nasz serwis oraz włączy go



Od tej pory nasza aplikacja będzie włączać się przy każdym starcie systemu jako serwis.

## Utworzenie bazy danych

### Schemat bazy danych



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”).

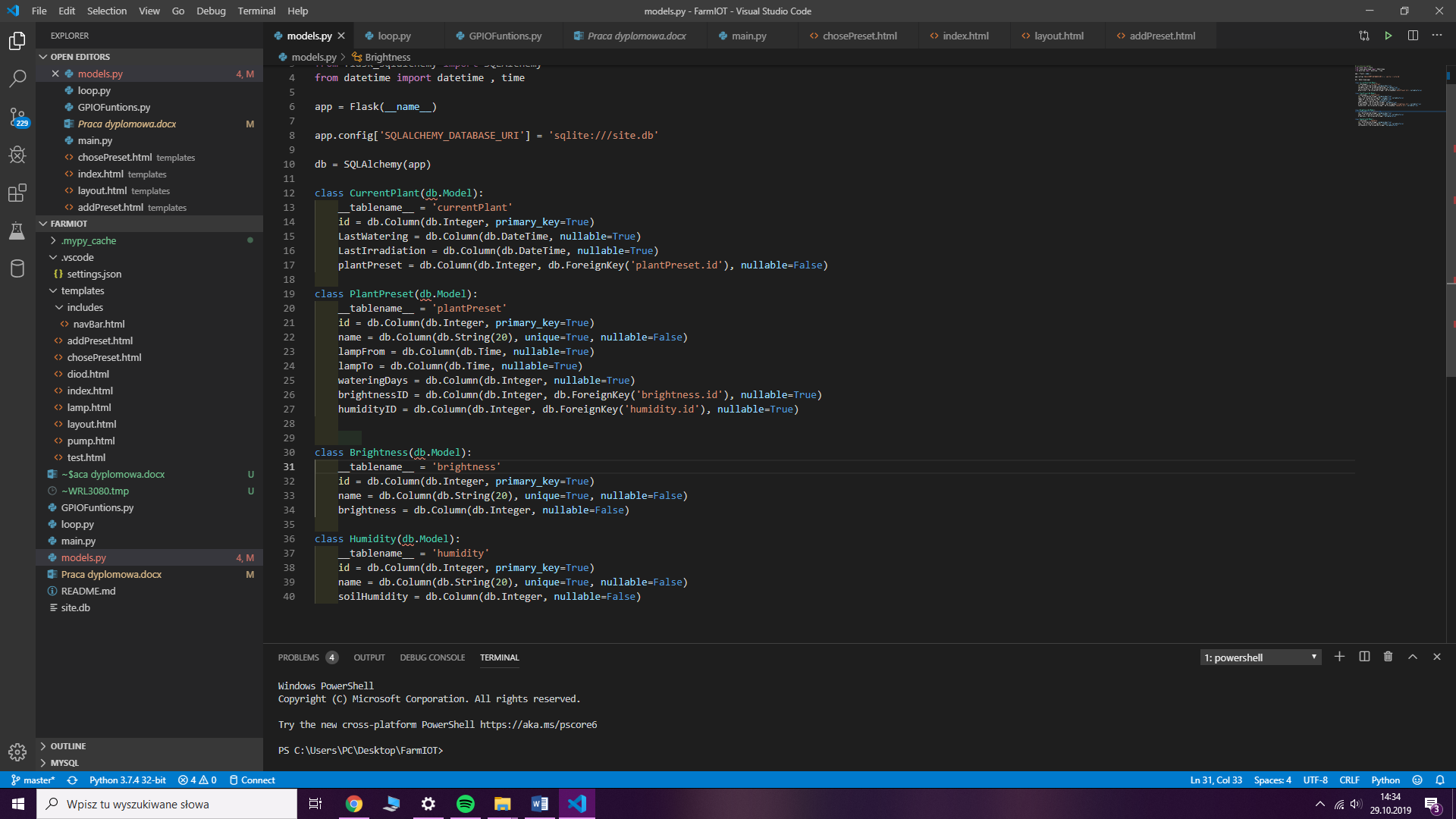
Humidity – tablica zawierająca docelowe wartości wilgotności gleby w formacie zrozumiałym dla systemu oraz przyjazną dla użytkownika nazwę (np. „wysoka wilgotność”).

### Dodanie bazy danych

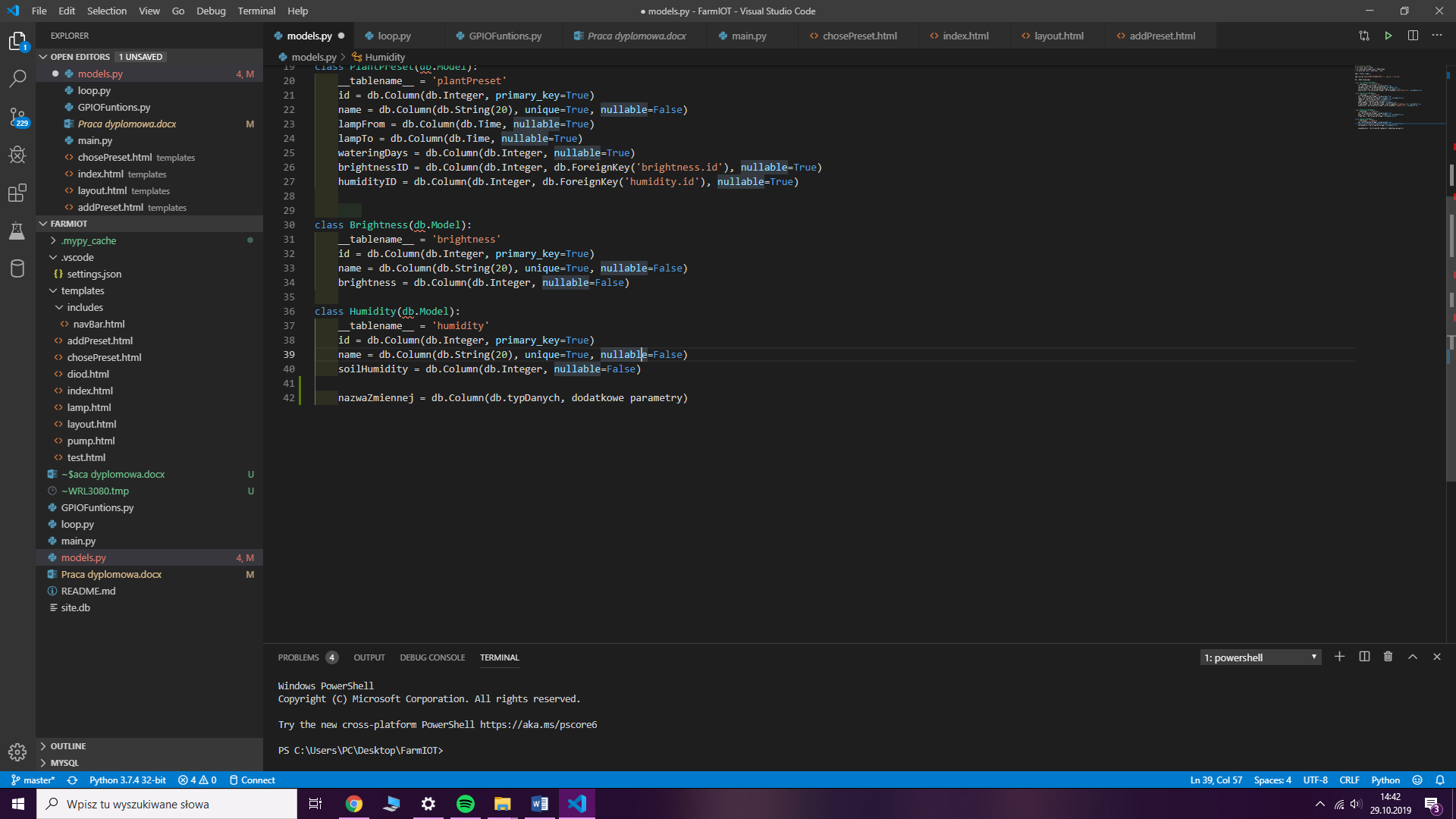
#### Code first aproach

W podejściu tym tworzymy najpierw modele(klasy) obiektów które chcemy mieć w bazie i dopiero później na ich postawie konstruowana jest (często automatycznie) baza danych.

#### Implementacja modeli



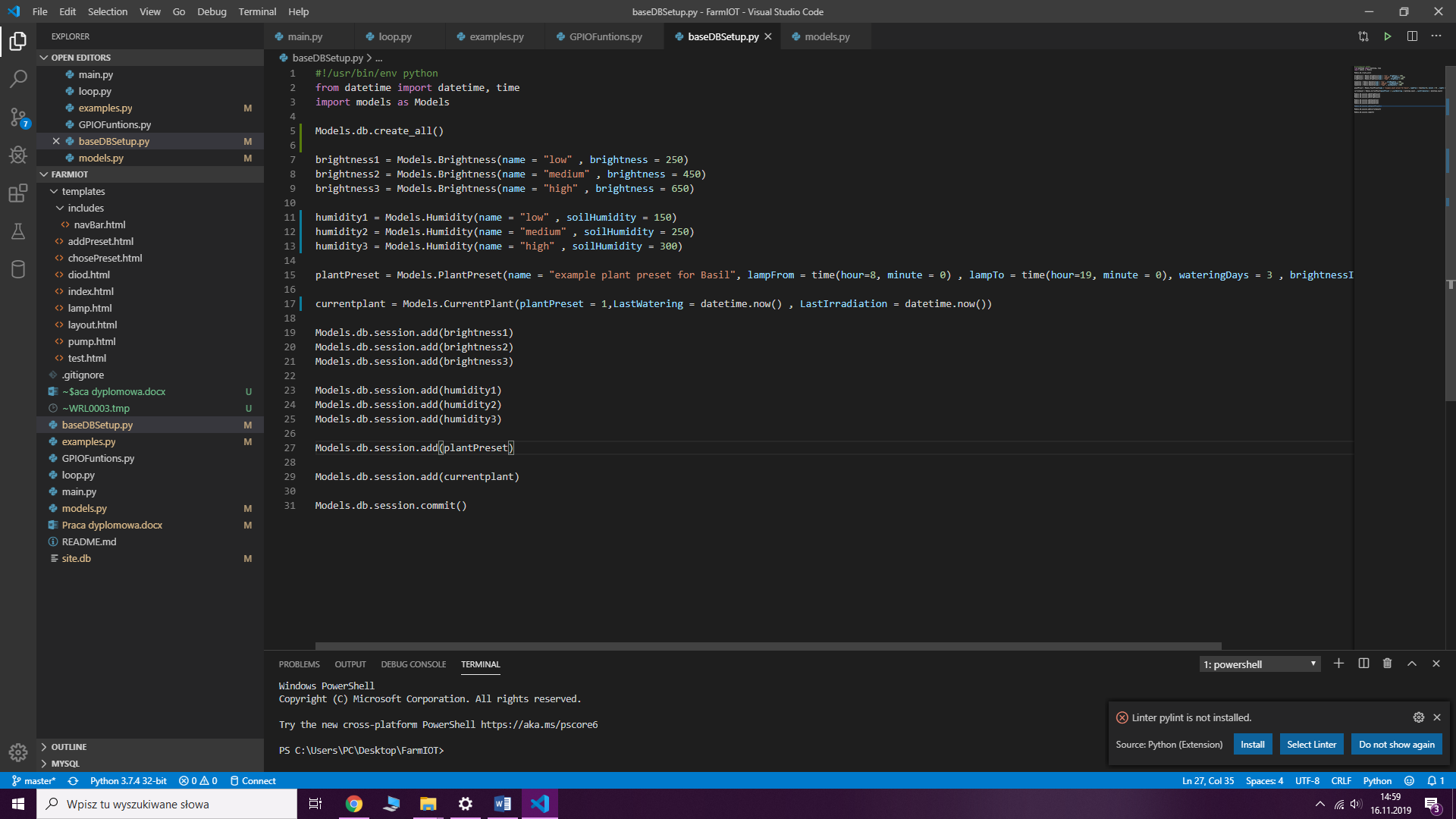
Każdy model widziany przez SQLAlchemy musi dziedziczyć po klasie Model oraz posiadać kolumny opisane w formacie :



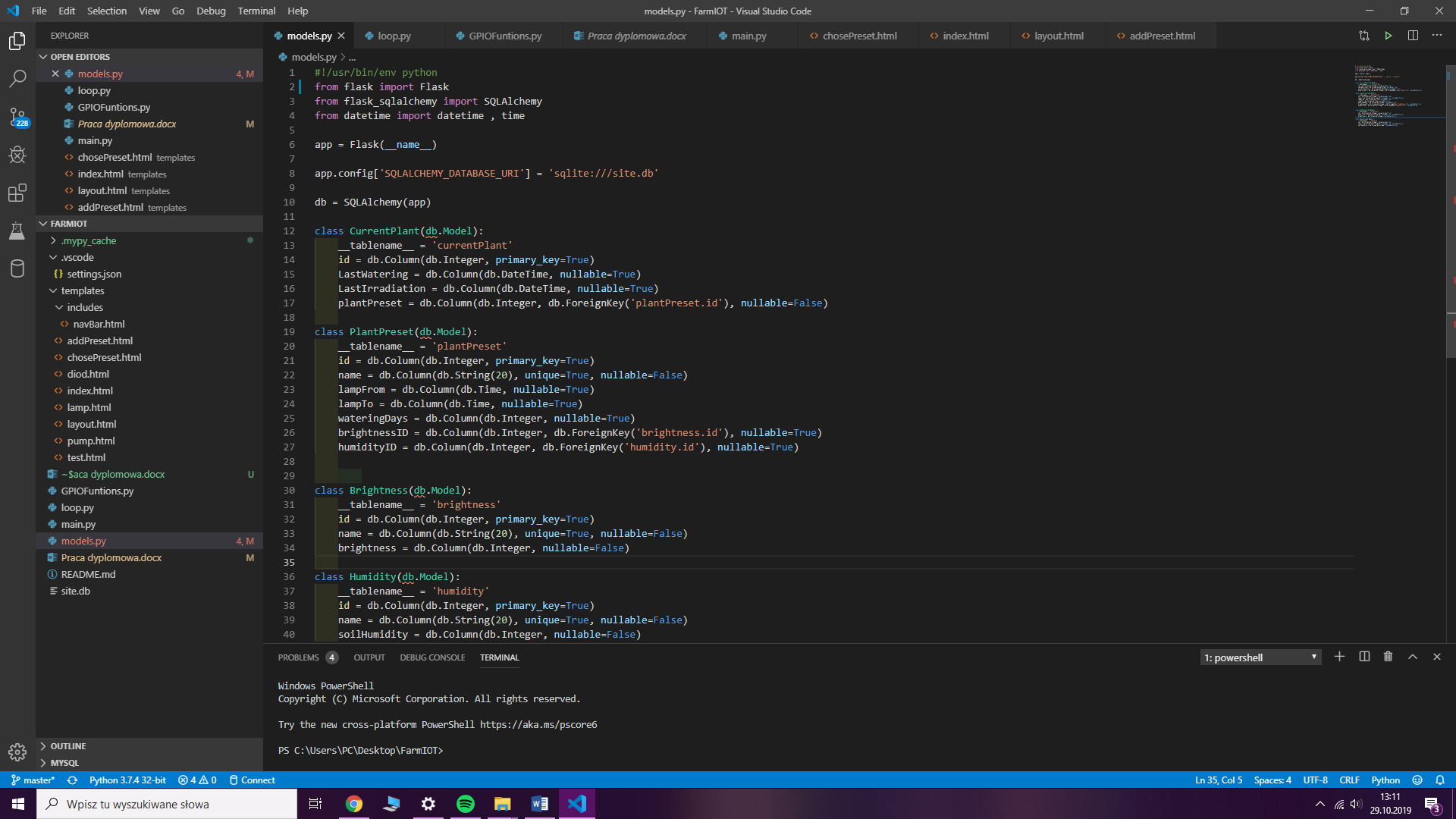
Przez dodatkowe parametry mam namyśli auto inkrementacje, indeksowanie itp.

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.

Teraz musimy utworzyć bazę danych z napisanych modeli, używamy do tego prostej komendy :



## Połączenie z bazą danych



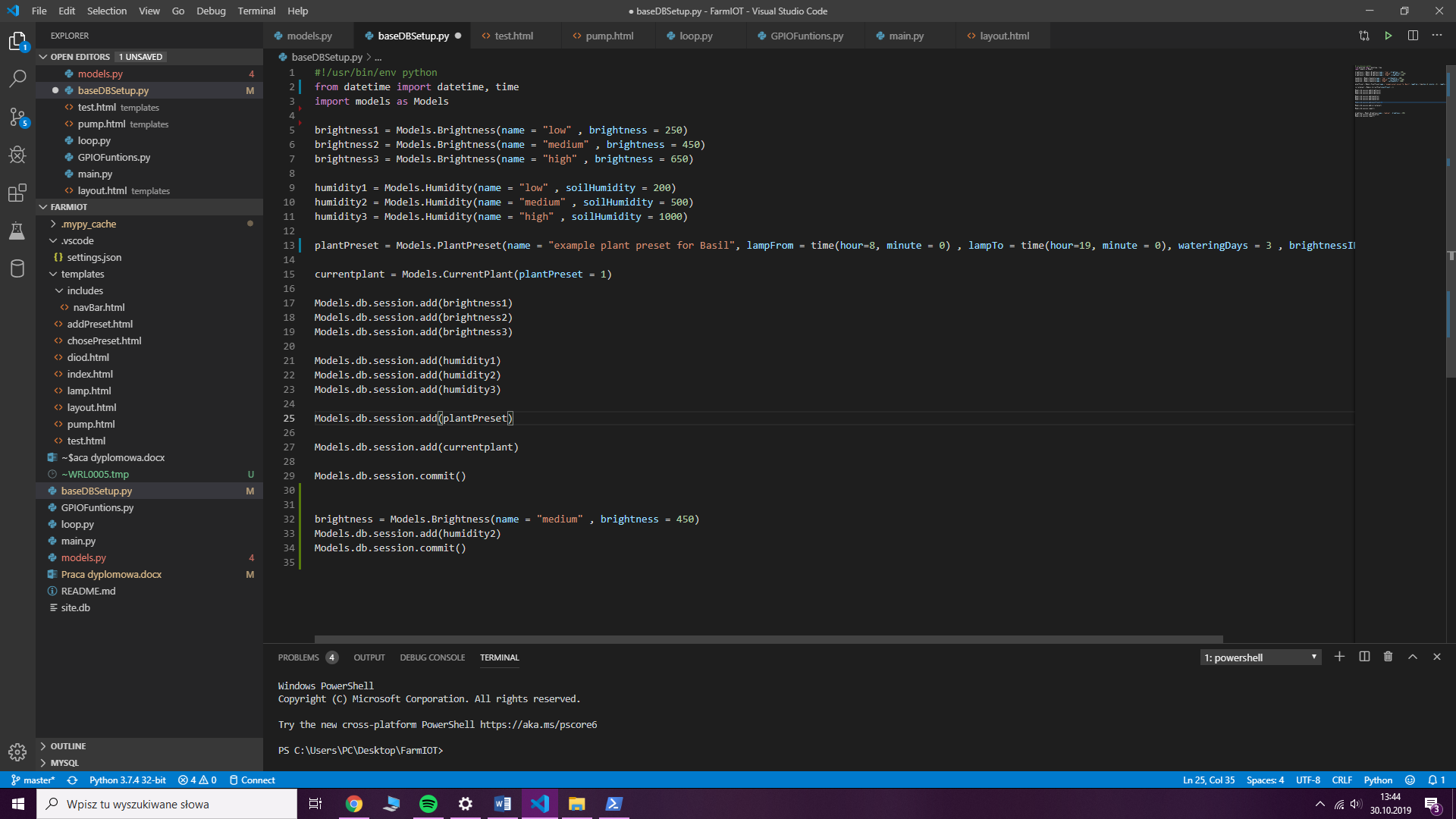
W tym przykładzie możemy zobaczyć dlaczego tak dużo osób preferuje język Python zwłaszcza przy tworzeniu małych aplikacji. Kilka linii kodu i już nasza baza danych jest podłączona. Kodu co prawda jest mało ale integrujemy tutaj aż 3 komponenty czyli :

* SQLite
* SQLAlchemy
* Flask

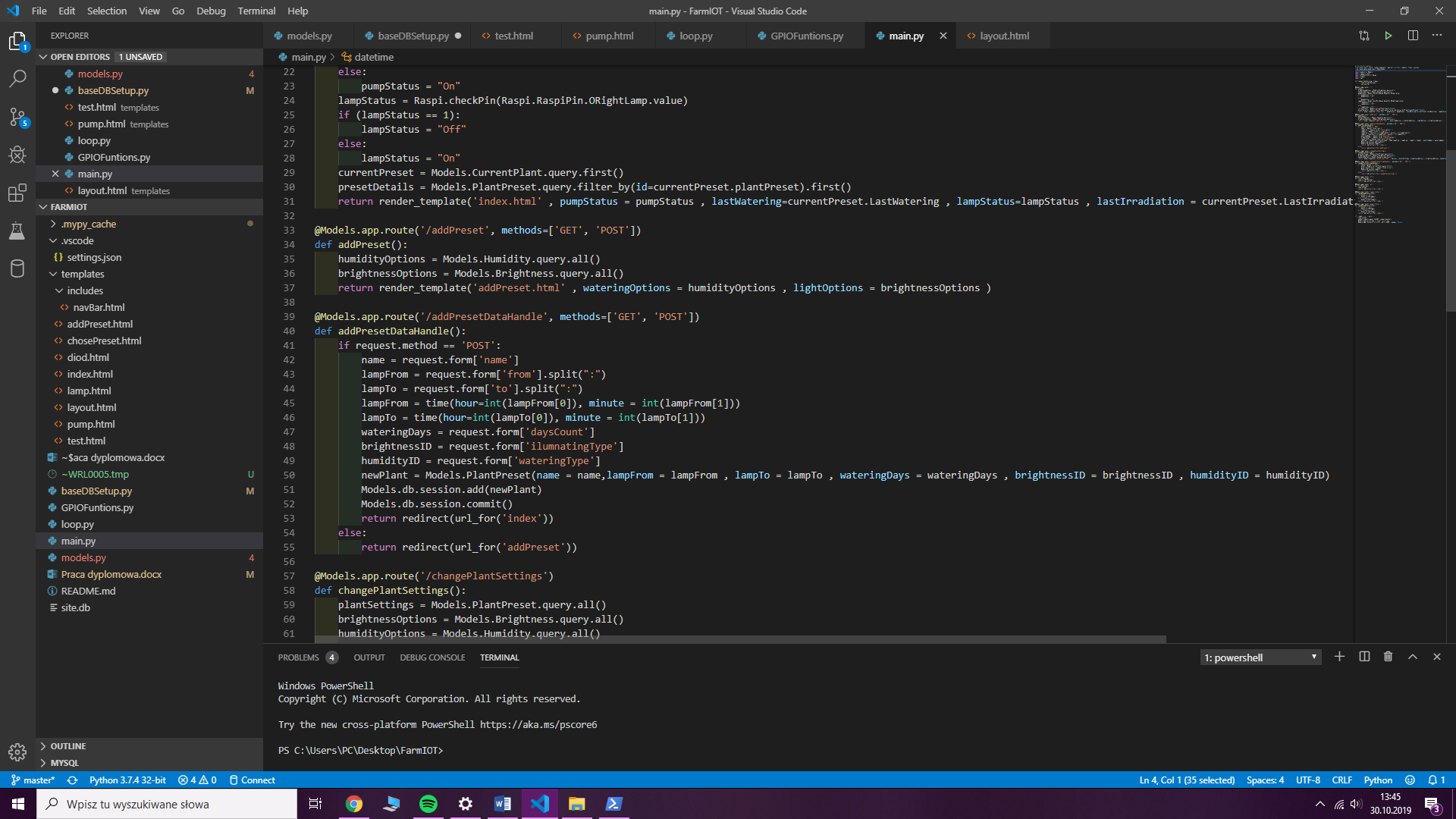
### CRUD (Create,Read,Update,Delete) w SQLAlchemy

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

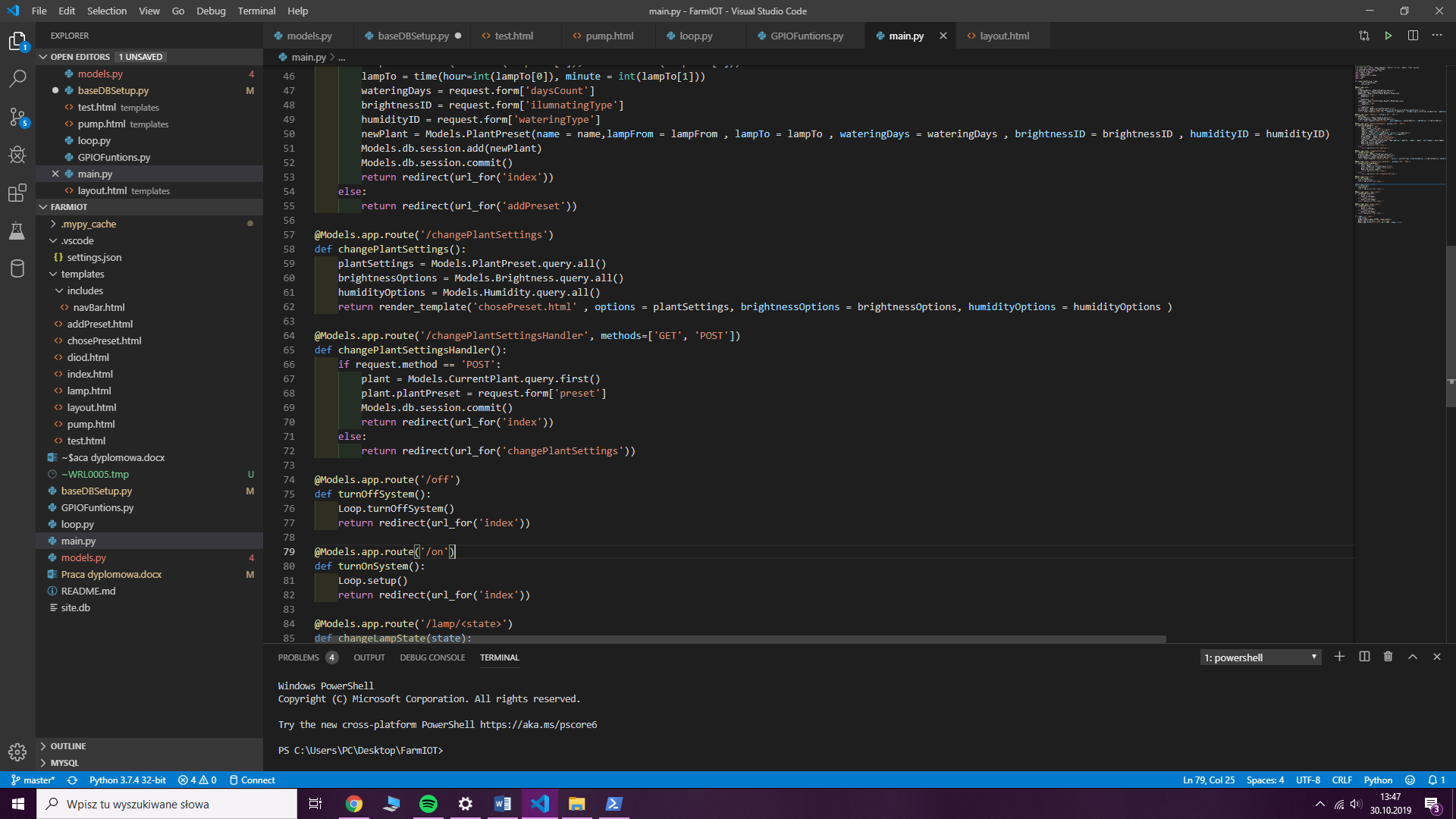
* Dodanie do bazy (Create)



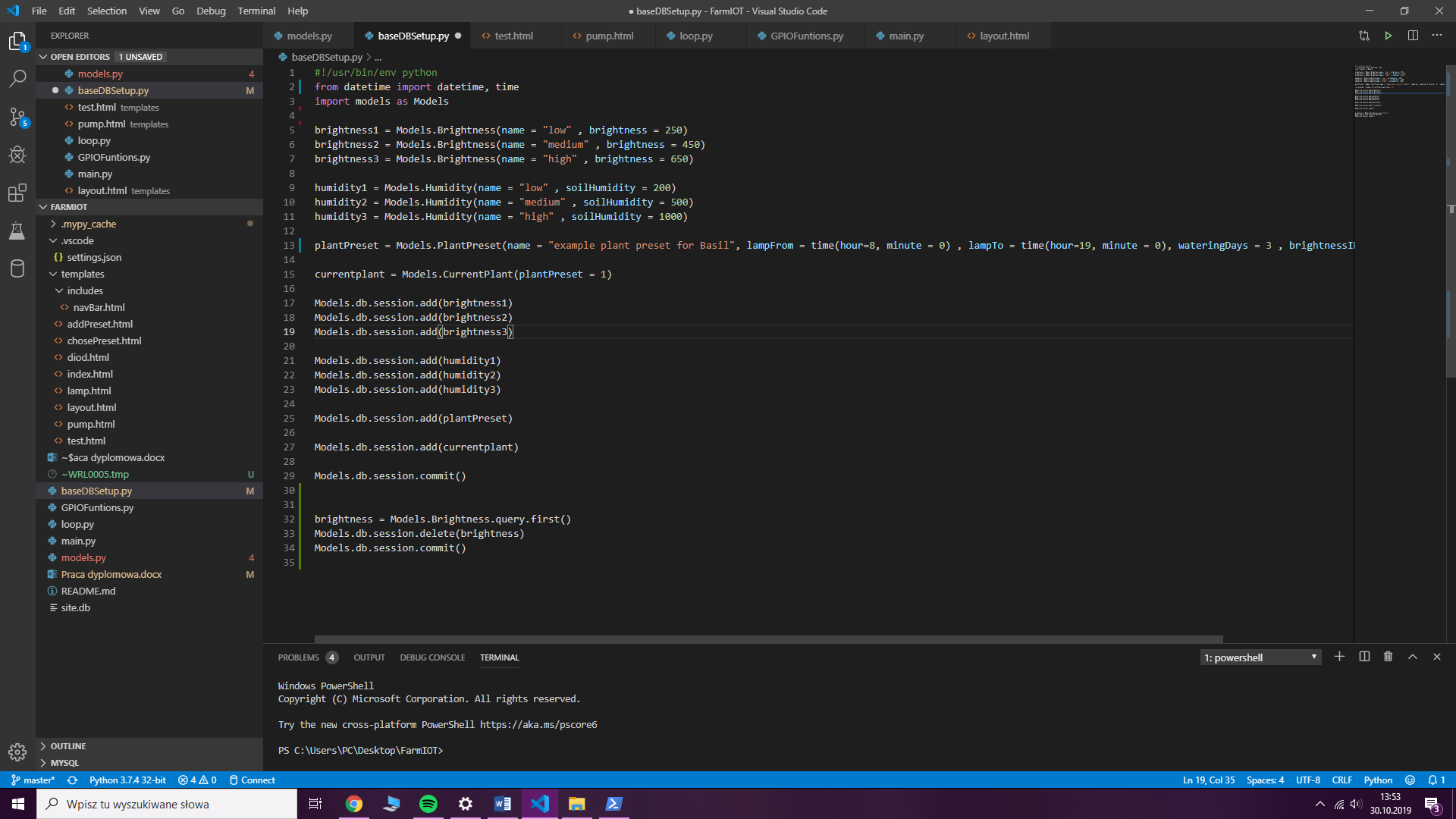
* Odczyt z bazy (Read)



* Edytowanie danych w bazie (Update)

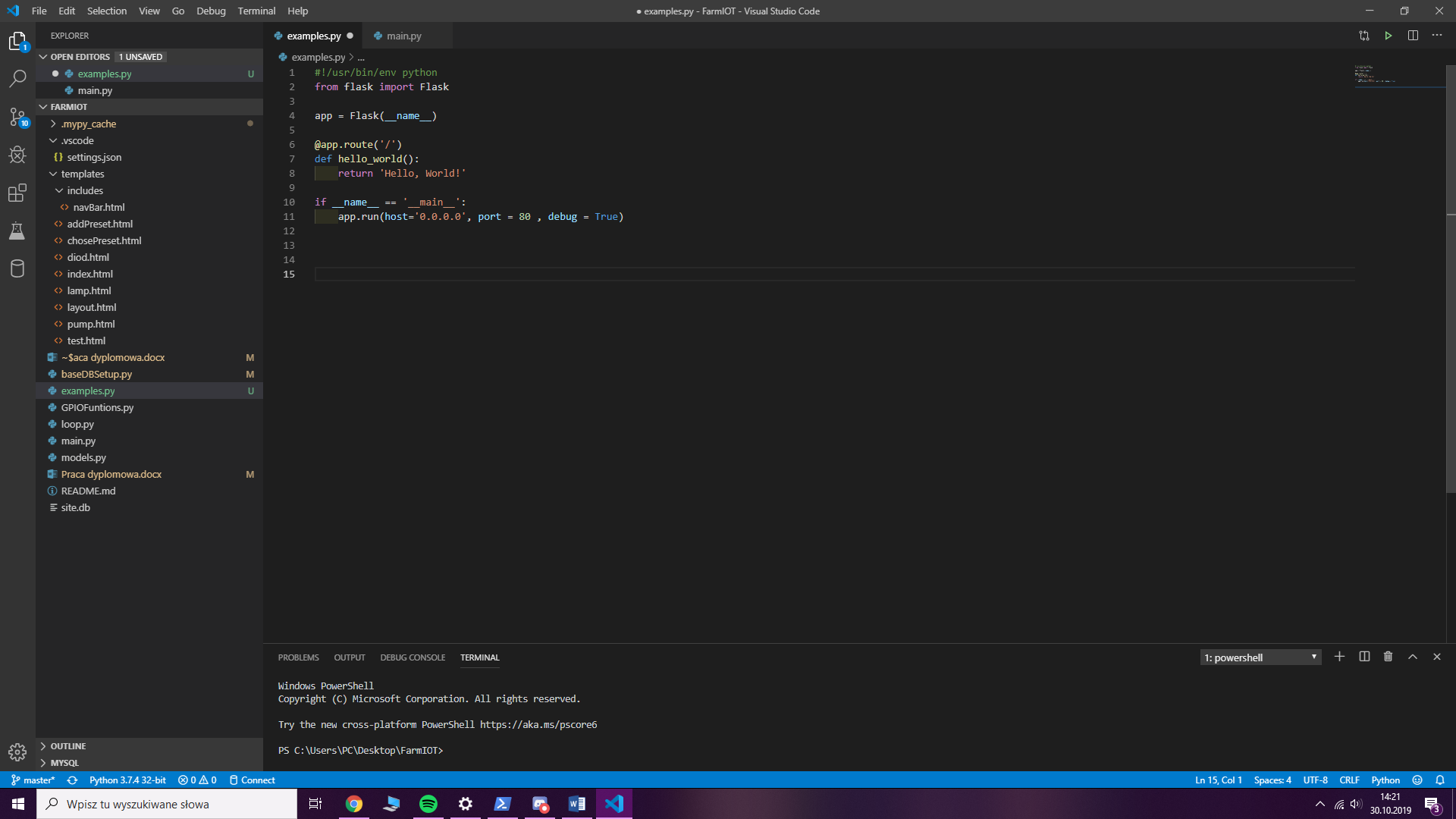


* Usuwanie z bazy (Delete)



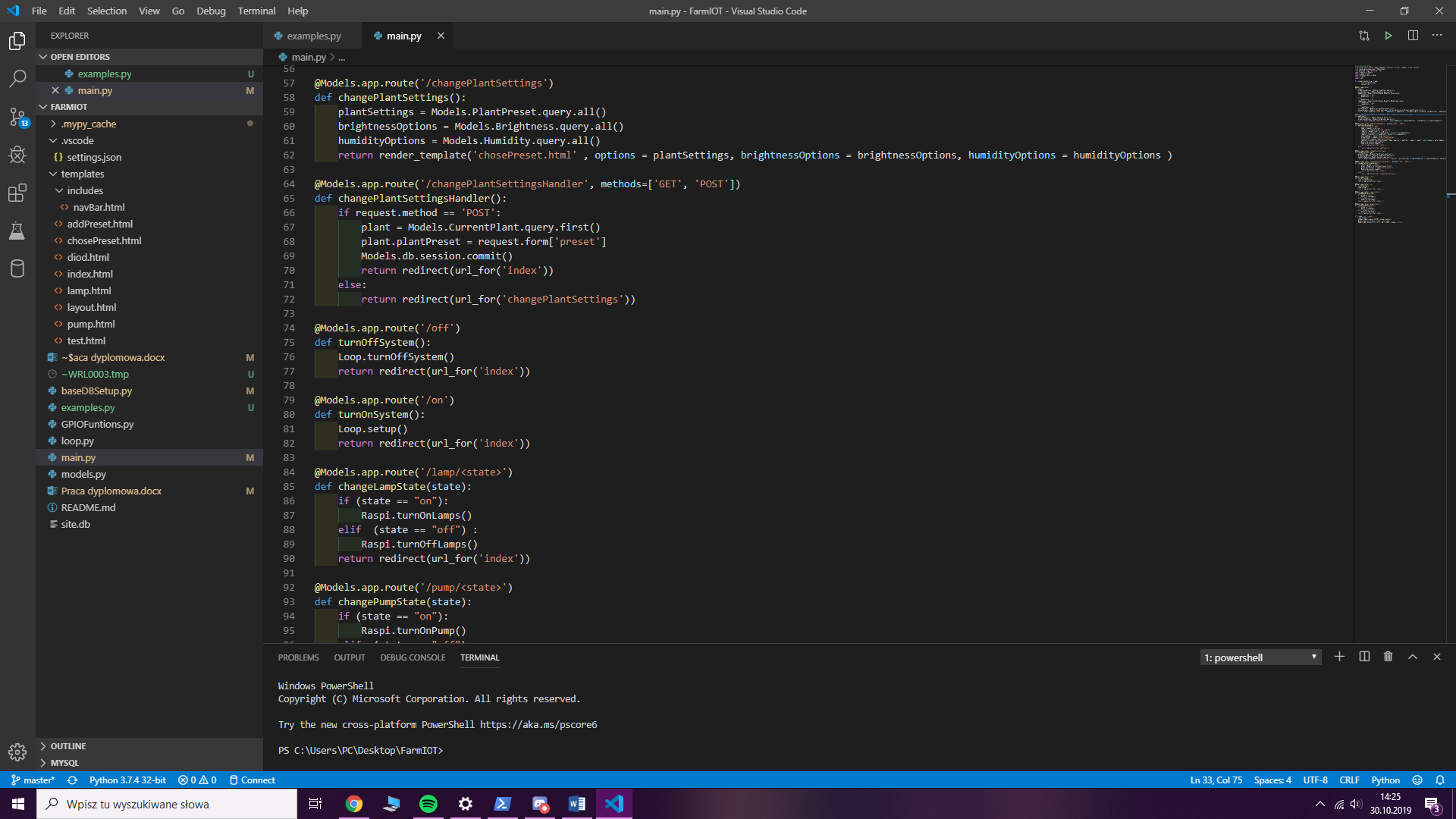
## API

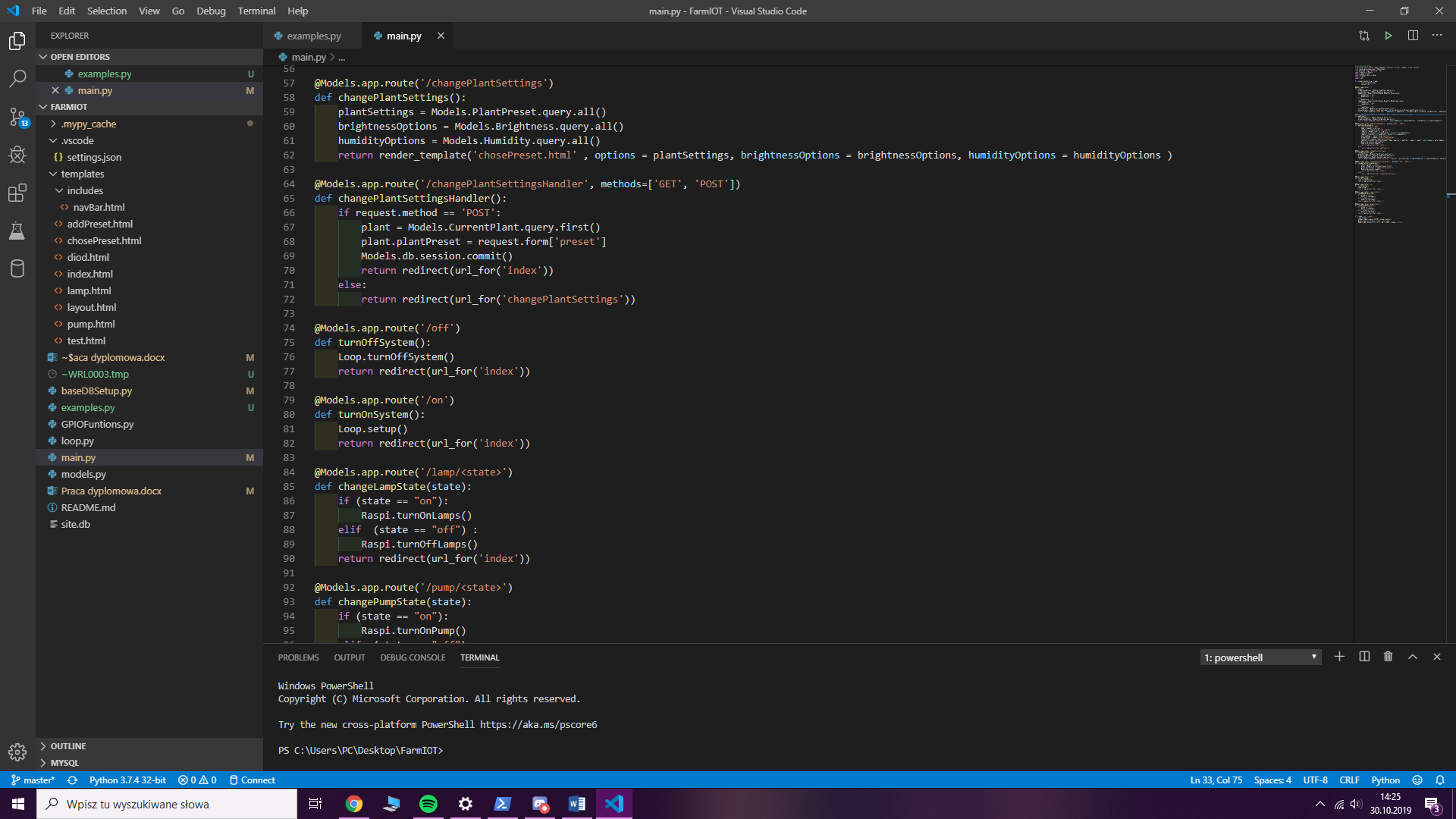
### Podstawy Web API



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. Należy pamiętać by finalnie opcja ta była ustawiona na False, ponieważ tryb ten pozwala na poznanie całej struktury aplikacji oraz wywołanie dowalonych funkcji.



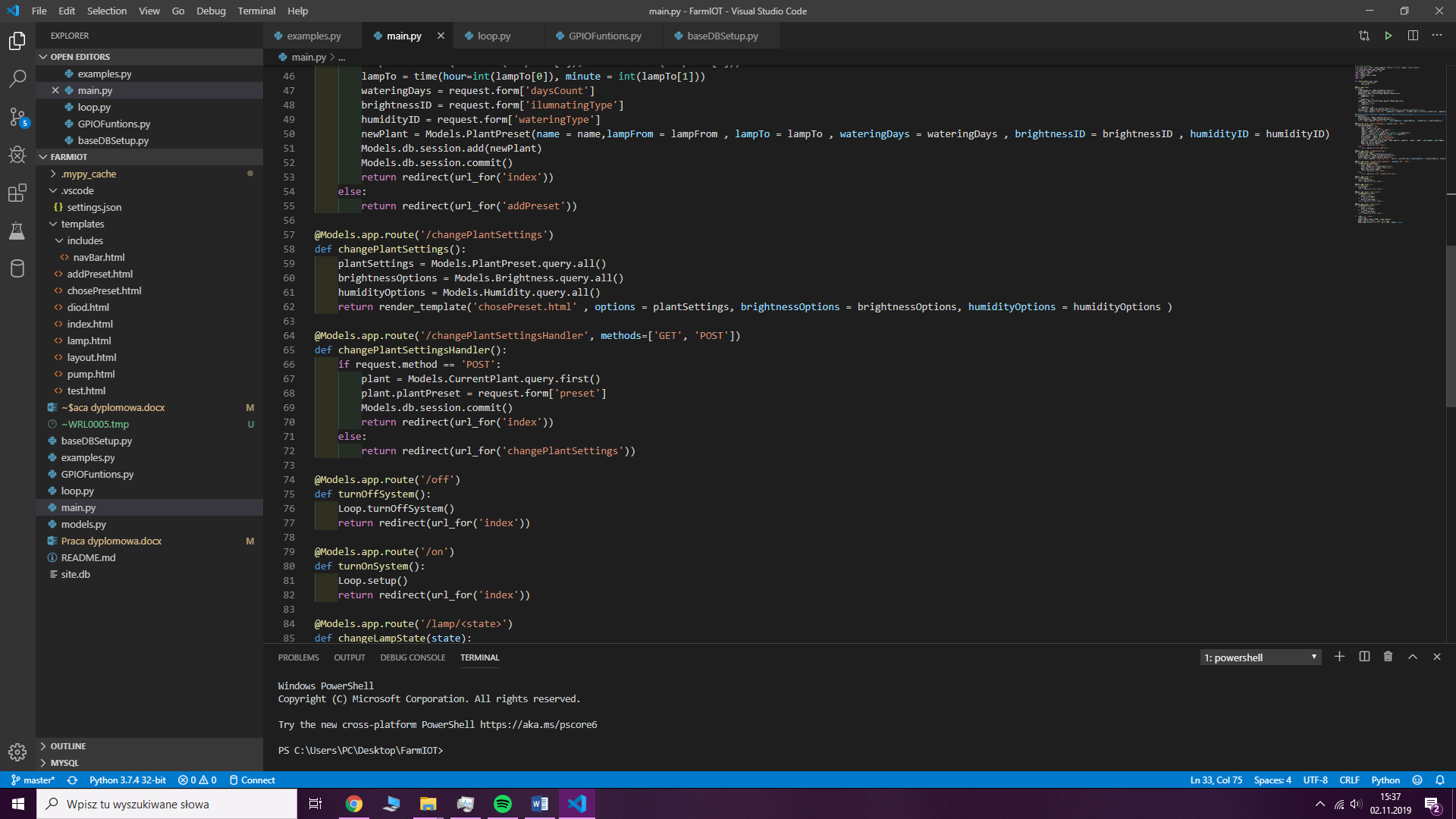
W Flasku jak w przypadku każdego frameworka do tworzenia API o architekturze RESTowej tworzymy tak zwane endpointy do których wysyłane są żądania HTTP, endpoint składa się z adresu domeny, parametrów oraz metody. Jeden URL może mieć do siebie przypisane kilka endpointów w zależności od użytej metody, 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 endpoint a następna rzecz po „/” będzie odczytana jako wartość zmiennej. 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 sukcesu) oraz odpowiedz która w architekturze REST jest zazwyczaj plikiem JSON, choć w przypadku opisywanego systemu w większości przypadków będzie to strona internetowa.

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), lub w przypadku przekierowania na inną stronę dostaniemy kod 302.

Przekierowanie do innego endpointa wygląda w następujący sposób :



### Generowanie strony

Gdy nasz endpoint kończy się w ten sposób



Zwracamy użytkownikowi status 200 wraz z wygenerowaną stroną w ciele odpowiedzi. Przebieg generowania strony jest następujący :

Endpoint odnosi się do konkretnego pliku html (w tym przypadku „index.html”) który powinien zaczynać się od

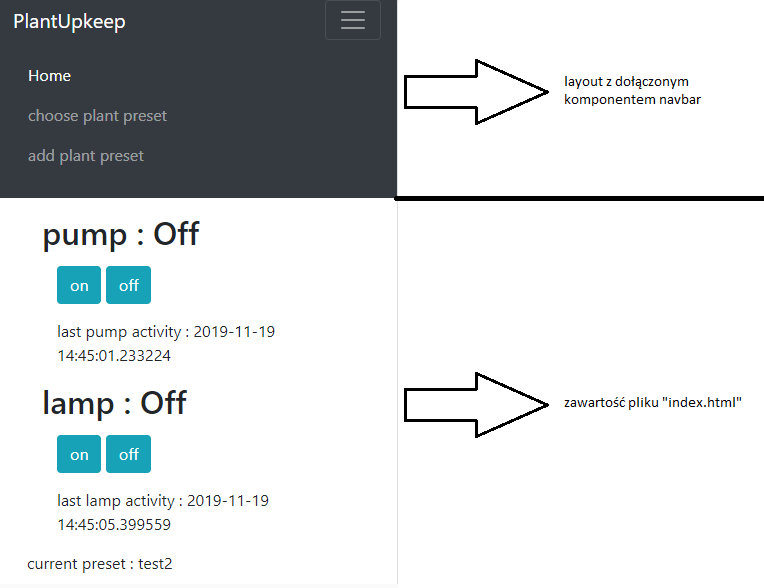
{% extends 'layout.html' %}

Który dołącza do strony część wspólną dla wszystkich podstron, następnie piszemy standardowy kod HTML. Możemy też doładować w dowolnym miejscu komponent

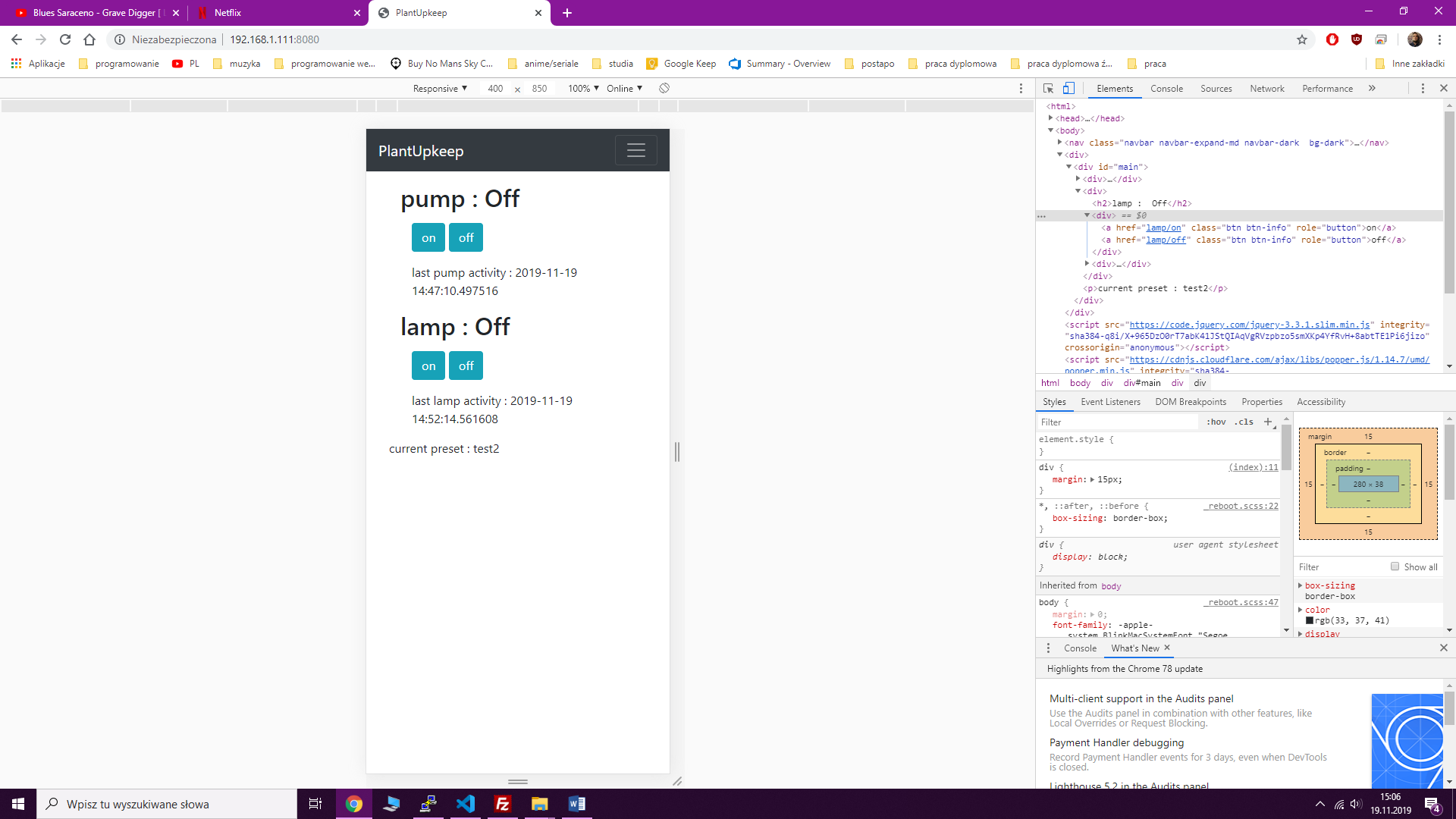
{% include 'includes/navBar.html' %}

Co załaduje nam cały HTML z danego pliku. W tym przypadku layout zawierający metadane doładowuje navBar.html ponieważ są to dwie rzeczy które będą znajdować się na każdej stronie.

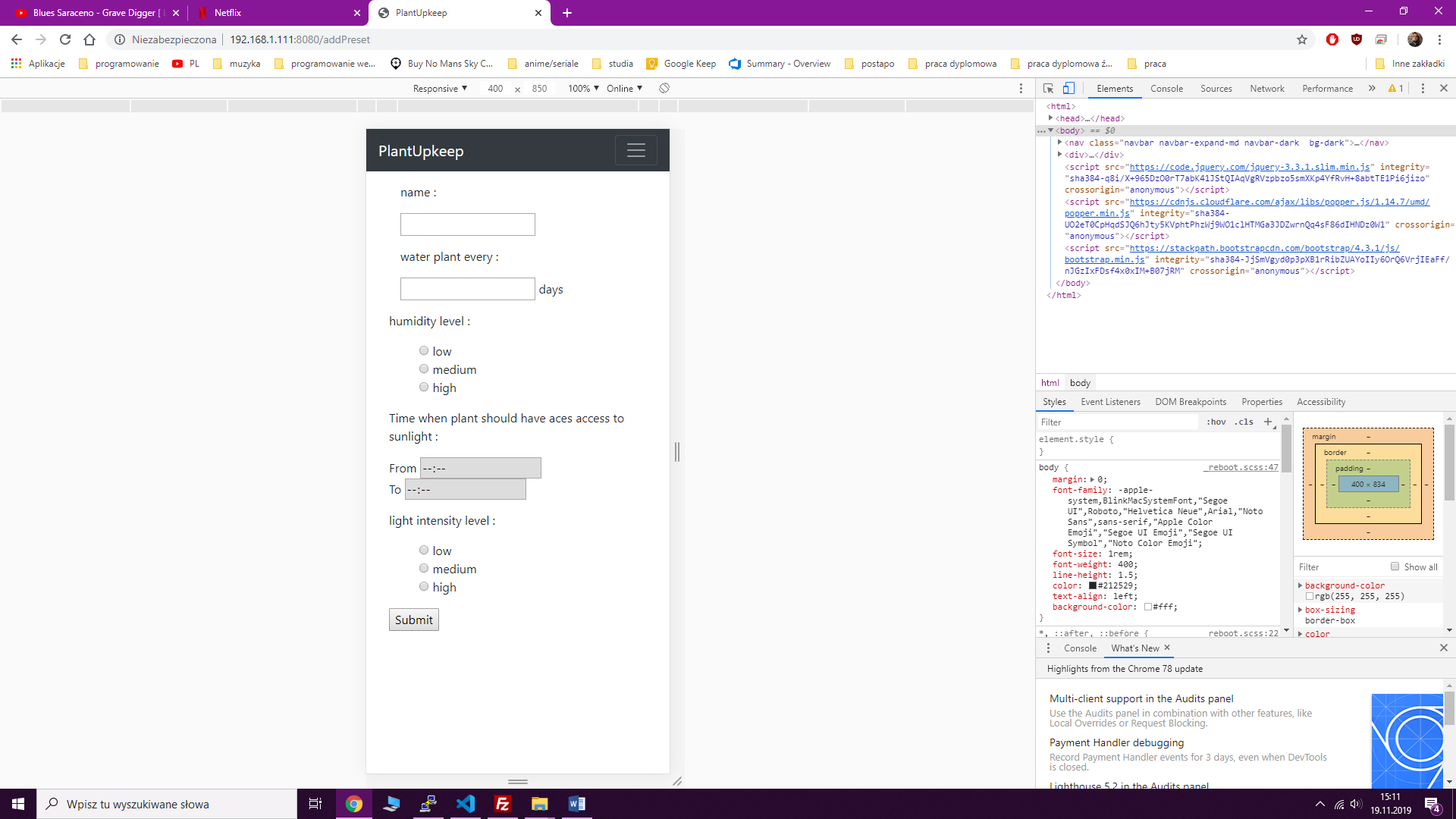
Wynik :



## Widoki

System zawiera 3 widoki

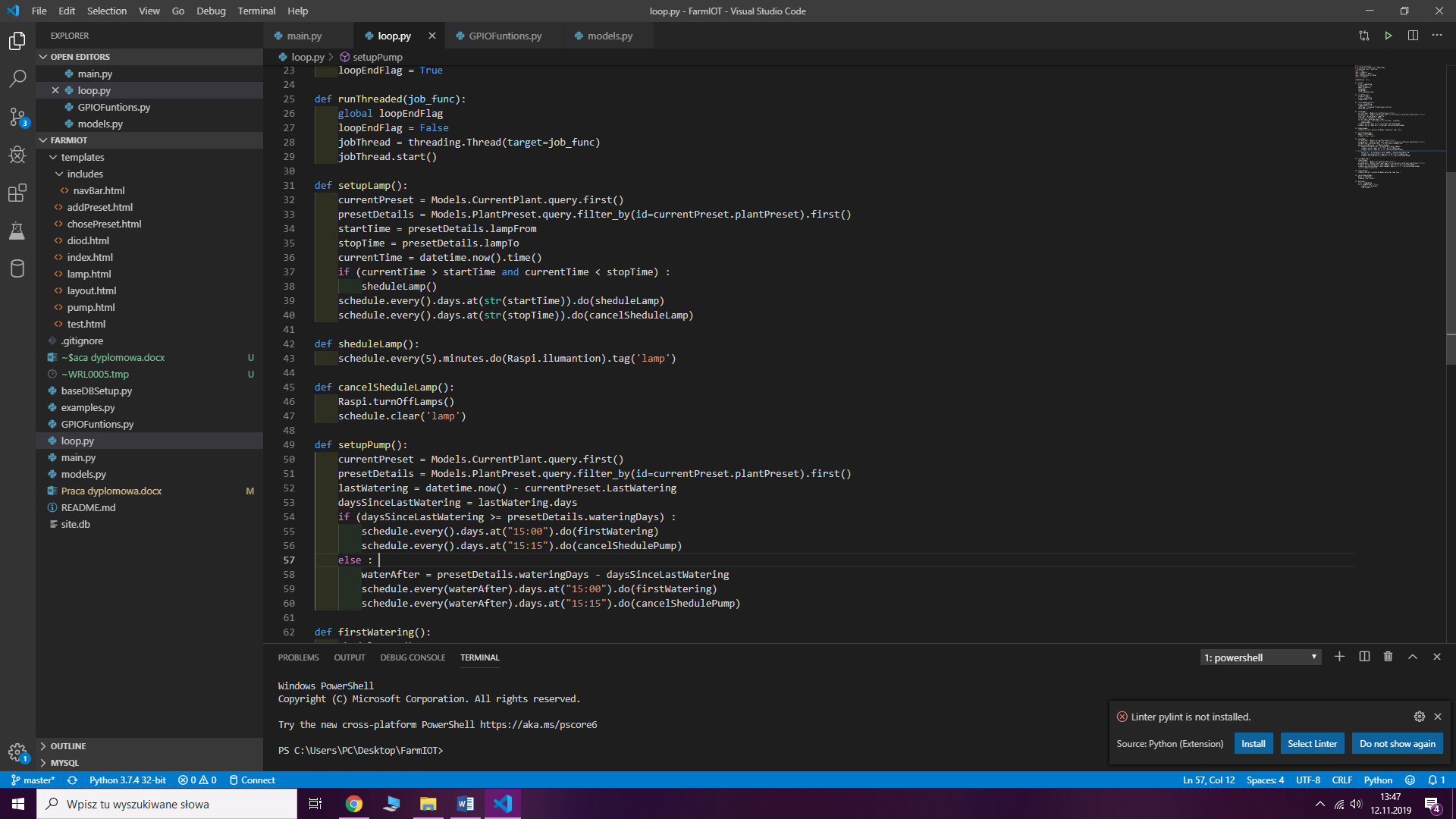
Home – widok w którym możemy zobaczyć aktualny stan aktorów oraz manualnie włączyć lub wyłączyć ich. Ponadto możemy zobaczyć czas ostatniego ich użycia oraz aktualnie włączone ustawienie.

Add new preset – widok w którym możemy dodać nowe ustawienie dla rośliny.

### Choose preset – widok który wyświetla nam wszystkie wcześniej utworzone ustawienia i pozwala na wybranie aktualnego. Algorytmy użycia aktorów

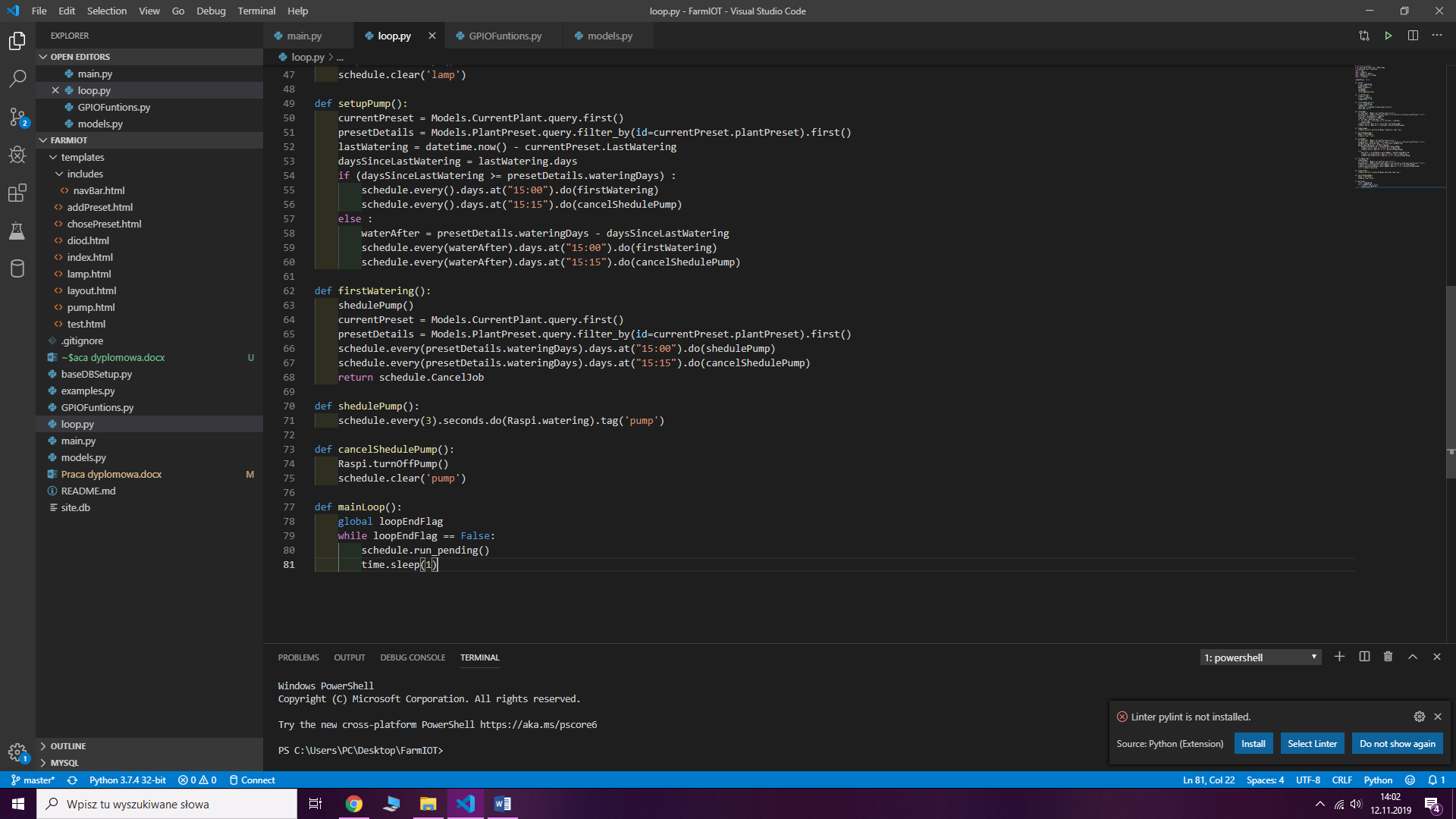
Samo korzystanie z aktorów jest dość proste: pobieramy z bazy danych docelową wartość parametru (nawodnienie lub nasłonecznienie) i jeśli sensor zwróci wartość poniżej wartości docelowej przypisany aktor jest włączany. Całe skomplikowanie programu nadzorującego polega na wyznaczeniu kiedy wspomniana wyżej operacja ma się wykonywać. Wygląda to następująco :

### Naświetlanie



Mamy tutaj funkcje ustawiającą (setupLamp) oraz dwa zdarzenia (sheduleLamp i cancelSheduleLamp). Funkcja ustawiająca ma za zadanie zapisać dwa zdarzenia by wykonały się o podanych w bazie danych godzinach. Pierwsze zdarzenie co 5 minut wykonuje pomiar naświetlenia i dostosowuje stan lamp, a zadanie drugie wyłącza zadanie pierwsze by roślina w nocy mogła odpocząć od światła.

### Nawadnianie



W tym przypadku też mamy funkcje ustawiającą (setupPump) oraz trzy zdarzenia (firstWatering, shedulePump oraz cancelShedulePump). Tutaj funkcja ustawiająca ma za zadanie ustawienie tylko pierwszego zdarzenia (firstWatering) ponieważ nawodnienie chcemy ustawić na x dni od poprzedniego nawodnienia a nie na x dni od aktualnej daty. Dopiero zdarzenie pierwszego nawodnienia ustawia nam regularne podlewanie rośliny co x dni (w zależności od ustawień rośliny). Sprawdzanie czujnika wilgoci odbywa się w wyznaczone dni co 3 sekundy od 15:00 do 15:15 tego samego dnia. Korekta działania pompy odbywa się o wiele częściej niż działania lamp ponieważ nadmiar wilgoci ma dużo większe znaczenie dla rośliny niż za długi czas naświetlania.

## Wykonywanie funkcji w zaplonowanym czasie

Do wykonywania zadań w danym okresie czasu wykorzystałem bibliotekę schedule stworzoną przez Daniela Badera. Biblioteka ta używa wzorca budowniczego do konfiguracji, co pozwala zapisywać nam do planera wszelkie funkcje, bądź inne wywoływalne obiekty (callable) by wykonywały się w wybranych przez nas okresach czasu.

### Wzorzec „Budowniczy” (builder pattern)

Jest to wzorzec projektowy z kategorii wzorców kreacyjnych który ma na celu budowanie złożonego obiektu krok po kroku z mniejszych obiektów co daje nam większą kontrole i przejrzystość obiektu.

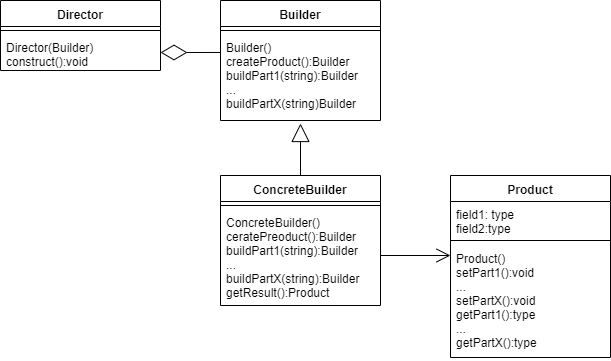
**Zalety :**

* Daje nam wyraźne rozróżnienie i dodatkową warstwę między konstrukcją i reperacją konkretnego obiektu stworzonego przez klasę.
* Pozwala na ponowne użycie kodu przy budowie różnych reprezentacji obiektu.
* Zasada pojedynczej odpowiedzialności (Single Reposibility Principle) : pozwala na izolacje złożonego kodu konstrukcji od logiki biznesowej obiektu.

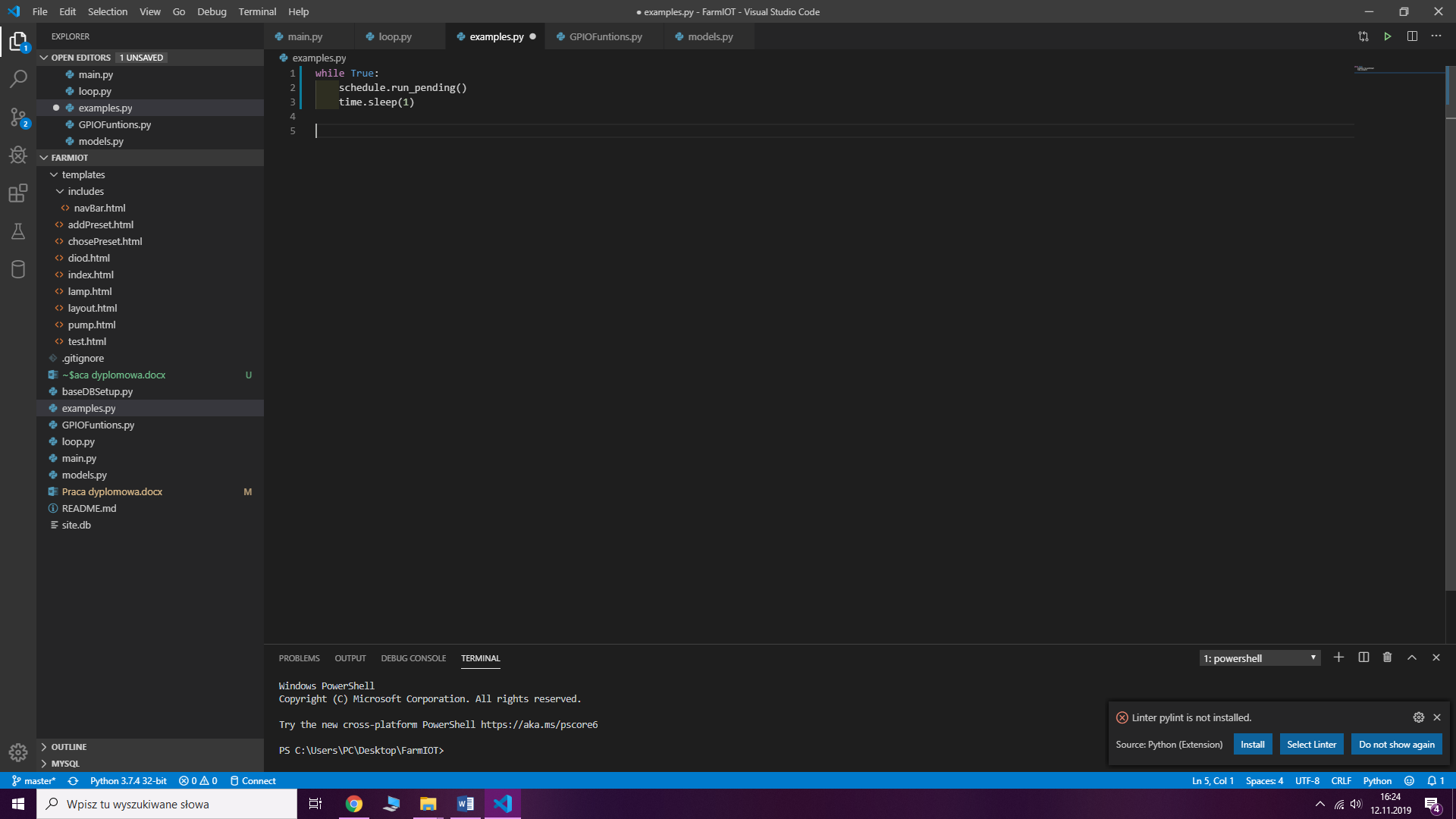
**Wady:**

* ogólna złożoność kodu jest większa, ponieważ wzorzec wymaga tworzenia wielu klas.

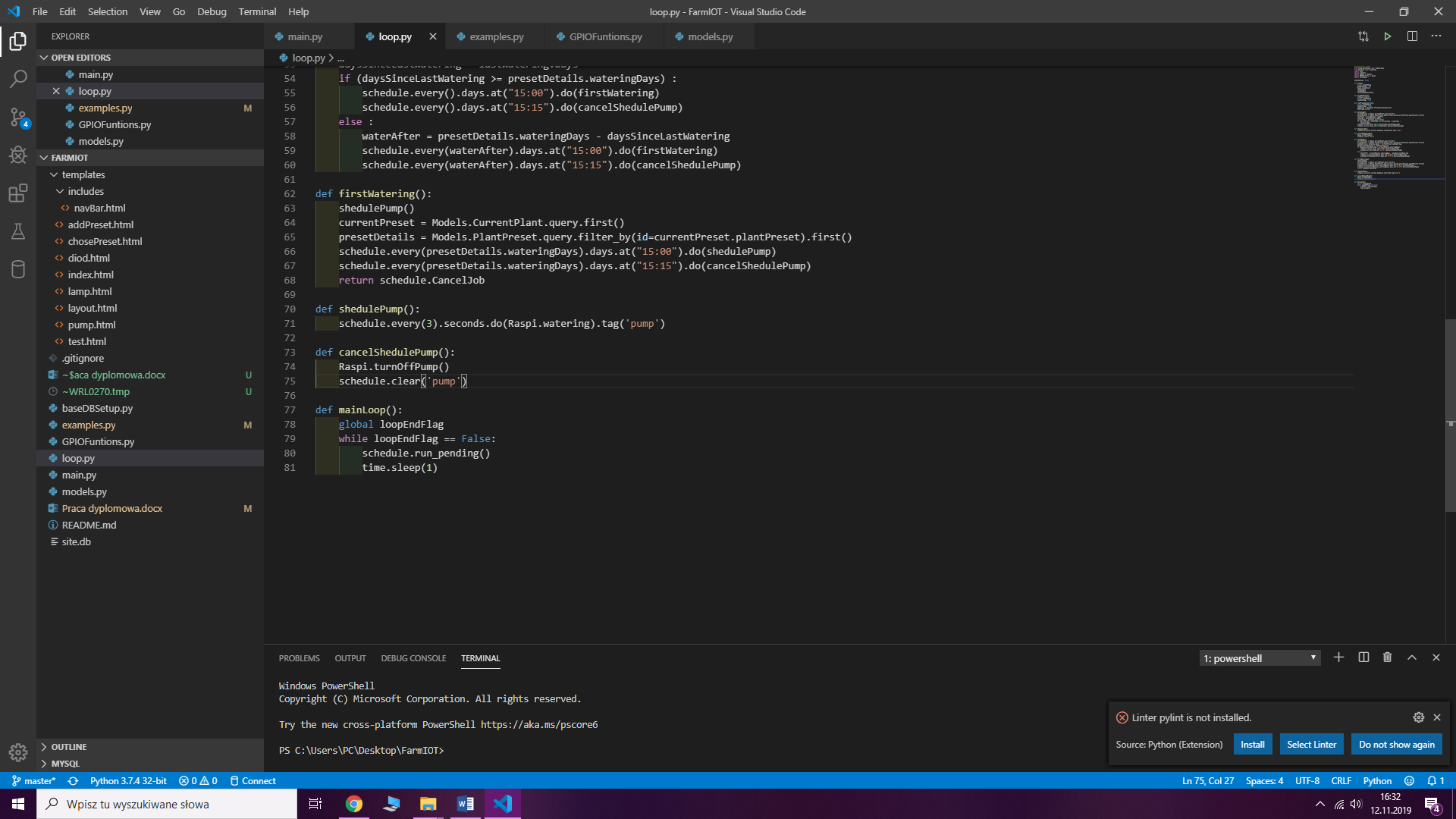
**Diagram UML :**



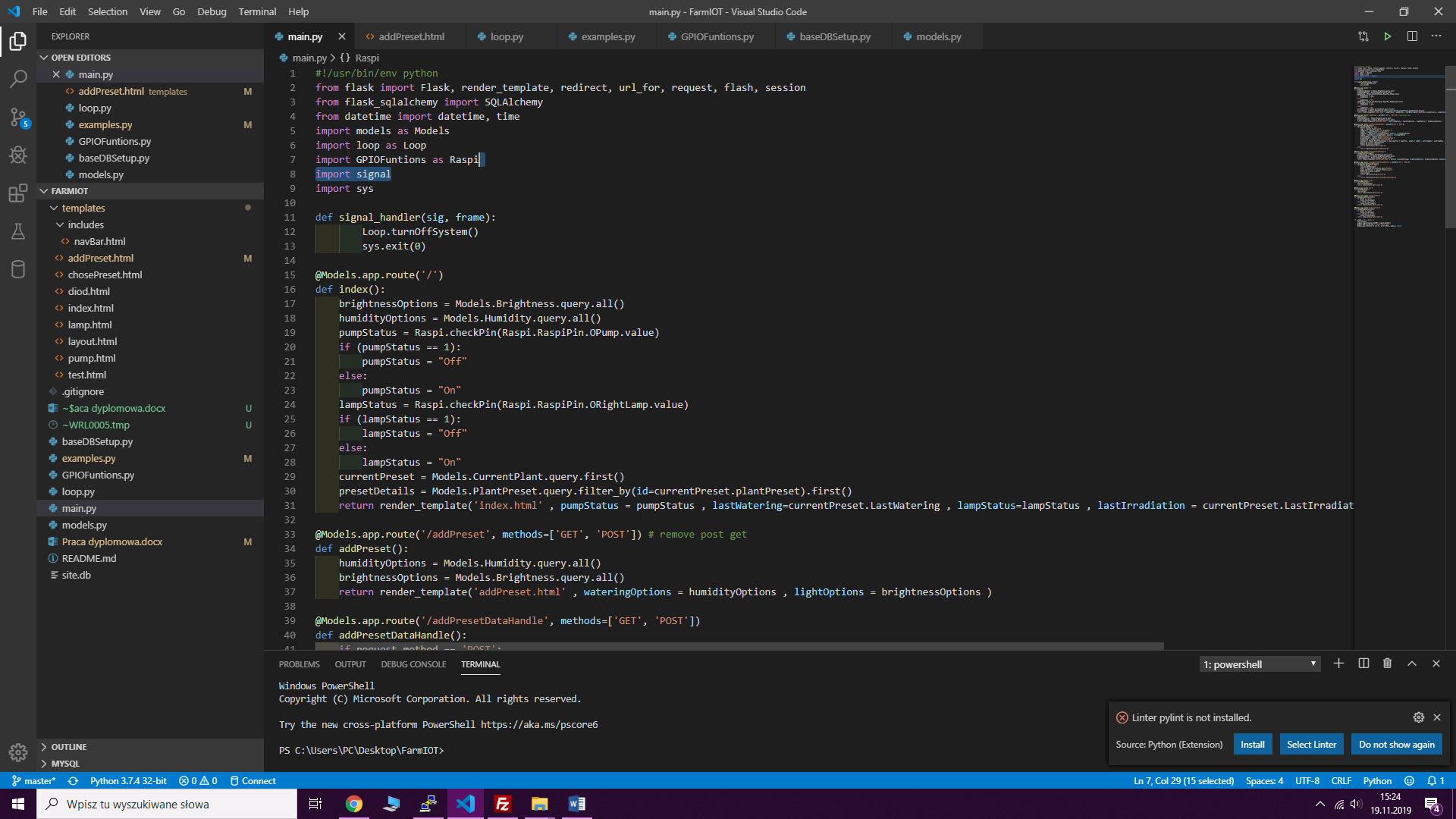
By aktywować zadania zapisane w planerze musimy wykonać kod :



Co blokowało by serwer przed odpowiedziami na rządzenia, więc trzeba ustawić pętle nadzorującą zadania jako osobny wątek, do tego użyłem modułu „threading” który pozwala na tworzenie interface`ów wątków ponad nisko poziomowym modułem „thread”.

Dla wygody stworzyłem globalną flagę która pozwala na zakończenie wątku by ten mógł być zresetowany bądź po prostu wyłączony razem z serwerem.  


Do wyłączenia z serwera korzystamy z sygnału SIGINT (ctrl+c) więc trzeba też zadbać o to by sygnał wyłączał cały serwer w tym też pętle nadzorującą, do tego skorzystałem z modułu „singal” który pozwala na przechwycenie sygnału i wyłączenie wszystkiego w wybranej przez programistę kolejności.



Tutaj upewniam się że system nadzorujący zostanie wyłączy po czym kończę pracę serwera.

## 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 swojego rodzaju dokumentacja jako że poza samym spisem endpointów mamy też przykładowe dane wejściowe.

W testach możemy sprawdzić :

* kod odpowiedzi
* wartości w nagłówku
* wartości w ciele
* czas wykonania

Co pozwala nam na zapewnienie że API spełnia wymakania funkcjonalne jak i nie funkcjonalne.

## 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 oraz więcej wiedzy o danej roślinie.

Ważne jest by przedłużyć żywotności czujnika wilgoci gleby poprzez dawanie mu zasilania wyłącznie w czasie pomiaru wartości, co trzeba zrobić odpowiednio wcześniej by mcp3008 nie pobrał błędnego wyniku ze względu na swoje opóźnienie.

Zabezpieczenie danych oraz połączenia z serwerem.

Zbieranie informacji o roślinie i wyświetlanie ich w postaci wykresu na stronie głównej.

# 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/>

<https://raspberrypi-aa.github.io/session3/spi.html>

<https://refactoring.guru/design-patterns/builder>

## Książki

Samouczek HTML – Karol Wierzchołowski

Learning Python 5ed – Mark Lutz

Mastering the Raspberry Pi - Warren Gay

## Artykuły

PYTHON: A PROGRAMMING LANGUAGE FOR SOFTWARE INTEGRATION AND DEVELOPMENT - M. F. SANNER[1]