**Poradnik Visual Studio Code + Python + PlatformIO + GIT**

**Radosław Smalec ver 0.1, 23.02.2022**

VS Code to rozbudowany notatnik, do którego można doinstalować wiele różnych dodatków ułatwiających pracę lub umożliwiających programowanie w różnych językach, na różny sprzęt.

[Download Visual Studio Code - Mac, Linux, Windows](https://code.visualstudio.com/Download)

Po pobraniu instalujesz sobie dodatki w zależności czym będziesz się zajmował. W naszym przypadku będzie to na pewno PlatformIO oraz polecam Ci też dodatek do pythona, jednak musisz wtedy i tak zainstalować interpreter pythona osobno, VS Code będzie tylko notatnikiem, gdzie piszesz kod i możesz go uruchamiać.

Obraz zawierający tekst, monitor, zrzut ekranu, ekran

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, ekran, srebrny

Opis wygenerowany automatycznie

Do tego wspomniany interpreter pythona oraz manager pakietów pip przez który będziesz mógł ściągać moduły (czyli biblioteki pythonowe dodane do oficjalnego repozytorium z kodem)

Interpreter: [Download Python | Python.org](https://www.python.org/downloads/)

Pip: <https://bootstrap.pypa.io/get-pip.py>

Instalacja pipa to nie jest tylko kliknięcie, potrzebujesz do tego już wcześniej zainstalowanego pythona i dodanego do zmiennej środowiskowej PATH, później będziesz musiał dodać pozostałe programy też w to miejsce więc od razu zaznaczam Ci je wszystkie:

Obraz zawierający tekst, zrzut ekranu, wewnątrz

Opis wygenerowany automatycznie

1. Interpreter pytona
2. Miejsce z różnymi skryptami pythonowymi, między innymi też pip-em
3. Program do obsługi GITa
4. VS Code

Dodanie czegoś do PATH powoduje, że możesz uruchamiać taki program z terminala bez potrzeby podawania pełniej ścieżki dostępu za każdym razem, czyli piszesz sobie w terminalu *python* i interpreter pythona startuje bez pytania.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Wracając do instalacji pip-a, kopiujesz tekst z podanej strony, zapisujesz w pliku *get-pip.py*, wchodzisz sobie w lokalizację (jak to zrobić opis poniżej), gdzie go zainstalowałeś przez terminal i wywołujesz komendę *python get-pip.py*, czyli uruchamiasz skrypt przed chwilą zapisany w pliku. Jak już dodasz sobie pip-a do PATH to będziesz mógł instalować nim moduły za pomocą polecenia, np. moduł o nazwie *pyserial* do zautomatyzowania komunikacji przez port COM instalujesz sobie pisząc w terminalu *pip install pyserial.*

Co do wchodzenia w lokalizację to jeżeli nie znasz żadnych komend do terminala to niech te dwie będą dla Ciebie dobrym początkiem *cd* – przenosi Cię do lokalizacji którą wpisałeś w komendzie, *cd ..* przenosi Cię do foldera o jeden wcześniej, *dir* wyświetla zawartość folderu, w którym jesteś. Zobacz na kilka przykładów poniżej.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Przydatne jest używanie klawisza *Tab*, działa to jak autouzupełnianie, jeżeli napiszesz przykładowo *cd Doc* oraz naciśniesz *Tab*, a w lokalizacji, której jesteś jest jakiś folder, który się tak zaczyna to w terminalu pojawi Ci się napis *cd Documents*.

Ale dość już o pythonie, wracamy do PlatformIO, jest to dodatek, przez który będziesz mógł programować wiele różnych procesorów, między innymi Atmegi i Attiny (chociaż w przypadku niektórych Attiny miałem problemy, może już to poprawili), ESP32, STM32 i wiele innych, o których pewnie nawet nie słyszeliśmy. Można też bez problemy programować Arduino, które tak naprawdę sprowadza się do procesora (często jakiejś Atmegi) połączonej z mostkiem USB-UART, który pozwala rozpoznawać urządzenie jako port COM w kompie i programować przez USB.

Obraz zawierający tekst, zrzut ekranu, monitor, czarny

Opis wygenerowany automatycznie

Jak klikniesz *Finish* to trzeba czasami trochę poczekać, aż środowisko przemieli i wygeneruje CI szablon, ale spokojnie, tak to czasami wygląda, szczególnie przy pierwszym uruchomieniu. PlatformIO jest od Arduino IDE tylko o krok bardziej skomplikowane, co daje większe możliwości konfiguracji. Pod spodem środowiska dzieje się to samo co w Arduino IDE, jest wykorzystywany toolchain dedykowany do danego procesora, generowane są pliki make, idzie linkowanie bibliotek, kompilacja i wgrywanie do procka.

Miejsca do których na początku będziesz zaglądał to plik *main.cpp*, gdzie piszesz swój kod oraz plik *platformio.ini*, który jest tym samym co wyklikiwanie z rozwijanych list w Arduino IDE typu procesora, portu komunikacyjnego itp. Tworząc projekt, środowisko tak naprawdę konfiguruje Ci plik *platformio.ini* i ściąga potrzebne do tego toolchainy, ale tym kompletnie nie musimy się przejmować na tym etapie.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Na dole na pasku masz dwa przyciski do buildowania projektu oraz do wgrywania, po naciśnięciu w wewnętrznym terminalu VS Code widzisz cały log kompilacji. Jest to też normalny terminal (jeżeli terminal CI się nie wyświetla to wciśnij CTRL + J), w którym możesz obsługiwać różne komendy, ale myślę, że to sam sobie będziesz rozgryzał podczas pracy jak już poczujesz, że może Ci się to do czegoś przydać. Zauważ, że zmianą w stosunku do Arduino IDE jest includowanie biblioteki Arduino.h, musi ona być jawnie zadeklarowana, żebyś miał dostępne wszystkie arduinowe funkcje do rozwoju softu.

Obraz zawierający tekst, zrzut ekranu, monitor, ekran

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Wracając do pliku platformio.ini to możesz sobie popatrzeć na dokumentację PlatformIO i tego jakie opcje można tu obsługiwać.

[“platformio.ini” (Project Configuration File) — PlatformIO latest documentation](https://docs.platformio.org/en/latest/projectconf/)

Natomiast na początek wystarczy Ci znajomość dwóch podstawowych poleceń: *upload\_port* oraz *upload\_speed*, które dobrze jest ręcznie dopisać do pliku, bo domyślnie ich nie ma.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Zapis *upload\_port = COM1*, oznacza, że wgrywanie będzie pod port COM1 i tylko tego portu będzie toolchain szukał, natomiast zapis *upload\_port = COM[348]* oznacza, że próba wgrania będzie pod jeden z portów 3, 4 lub 8. Szczególnie to jest przydatne jeżeli jeden kod testujesz na fizycznie różnych płytkach i każda z płytek wykrywana jest jako inny numer portu, wtedy nie musisz nic zmieniać w konfiguracji a jedynie skupiasz się na kodzie.

Jeszcze poruszę kwestię uruchamiania projektów, które masz już stworzone. Ponieważ trzeba uruchomić cały katalog, a nie tylko pojedynczy plik, to metody stosuję dwie. Albo File->Open Folder i szukam folderu z moim projektem, czyli w moim przypadku pod ścieżką:

C:\Users\RSMALEC\Documents\PlatformIO\Projects\Test\_dupa

Albo wchodzisz przez terminal pod powyższą ścieżkę i wywołujesz polecenie *code* ., dla tego polecenia ważne jest, żeby VS Code było dodane do PATH.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Metoda druga jest o tyle dobra, bo otwarty pod daną ścieżką terminal będzie nam potrzebny do obsługi GITa o czym już za chwilę.

Jeżeli chodzi o GITa to musisz wiedzieć i rozróżniać dwie rzeczy. GIT to narzędzie do zarządzania kodem, umożliwia łatwą współpracę nad kodem przez kilka osób, kontrolę co zostało zmienione, gromadzenie różnych wariantów kodu i w razie nie trafionych zmian i pomysłów, możesz sobie luźno wrócić do jakiejś poprzedniej wersji i zacząć pracę od jakiegoś wcześniejszego miejsca w czasie. GIT obsługuje sporo różnych komend, ale na początek wystarczy ich zaledwie kilka. Natomiast druga sprawa to serwery, gdzie ten kod gromadzisz. Dwa najbardziej popularne to GitHub i GitLab. Upewnij się, że masz konto na tym pierwszym, bo z tego będziemy korzystali.

Pierwsza kwestia to ściągnij sobie narzędzie do GITa.

[Git for Windows](https://gitforwindows.org/)

To narzędzie dodaj sobie do zmiennej PATH, tak jak pozostałe programy. W wyglądzie przypomina ono terminal windowsowy, ale z tą drobną różnicą, że obsługuje ono komendy linuxowe a nie windowsowe, komenda *cd* to nadal *cd*, ale *dir* to już *ls*. Natomiast to nie jest tak upierdliwe jak to, że linxowy format ścieżek jest inny niż windowsowy, bo zamiast backslashy \ ma slashe. Trzeba o tym pamiętać, jeżeli chciałbyś przekopiować sobie ścieżkę z windowsa i wkleił ją do komendy cd w terminalu GITa to musisz jeszcze zmienić backslashe na przeciwne. Ale to tylko taka drobna dygresja.

Żeby skonfigurować sobie GITa na początek to musisz wklepać kilka poleceń, gdzie ustawiasz dane do swojego konta na GitHubie, pokażę Ci co ja wpisywałem:

*git config –global user.name=r-smalec*

*git config –global user.email=rsmalec@kplabs.pl*

Teraz dodamy sobie repozytorium na GitHuba. Repozytorium to jest po prostu folder z naszym kodem na serwerze gita, który zawiera określony projekt.

Obraz zawierający tekst, zrzut ekranu, monitor, czarny

Opis wygenerowany automatycznie

W nazwie repozytorium używamy tej samej nazwy co nazwa projektu, dobrze jest dodać opis oraz plik README.md, który w gotowym projekcie opisuje użytkownikowi co to za projekt, co powinien o nim wiedzieć, jak wygląda jego używanie, integracja z kodem, jeżeli jest to jakiś moduł/biblioteka.

Obraz zawierający tekst, zrzut ekranu, ekran

Opis wygenerowany automatycznie

Kiedy repozytorium jest już gotowe, to możemy przerzucić tam nasze pliki z VS Code. Najłatwiej otworzyć folder z projektem, zaznaczyć wszystkie pliki i przeciągnąć je na miejsce w GitHubie.

Obraz zawierający tekst, monitor, wewnątrz, zrzut ekranu

Opis wygenerowany automatycznie

W ten sposób zrobiłeś pierwszego commita, czyli wypchnąłeś pliki na serwer. Warto dodać opis co się zmieniło, dodało w każdym kolejnym commicie, który będziesz robił. Tu można wpisać po *prostu initial commit*. I klikasz *commit changes*.

I teraz żeby najłatwiej sobie wszystko skonfigurować, a przy okazji żebyś poznał polecenie do ściągania repozytoriów na kompa zarówno swoich jak i znalezionych w necie w których chcesz pogrzebać to usuń z dysku projekt z VS Code, albo przenieś go sobie w jakieś inne miejsce. Kopiujesz sobie teraz link do repo i używasz komendy *git clone <link>*. Ważne, żeby przed klonowaniem być w folderze, do którego chce się to ściągnąć.

Obraz zawierający tekst, monitor, zrzut ekranu, ekran

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Teraz wchodzimy do ściągniętego folderu i uruchamiamy projekt w VS code.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Wprowadź sobie jakieś zmiany do maina, np. dodaj zmienną. Zobaczysz, że środowisko zaznaczy Ci ten plik na żółto z literką M jako zmieniony, a jeżeli utworzysz jakiś plik to na zielono z literką U.

Obraz zawierający tekst, zrzut ekranu, monitor, ekran

Opis wygenerowany automatycznie

Jak widzisz powyżej, czasami przy o otwieraniu jakiś wewnętrzy plik środowiska załaduje się w zły sposób i jest na czerwono, zazwyczaj wystarczy w niego wejść i wyjść, środowisko przeładuje sobie co trzeba i już nie wyświetla problemu.

Jak już wprowadzisz zmiany i chcesz wypchnąć pliki z powrotem na serwer to wprowadzasz komplet komend:

*git add .*

*git commit -m ‘nazwa commitu, co zostało dodane, zmienione żebyś wiedział jak następny raz siądziesz do kodu’*

*git push -u origin*

Czasami pojawia się problem z zakończeniami linii w plikach, jest windowsowe LF, zamiast linuxowego CRLF, dlatego potrzebna jest komenda *git config –global core.autocrlf true* i dalej idzie już normalnie.

Obraz zawierający tekst, zrzut ekranu, ekran

Opis wygenerowany automatycznie

I tak to powinno wyglądać dalej.

Obraz zawierający tekst

Opis wygenerowany automatycznie

I myślę, że tyle na sam początek wystarczy. Jak ta wiedza Ci się ułoży to ogarnij czym są branche w gicie, jak na nich możesz gromadzić alternatywne, robocze wersje oprogramowania i potem łączyć (mergować) z główną gałęzią (to od niej na powyższych screenach masz niebieski napis *main* lub wg starej nomenklatury *master*).

Obraz zawierający tekst

Opis wygenerowany automatycznie