POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

KIERUNEK: INFORMATYKA

Praca dyplomowa inżynierska

System pomiaru wilgotności gleby roślin doniczkowych

Autor: Rafał Brauner

Kierujący pracą: dr inż. Dariusz Myszor

Gliwice, luty 2022

Spis treści

Streszczenie

Celem niniejszego projektu było stworzenie systemu umożliwiającego monitorowanie wilgotności gleby, monitorowanie temperatury i wilgotności otoczenia, nawadnianie gleby oraz ustawienie automatycznego nawadniania w zależności od wilgotności gleby. Projekt ten ma na celu pomóc użytkownikom w zaopiekowaniu się swoimi roślinami doniczkowymi, wykorzystując do tego gotowe do użytku urządzenie oraz prostą w obsłużeniu aplikację desktop'ową. Po przeanalizowaniu problemu, w tej pracy został wykorzystany zestaw EcoDuino, płytka została zaprogramowana w języku Arduino bazującym na języku C++, a aplikacja desktop'owa zaprojektowana i napisana w języku Dart przy użyciu zestawu narzędzi interfejsu użytkownika Flutter.

Słowa kluczowe: ecoduino, flutter, system, rośliny doniczkowe, wilgotność gleby

Rozdział 1

Wstęp

Temat niniejszej pracy obejmuje dość powszechny problem goszczący w wielu domach, a mianowicie zbyt rzadkie albo odwrotnie zbyt częste podlewanie roślin doniczkowych. Ludzie żyją w świecie, w którym liczy się kariera, nauka, rodzina i zapominają o niektórych obowiązkach, takich jak podlewanie roślin, co skutkowało ich uschnięciem i obumarciem. Po takiej sytuacji ludzie często próbują lepiej zaopiekować się innymi roślinami, poświęcając im większą uwagę, poprzez częstsze podlewanie, co niestety również nie jest dobre dla rośliny, która potrzebuje zrównoważonego poziomu wilgotności i tlenu w glebie. Korzenie przelanych roślin obumierają, liście żółkną, aż cała roślina traci żywot.

Przy takim problemie, mogą być przydatne wszelkiego rodzaju cykliczne przypomnienia w kalendarzu, karteczki naklejone w zasięgu oka czy też inne sposoby. Do takich problemów można także wykorzystać technologię, która jest wszechobecna, od serwerowni, poprzez komputery osobiste, telefony, smartwatch'e, aż po elementy inteligentnego domu jak np. automatyczne oświetlenie pomieszczenia, inteligentne lodówki. Wszystkie te rzeczy pomagają ludzkości w codziennym życiu, pracy, pozwalając automatyzować powtarzalne czynności, zarządzać różnymi urządzeniami, a także monitorować różne współczynniki np. wilgotność czy temperatura powietrza. Dlatego można wykorzystać technologię do rozwiązania powyższego problemu.

Celem tego projektu było wykorzystanie technologii, aby stworzyć system pomagający w opiekowaniu się roślinami doniczkowymi. System ten w zamyśle miał

być prosty w montażu oraz obsłudze przez przeciętnego użytkownika, umożliwiając jednocześnie wszystkie najpotrzebniejsze do zarządzania i monitorowania możliwości.

Praca ta obejmuje zakres przygotowania fizycznego urządzenia, zaprogramowania go przy użyciu ogólnodostępnych narzędzi oraz stworzenie aplikacji desktop'owej umożliwiającej dostęp do urządzenia w celu monitorowania jego czujników, a także zarządzania nim. Dodatkowo przed rozpoczęciem pracy została przeprowadzona analiza tematu, a po wykonaniu zostały przeprowadzone odpowiednie testy.

Cały projekt składa się z kilku części jego realizacji przedstawionych poniżej:

- **Analiza tematu** - przeanalizowanie problemu, dobór odpowiednich narzędzi oraz technologii użytych w ramach tego projektu,
- **Wymagania i narzędzia** - stworzenie wymagań funkcjonalnych i niefunkcjonalnych, zaprojektowanie odpowiednich diagramów UML, wstępne ustalenie metodyk pracy,
- **Specyfikacja zewnętrzna** - ustalenie wymagań sprzętowych i programowych, przedstawienie sposobu instalacji, konfiguracji, podłączenia urządzeń, sposób obsługi systemu, przykład działania,
- **Specyfikacja wewnętrzna** - przedstawienie technicznych aspektów systemu, takich jak przedstawienie idei, architektury systemu, użytych w aplikacji bibliotek, przedstawienie najważniejszy klas, widget'ów, szczegóły implementacji najważniejszych fragmentów kodu, zastosowane ideologie, stworzenie diagramów UML,
- **Walidacja i weryfikacja** - przedstawienie sposobów testowania poszczególnych aplikacji, a także całego systemu, dodatkowo przedstawienie wykrytych przy testach błędów, oraz sposobu ich naprawy.

Rozdział 2

Analiza tematu

W poniższym rozdziale przedstawiony jest problem, a także brane pod uwagę rozwiązania. Dodatkowo jest przedstawione omówienie i analiza jednego, wybranego w ramach projektu.

2.1 Sformułowanie problemu

Dana praca rozwiązuje problem związany z monitorowaniem wilgotności gleby roślin doniczkowych, a także ich manualnego oraz automatycznego podlewania. Podczas pracy została wykonana dokonana analiza tematu, z wybraniem odpowiedniego rozwiązania oraz użytych technologii.

2.2 Przedstawienie rozwiązań

Pierwszym pomysłem było użycie użycie zestawu EcoDuino wraz z Raspberry Pi oraz przygotowaną do zarządzania aplikacją webową napisaną od strony backend'owej przy użyciu języka programowania Java lub Kotlin, a do frontend'owej przy użyciu framework'a Vue.js oraz szablonów AdminLTE.

Drugim pomysłem było użycie tego samego zestawu EcoDuino, który został przedstawiony w pierwszym rozwiązaniu, a do zarządzania urządzeniem przygotowaną aplikacją desktop'ową napisaną w zestawie narzędzi Flutter.

2.3 Analiza wybranego rozwiązania

Po przeprowadzeniu analizy zostało użyte drugie rozwiązanie. Jednym z bardziej przemawiających argumentów był niższy koszt wdrożenia. Hardware użyty w tym rozwiązaniu sprowadzał się tylko do zakupu jednego urządzenia zamiast dwóch, a także generował mniejsze zużycie prądu. Dodatkowym benefitem była prostota całego rozwiązania. W celu zarządzania rośliną potrzebne jest jedno urządzenie przyłączone do doniczki, a także komputer na którym dostępna jest aplikacja pozwalająca na zarządzanie urządzeniem. Komunikacja między urządzeniem zestawu EcoDuino a aplikacją desktop'ową przebiega z wykorzystaniem przewodu microUSB. Po analizie i porównaniu z kilku dostępnych opcji takich jak WiFi, czy też Bluetooth, połączenie przewodowe jest bezpieczniejsze i stabilniejsze, co zadecydowało o jego wyborze.

Urządzenie może być zasilane poprzez użycie gniazda DC, jak i również gniazda microUSB, ale jeśli jest potrzeba użycia pompy, wtedy konieczne już jest użycie gniazda DC. Do gniazda DC można podpiąć dostępny w zestawie koszyk na 6 baterii AA, lub zasilacz 6-12V. Takie rozwiązanie pozwala na mobilność całego systemu. Aplikacja na płytkę jest napisana w języku Arduino bazującym na Wiring i C/C++. W porównywaniu zostały objęte też takie rozwiązania jak PlatformIO czy też wykorzystanie języka C++ wraz z narzędziami do zaprogramowania płytek AVR, m. in. avr-g++. Ostateczny wybór jaki został wyznaczony to Arduino, ze względu na swoją prostotę względem konkurencji jak i w samym języku, dołączaniu zewnętrznych bibliotek, korzystaniu z monitorowania serial port'u, a także samego wgrywania programu na płytkę.

Od strony aplikacji dla końcowego użytkownika zostały użyte najnowsze i rozwijane technologie tak jak Flutter stworzony przez firmę Google. Podczas analizy rozpatrywany był także język programowania Kotlin firmy JetBrains, ale w porównaniu do Flutter'a, nie posiadał on tylu benefitów co użyty w projekcie konkurent. Flutter jest to zestaw narzędzi dla języka Dart i posiada ona bogaty asortyment widget'ów, pozwalających w prosty i szybki sposób stworzyć dobrze wyglądająca aplikację z motywem Material. Dodatkowo Flutter i język Dart są wieloplatformowe, dzięki czemu przyszłościowo jest możliwość przeniesienia przygotowanej aplikacji na inne platformy, w tym mobilne, przy niedużym nakładzie

pracy. Sam język Dart, jest także należącym do firmy Google, silnie typowanym językiem programowania, wspierającym zasadę null safety, dzięki czemu napisany w nim kod jest dużo bezpieczniejszy przed niepowołanymi błędami.

Cała komunikacja między programem na płytce, a aplikacją desktop'ową na komputerze przebiega poprzez użycie kabla microUSB. Rozwiązanie to zostało użyte, głównie ze względów na swoją bezawaryjność, a także prostotę i bezpieczeństwo. Dzięki temu program zawsze zadziała, a także zostanie obsłużony przez odpowiednie osoby.

Komunikacja przebiega na podstawie zmodyfikowanego protokołu HTTP. Zostały zachowane koncepcje wykonywania request'ów do płytki i zwracanych response'ów do aplikacji z danymi zwrotnym REST'owego API. Sam request został okrojony do wysłania metody GET i POST oraz route'a, mogącego zawierać w sobie parametry, a response zwraca JSON z statusem, kodem i danymi. Celem zastosowania takiego rozwiązania było użycie powszechnie znanego, używanego wzorca i uproszczenie go pod wymagania projektu, rezygnując z i tak niewykorzystanych funkcjonalności jakie oferuje protokół.

Rozdział 3

Wymagania i narzędzia

W niniejszym rozdziale przedstawione są wymagania i narzędzia użyte w ramach pracy. Na początku są omówione wymagania funkcjonalne i niefunkcjonalne, następnie przypadki użycia przedstawione na diagramie UML, a na samym końcu metodyka pracy nad projektem i jego implementacja.

3.1 Wymagania funkcjonalne i niefunkcjonalne

W celu uruchomienia projektu potrzebny jest zestaw EcoDuino z odpowiednio podłączonymi do płytki czujnikami, pompą i zasilaniem. Potrzebny jest też komputer z zainstalowanym systemem operacyjnym Windows 7 SP1 lub nowszy [3], aby uruchomić aplikację napisaną w Flutter. Dodatkowo warto zaopatrzyć się w 6 baterii AA w celu zasilenia pompy, alternatywnie można użyć zasilacza DC 6-12V. Przed uruchomieniem należy także pamiętać o przygotowaniu źródła wody (na przykład: pudełko, szklanka) i odpowiednio uzupełniania jego poziomu.

Program stworzony na płytkę z zestawu EcoDuino po uruchomieniu powinien:

- być dostępny 24/7,
- być niezawodnym, obsługiwać tylko poprawne request'y i zwracać odpowiednie response'y,
- w przypadku niepoprawnego request'a, zwrócić odpowiedni kod i komunikat błędu,

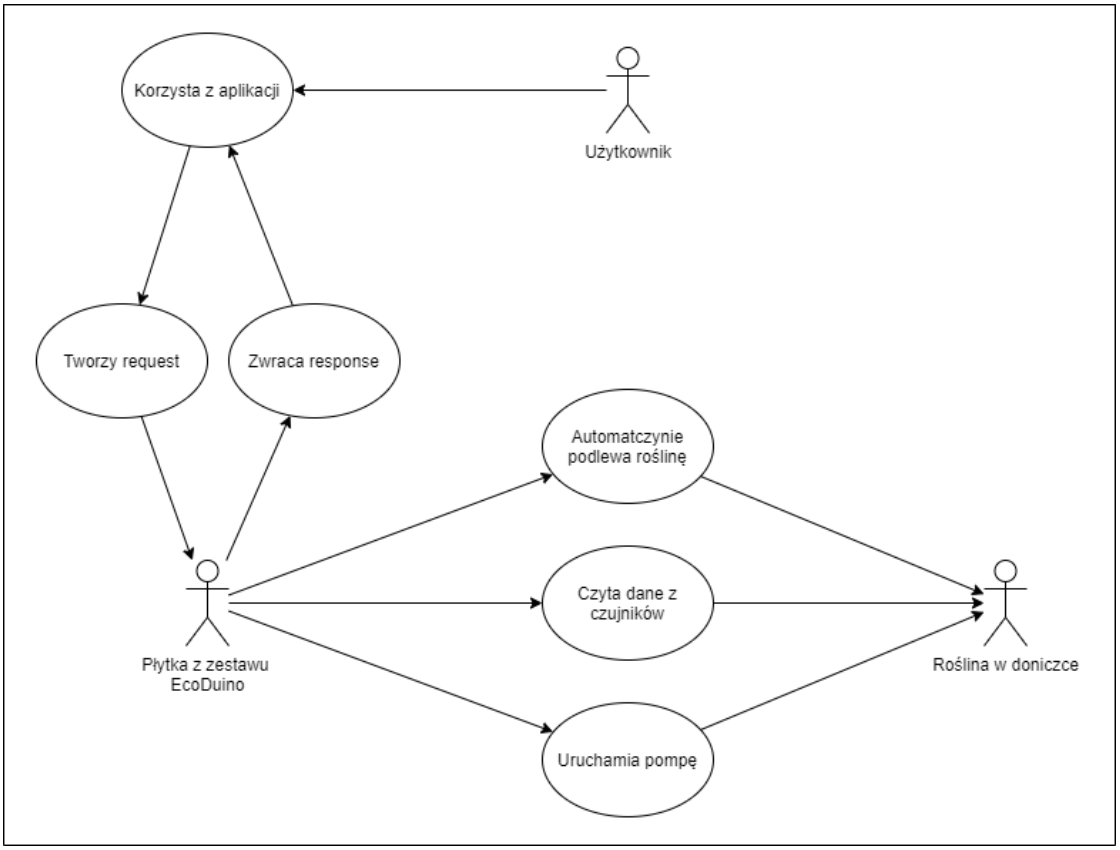
- kolejkować wszystkie żądania i wykonywać po koleii,
- posiadać możliwość zatrzymania request'u w dowolnym momencie,
- posiadać możliwość skonfigurowania automatycznego podlewania roślin.

Aplikacja desktop'owa po włączeniu powinna:

- być stabilna w działaniu,
- udostępniać prosty i przejrzysty interfejs użytkownika,
- pozwalać na wybranie podłączonej za pomocą przewodu microUSB płytki z zestawu EcoDuino, a także odświeżenia listy dostępnych urządzeń, w przypadku gdyby płytka nie została wylistowana,
- udostępniać funkcję monitorowania dostępnych czujników (w razie niedostępności jakiegoś z czujników, powinien być wyświetlony odpowiedni komunikat, na przykład "N/A"). Monitorowanie powinno działać asynchronicznie, nie blokując całego interfejsu aplikacji podczas sczytywania danych,
- udostępniać funkcję manulanego włączenia/wyłączenia pompy,
- udostępniać funkcję automatycznego podlewania rośliny z możliwością ustawienia zakresu wilgotności gleby.

3.2 Przypadki użycia

Przypadki użycia zostały przedstawione na rysunku 3.1.



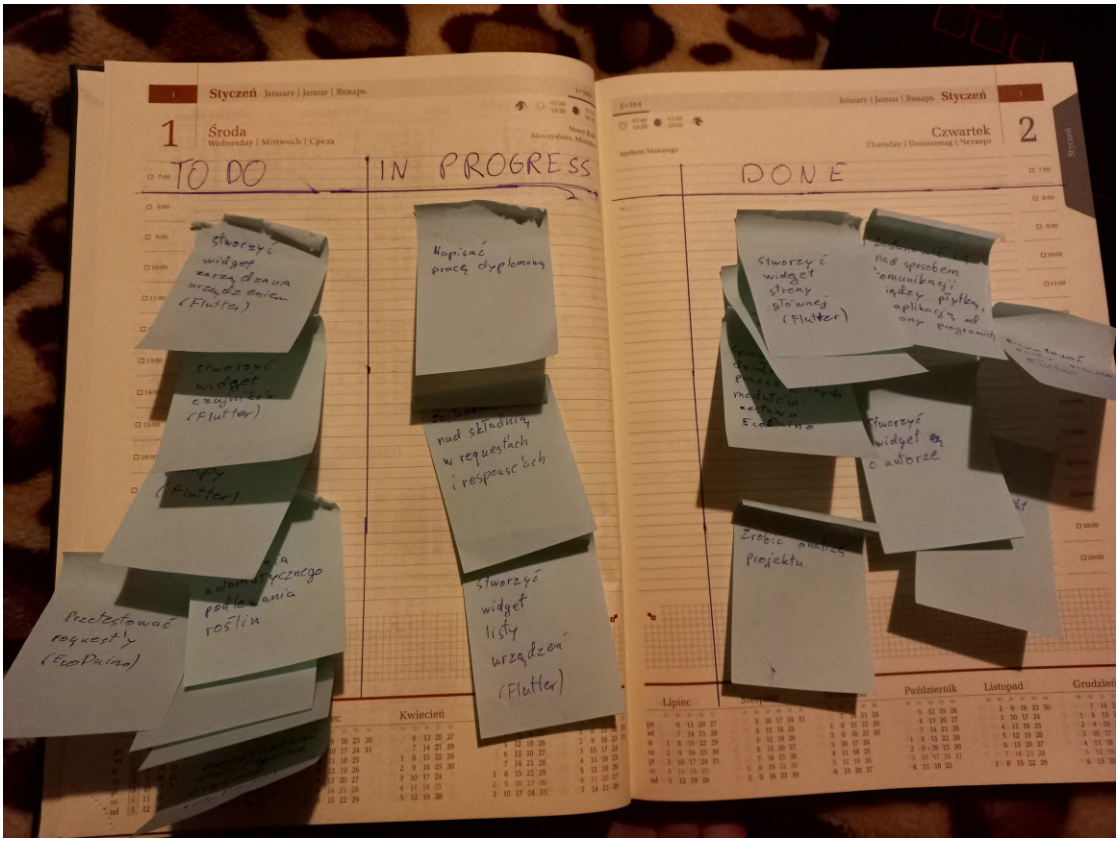
Rysunek 3.1: Model UML przypadków użycia.

Do przeprowadzenia walidacji poprawności działania systemu został użyty Serial Port Monitor w Arduino IDE, testy jednostkowe aplikacji dekstop'owej oraz testy manualne całości. Fizycznie do testów, w pierwszej kolejności zostało użyte napełnione wodą pudełko, a finalnie roślina doniczkowa, aby zweryfikować działanie na prawdziwym przypadku użycia.

3.3 Metodyka pracy nad projektowaniem i implementacją

Podczas pracy nad projektem została użyta metodyka Kanban, której głównym celem jest wizualizacja procesu wykonywania danego projektu. Aby skorzystać z

wyżej wspomnianej metodyki, należy przygotować fizyczną lub wirtualną tablicę z minimum 3 kolumnami (“To do”, “In progress” i “Done”), oraz karteczkami opisującymi poszczególne etapy/zadania projektu. Dane karteczki zostają przekładane z jednej kolumny na kolejną wraz z postępami.



Rysunek 3.2: Metodyka pracy nad projektem.

Projekt został zaplanowany według wyżej wspomnianej metodyki i realizowany małymi etapami (Rysunek 3.2). Nowe zadania zostały na bieżąco dodawane do kolumny “To do” w miarę rozwoju projektu. Pierwszymi zakończonymi etapami były analiza tematu, wraz z zebraniem wszelkich informacji i poszerzenia aktualnego stanu wiedzy o danym problemie i jego rozwiązaniach. Kolejnym etapem był zakup zestawu EcoDuino oraz przetestowanie jego modułów w celach weryfikacji działania, a także zdobycia wiedzy w jaki sposób użyć danych modułów. Następnie

zostały stworzone puste projekty, osobno dla programu płytki z zestawu EcoDuino oraz aplikacji desktop'owej, a także nauka nowej technologii. Kolejnym krokiem było stworzenie kilku prostych widoków w aplikacji Flutter'owej, a także zastanowienie się nad składnią w żądaniach i zwrotkach w komunikacji w systemie. Po przemyśleniu wyżej wspomnianego problemu został przygotowany i przetestowany za pomocą Serial Port Monitor'u program obsługujący kilka podstawowych request'ów takich jak odczyt danych z sensorów lub uruchomienie i zatrzymanie pompy. Następnie zostały przygotowane widoki w aplikacji desktop'owej do obsługi tych request'ów. Została dodana nowa funkcjonalność automatycznego podlewania po stronie programu na płytce jak i w aplikacji desktop'owej, po czym został przetestowany cały system. Podczas realizacji projektu zostały napotkane różne problemy, które zostały wyznaczone jako osobne zadania i w późniejszym czasie realizacji projektu zostały one naprawione. Jednym z największych problemów był problem związany z asynchronicznością aplikacji przy jednoczesnym odczytywaniu kilku czujników w tym samym czasie. Został on naprawiony wydzieleniem każdego żądania do osobnego wątku aplikacji.

3.4 Wykorzystane narzędzia i technologie

Podczas tworzenia projektu zostały użyte narzędzia i technologie przedstawione poniżej:

- Arduino - język programowania pozwalający tworzyć oprogramownie na różne płytki Arduino, jest bazujący na Wiring oraz C++, został zapoczątkowany w 2005 roku,
- Arduino IDE - środowisko programistyczne dla języka programowania Arduino, pozwala ono między innymi na napisanie kodu programu, pobranie dodatkowych bibliotek z repozytorium, kompilacje, wgranie programu na płytkę, wypalenie bootloader'a oraz użycie Serial Port Monitor'a,
- Flutter - zestaw narzędzi dla języka programowania Dart, pozwalających na napisanie natywnej, wieloplatformowej aplikacji z nowoczesnym design'em, został stworzony w 2017 roku przez firmę Google,

- Dart - język programowania cechujący się obiektowością, posiadaniem garbage collector'a, kompilacją zarówno do kodu natywnego, a także do JavaScript'u, został on stworzony w 2013 roku przez firmę Google,
- Visual Studio Code - edytor kodu źródłowego stworzony przez firmę Microsoft w roku 2015, posiada w sobie wiele wbudowanych funkcjonalności, między innymi podpowiadanie składni kodu, refaktoryzacja, debugger, kolorowanie składni i inne, dodatkowo można dodać nowe funkcjonalności instalując rozszerzenia. W projekcie został użyty do stworzenia aplikacji desktop'owej w Flutter'ze, dodatkowo została użyta wtyczka "Dart-Code.flutter",
- Windows 11 - system operacyjny stworzony przez firmę Microsoft w 2021 roku, został on wykorzystany podczas całego procesu tworzenia projektu i pracy dyplomowej, aplikacja na płytkę oraz aplikacja desktop'owa zostały skompilowane na tym systemie,
- diagrams.net - narzędzie do tworzenia diagramów UML, poprzednio nazywane było draw.io. Narzędzie zostało wykorzystane do utworzenia diagramu przypadków użycia,
- PlantUML - narzędzie do tworzenia diagramów UML przy użyciu napisanego kodu, zostało stworzone w 2009 roku przez Arnaud Roques'a. Narzędzie zostało wykorzystane przy tworzeniu diagramów UML klas, zarówno w programie na płytkę z zestawu EcoDuino, a także w aplikacji desktop'owej,
- hpp2plantuml - narzędzie konsolowe pozwalające wygenerować kod diagramu UML dla narzędzia PlantUML z plików języka C++. Narzędzie zostało wykorzystane do utworzenia diagramu klas w programie na płytkę z zestawu EcoDuino,
- dcdg - narzędzie konsolowe, napisane w języku Dart, pozwalające wygenerować kod diagramu UML dla narzędzia PlantUML z plików języka Dart. Narzędzie zostało wykorzystane do utworzenia diagramu klas aplikacji desktop'owej.

Rozdział 4

Specyfikacja zewnętrzna

W tym rozdziale ukazana jest specyfika zewnętrzna projektu, tzn. przedstawione są wymagania zarówno sprzętowe jak i programowe, schemat płytki, diagram zestawu EcoDuino, a także ukazany jest sposób instalacji poszczególnych części zestawu oraz instalacja programu na płytkę czy też aplikacji desktop'owej. Dodatkowo opisany jest sposób obsługi, kwestie bezpieczeństwa, a także przykładowy scenariusz działania.

4.1 Wymagania sprzętowe i programowe

Wymagania sprzętowe i programowe dla danego projektu jakie trzeba spełnić to:

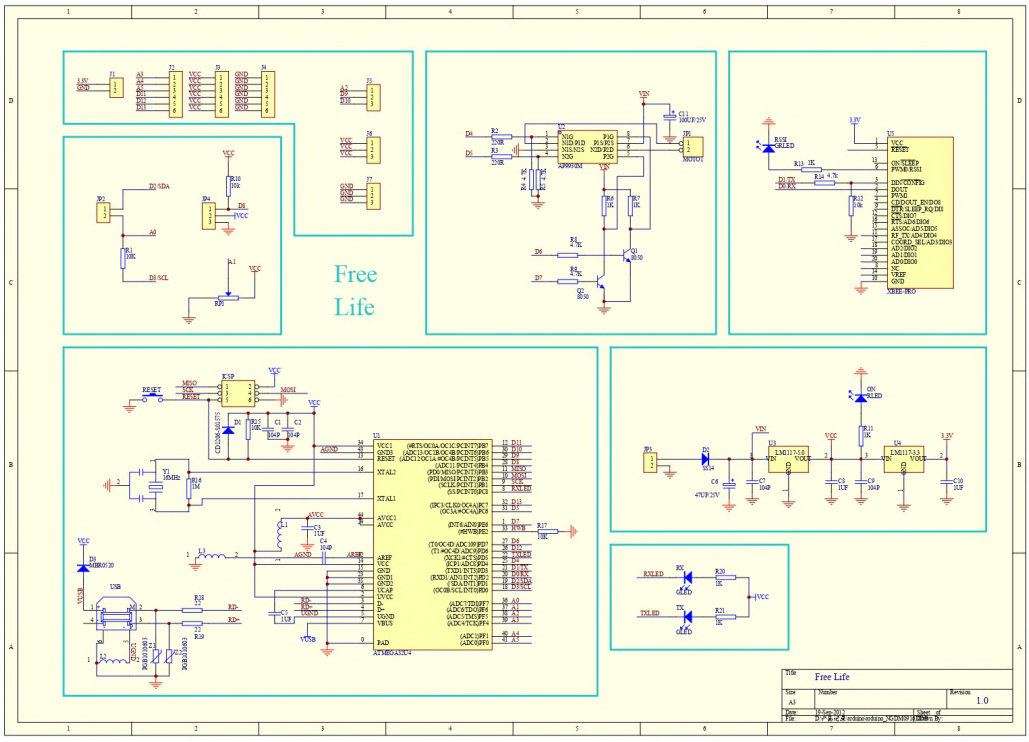
- posiadanie zestawu EcoDuino,
- posiadanie komputera z zainstalowanym systemem Windows 7 SP1 lub nowszy [3],
- opcjonalnie posiadanie 6 baterii AA lub zasilacza DC 6-12V w celu zasilenia zestawu EcoDuino oraz pompy (sam zestaw EcoDuino można alternatywnie zasilić przy użyciu przewodu microUSB, ale pompa potrzebuje już większego zasilania).

Wykorzystany w projekcie zestaw EcoDuino składa się z kilku elementów, w których skład wchodzą:

- płytka główna FreeLife bazująca na Arduino Leonardo, posiadająca:
  - mikrokontroler ATmega 32u4,
  - 5 pinów cyfrowych I/O,
  - 4 wejścia analogowe,
  - gniazdo do czujnika wilgotności,
  - gniazdo do podłączenia pompy,
  - gniazdo do podłączenia modułu komunikacji Xbee,
  - gniazdo micro USB,
- pompa o parametrach:
  - napięcie zasilania pompy 4.5V - 12V,
  - moc maksymalnie 5W,
  - przepływ 100 - 350 L/H,
- obudowa do płytki,
- czujnik wilgotności gleby,
- czujnik temperatury i wilgotności powietrza DHT11,
- wężyk do wody,
- koszyk na 6 baterii AAA,
- przewód microUSB,
- dwa śrubokręty,
- gniazdo DC,
- zestaw przewodów.

4.2 Schemat płytki

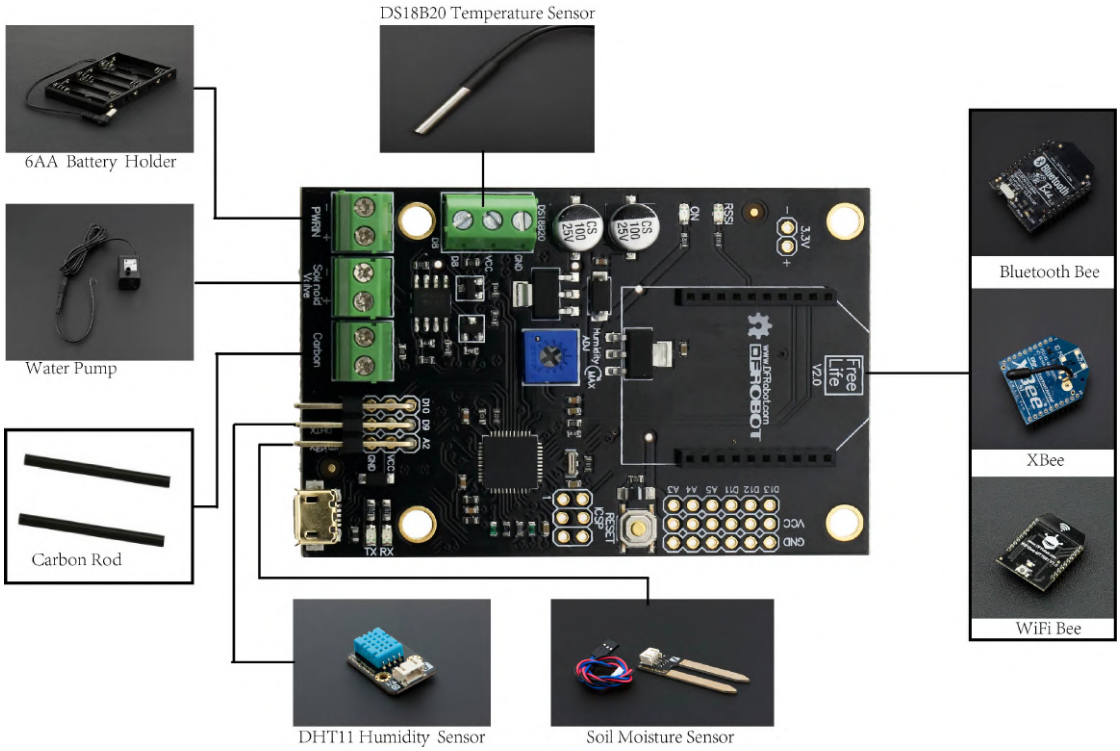
Schemat płytki z zestawu EcoDuino został przedstawiony na rysunku 4.1.



Rysunek 4.1: Schemat płytki. [8]

4.3 Diagram zestawu EcoDuino

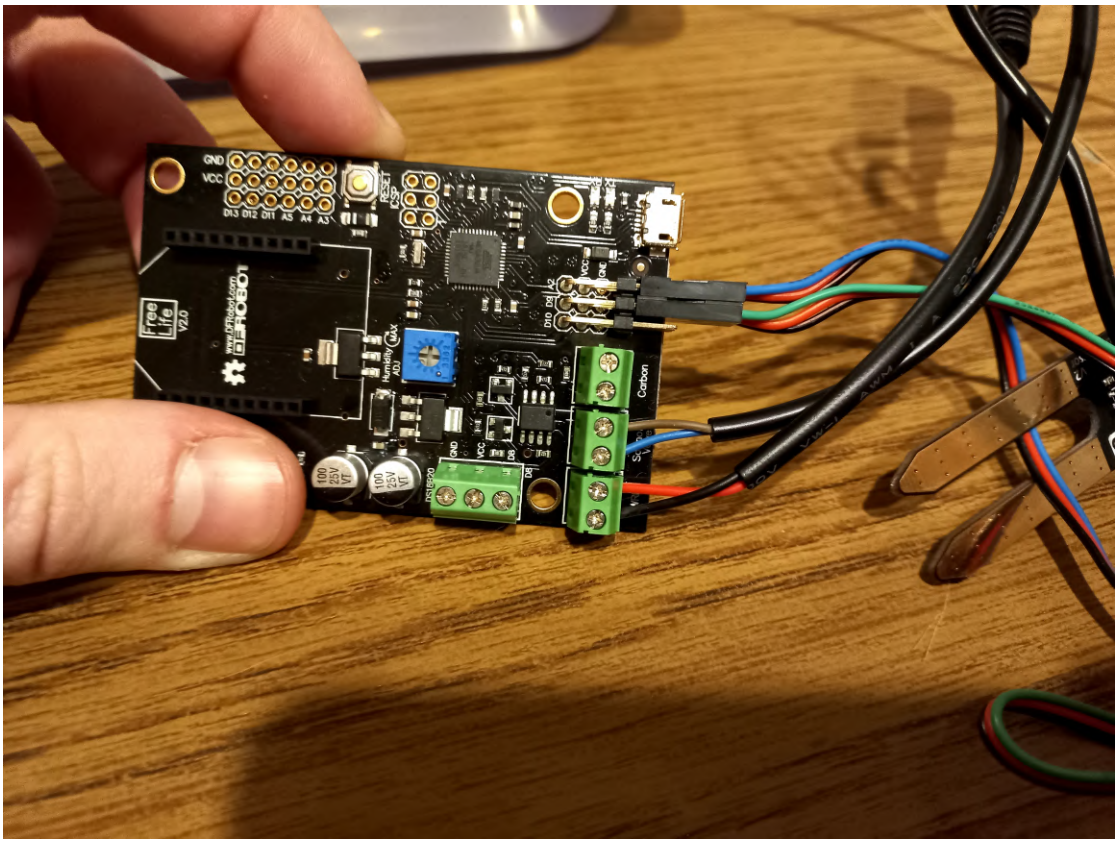
Diagram zestawu EcoDuino został przedstawiony na rysunku 4.2.



Rysunek 4.2: Diagram zestawu EcoDuino. [1]

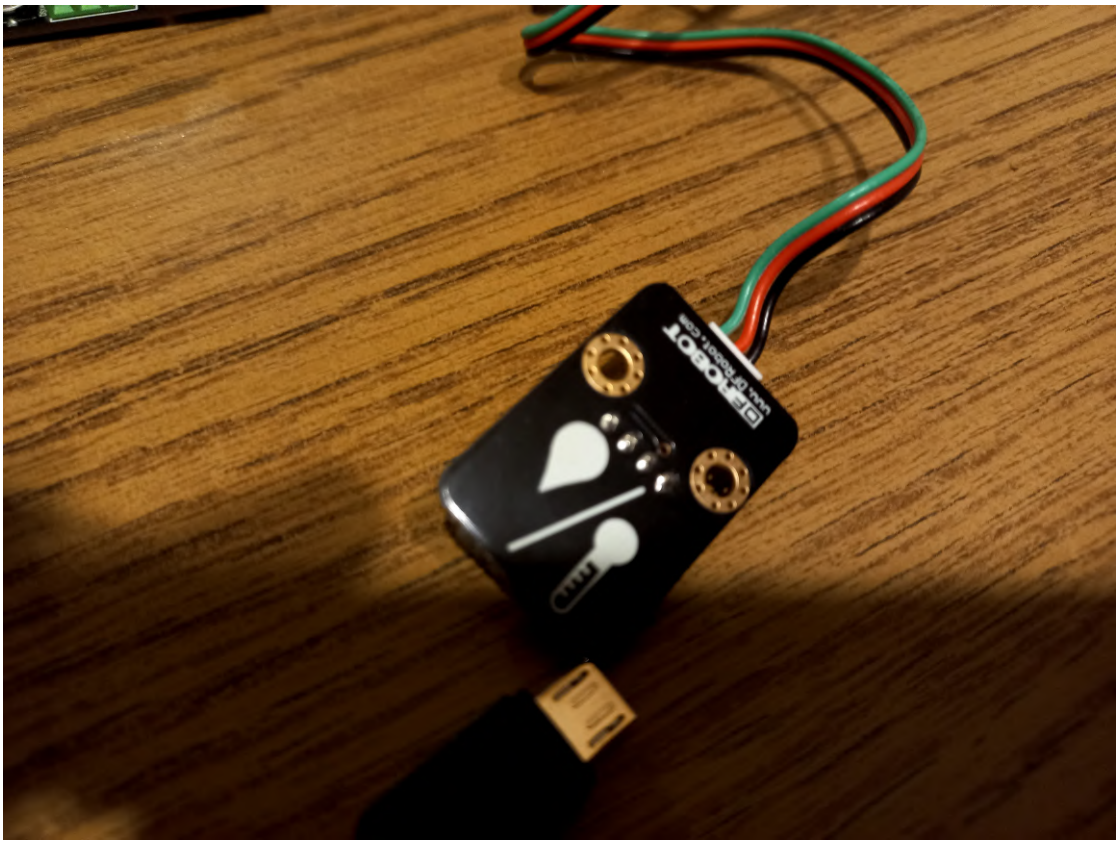
4.4 Sposób instalacji

Aby zestaw EcoDuino działał, należy najpierw podłączyć prawidłowo wszystkie przewody, kolejność obojętna (Rysunek 4.3).



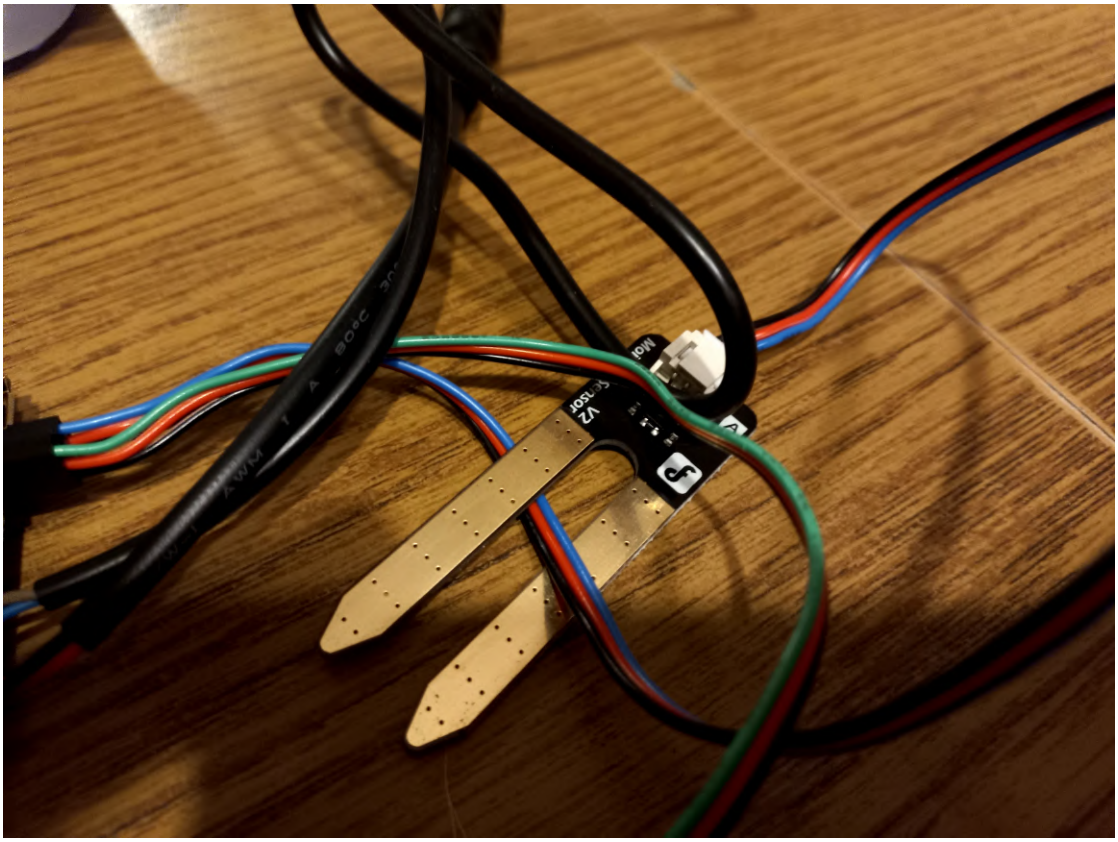
Rysunek 4.3: Podłączenie przewodów na płytce.

Aby podłączyć czujnik temperatury i wilgotności powietrza DHT11 należy wykorzystać jeden z dostępnych w zestawie przewodów, w projekcie został użyty czarno-czerwono-zielony. Pierwszy pin należy podpiąć pod digital'owy pin D9 na płytce, drugi pod VCC, a trzeci pod GND (Rysunek 4.4).



Rysunek 4.4: Podłączenie przewodu pod czujnik temperatury i wilgotności powietrza.

Kolejnym elementem jest czujnik wilgotności gleby i aby podłączyć należy wykorzystać drugi z dostępnych w zestawie przewodów, na rzecz projektu został użyty przewód czarno-czerwono-niebieski. Pierwszy pin w czujniku należy podpiąć do analogowego pinu A2, drugi pod VCC, a trzeci pod GND (Rysunek 4.5).



Rysunek 4.5: Podłączenie przewodu pod czujnik temperatury.

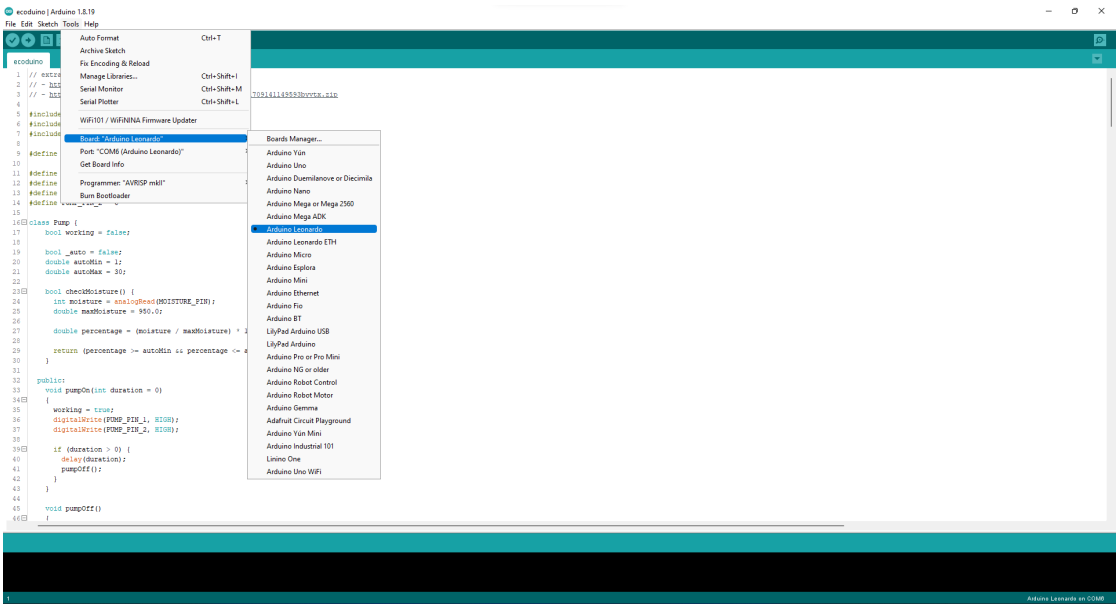
Następnym użytym elementem jest pompa, która jest podłączona do “Soneloid Valve” odpowiednio brązowym przewodem do “+” i niebieskim do “-”. Przewód został przykręcony dołączonym do zestawu śrubokręty krzyżykowym.

Ostatnim elementem jest przewód z gniazdem DC, który jest podłączony do “PWRN” odpowiednio czerwonym przewodem do “+” i czarnym do “-”. Przewód ten, jak powyżej także został przykręcony śrubokrętem.

Na płytkę został wgrany odpowiednio przygotowany wcześniej program, który

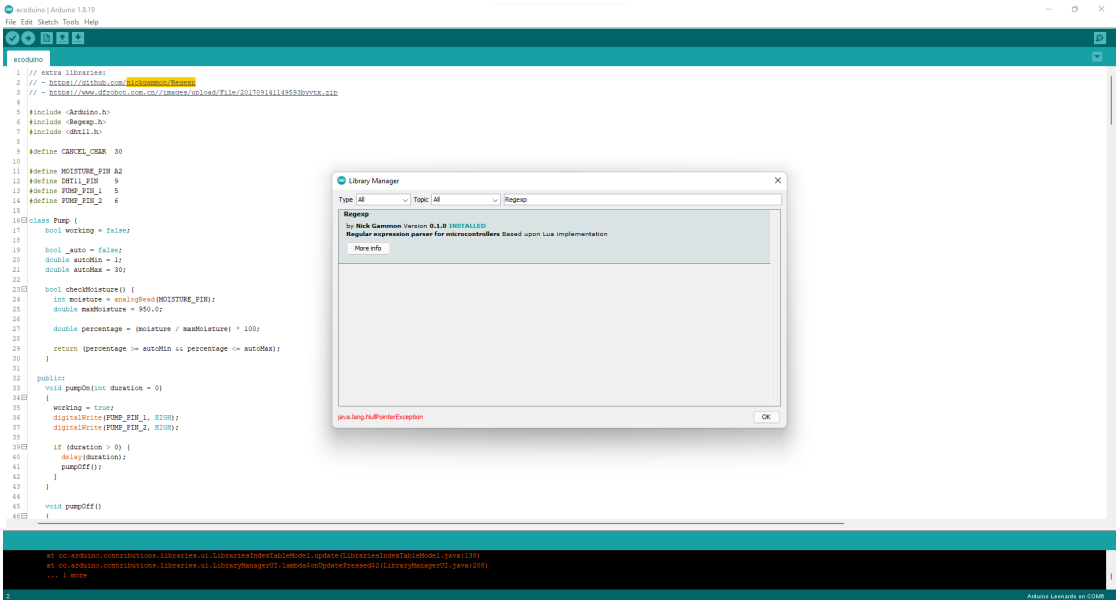
zostaje uruchomiony zaraz po podłączeniu źródła prądu. Aplikacja desktop'owa, po uprzednim zbudowaniu, nie wymaga żadnych dodatkowych kroków podczas uruchomienia.

Aby zainstalować program na płytkę z zestawu EcoDuino należy zainstalować Arduino IDE, otworzyć kod programu "ecoduino.ino", wybrać z paska menu głównego opcję "Tools" →"Board" →"Arduino Leonardo" (Rysunek 4.6) oraz port w "Tools" →"Port" →"Wybrać odpowiedni port" (na przykład "COM6").



Rysunek 4.6: Ustawienie płytki w Arduino IDE.

Następnie należy doinstalować biblioteki “nickgammon/Regexp” i “DHT11”. Aby zainstalować bibliotekę “Regexp” należy wybrać z paska menu głównego “Tools” →“Manage libraries...”, wyszukać frazę “Regexp” i zainstalować paczkę (Rysunek 4.7). Aby zainstalować paczkę “DHT11” należy pobrać ją z linku na początku kodu źródłowego pliku “ecoduino.ino” [4], rozpakować paczkę i przenieść do folderu “C:\Users\<nazwa_uzytkownika>\Documents\Arduino\Libraries”. Po tych akcjach obie biblioteki będą dostępne w środowisku.



Rysunek 4.7: Instalacja biblioteki “Regexp”.

Kolejnym krokiem jest wybranie przycisku “Verify” z górnej belki w celu weryfikacji i kompilacji kodu, a następnie “Upload” w celu posłania skompilowanego programu na płytkę z zestawu EcoDuino.

Aby zbudować aplikację desktop’ową należy zainstalować Flutter’a korzystając z oficjalnej dokumentacji [2]. W pierwszym kroku należy pobrać ukazaną paczkę z najnowszym Flutter SDK i rozpakować ją w wybranym miejscu oraz dodać folder “flutter\bin” do zmiennej środowiskowej “PATH”. Po tej akcji w konsoli PowerShell powinno być dostępne polecenie “flutter”, dzięki czemu możemy już korzystać z wszystkich jego możliwości. Aby dokończyć instalacje należy w konsoli PowerShell

wykonać komendę “flutter doctor”, zainstaluje ona wszystkie zależności dla platformy oraz wyświetli raport statusu instalacji Flutter’a. Aby aplikacja działała na Windows’ie należy dodatkowo zainstalować Visual Studio 2019 oraz podczas instalacji wybrać paczkę “Desktop development with C++”, a następnie w konsoli wykonać komendę “flutter config --enable-windows-desktop”. Po wykonanych akcjach można zweryfikować czy instalacja została wykonana pomyślnie wykonując polecenie “flutter doctor” i oberwawszy brak błędu dla środowiska Windows (Rysunek 4.8).

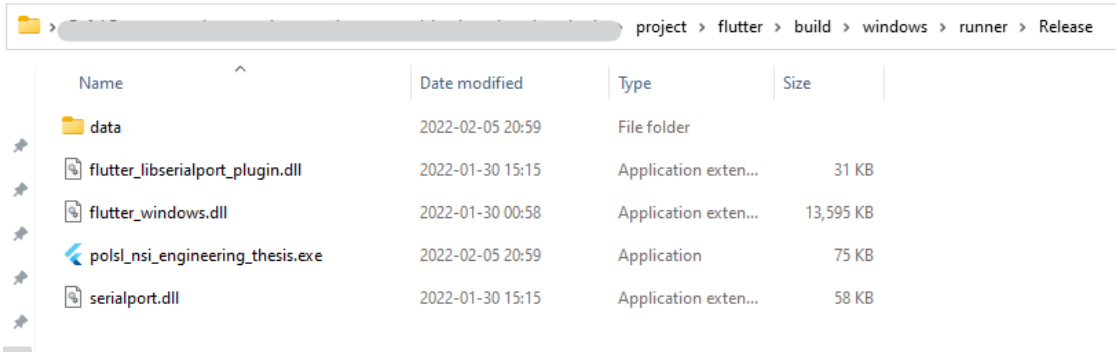
```
PS C:\Users\rafal> flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 2.8.1, on Microsoft Windows [Version 10.0.22000.469], locale en-US)
[✓] Android toolchain - develop for Android devices (Android SDK version 32.1.0-rc1)
[✓] Chrome - develop for the web
[✓] Visual Studio - develop for Windows (Visual Studio Community 2019 16.11.9)
[✓] Android Studio (version 2021.1)
[✓] VS Code, 64-bit edition (version 1.64.0)
[✓] Connected device (3 available)

• No issues found!
PS C:\Users\rafal>
```

Rysunek 4.8: Wykonanie polecenia “flutter doctor”.

Aby zbudować aplikację należy wykonąc polecenie “flutter build windows” (Rysunek 4.9). Dzięki tej komendzie zostanie zbudowana finalna aplikacja i będzie ona dostępna w podkatalogu projektu “build\windows\runner\Release\nazwa_projektu.exe” (Rysunek 4.10).

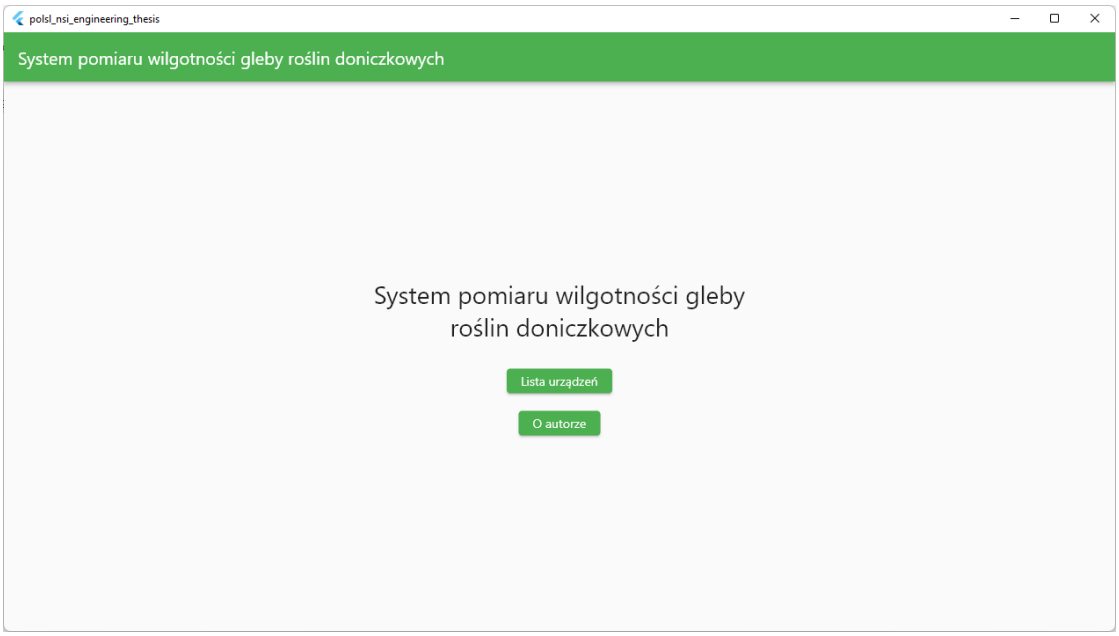
```
PS C:[redacted]project\flutter> flutter build windows

💪 Building with sound null safety 💪

Building Windows application...
PS C:[redacted]\project\flutter>
```

Rysunek 4.9: Wykonanie polecenia “flutter build windows”.



Rysunek 4.10: Przedstawiona finalna lokalizacja aplikacji.

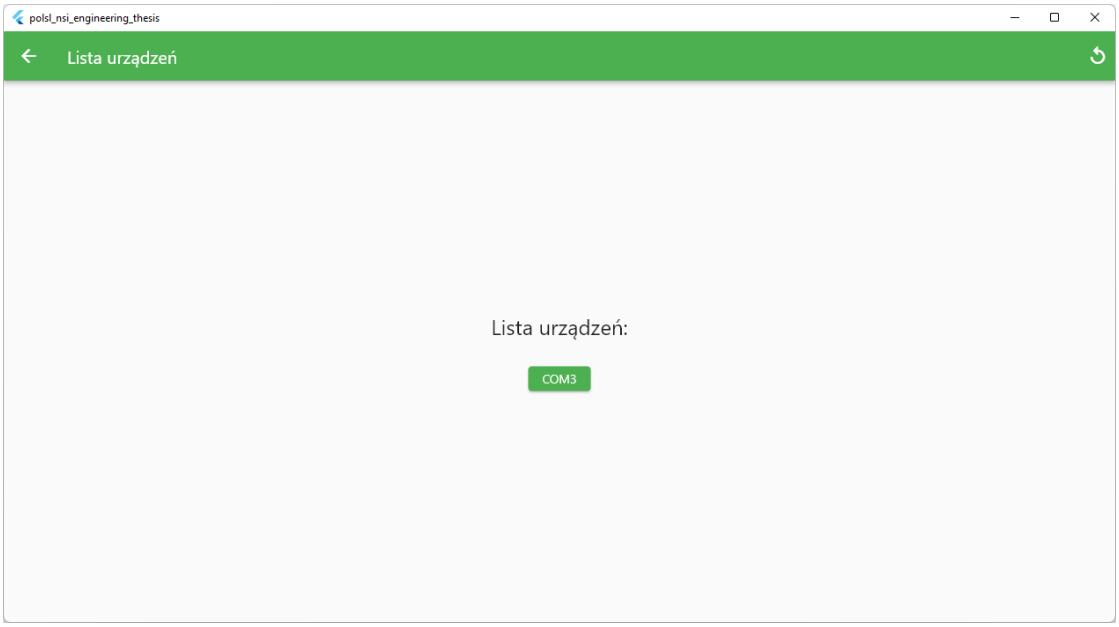
Po uruchomieniu finalnego pliku, ukazuje się ekran startowy aplikacji (Rysunek 4.11).



Rysunek 4.11: Ekran startowy aplikacji.

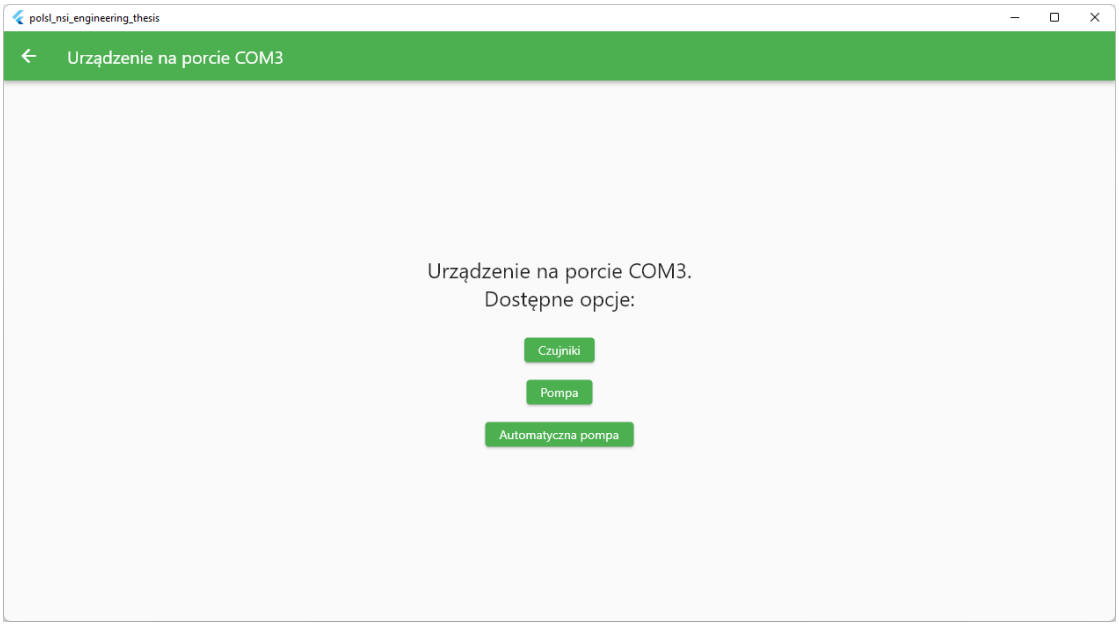
4.5 Sposób aktywacji

Po instalacji programu na płytkę oraz aplikacji desktop'owej system jest gotowy do użytku. Należy podłączyć urządzenie do prądu, uruchomić aplikację na komputerze i podpiąć kabel microUSB do urządzenia. Po wykonaniu tych czynności, w aplikacji na liście urządzeń powinno pojawić się nowe urządzenie, ewentualnie należy odświeżyć widok (Rysunek 4.12).



Rysunek 4.12: Lista dostępnych urządzeń.

Po wybraniu urządzenia dostępna już jest możliwość jego zarządzania (Rysunek 4.13).



Rysunek 4.13: Zarządzanie urządzeniem.

4.6 Sposób obsługi

Do zarządzania urządzeniem dostępne są trzy opcje:

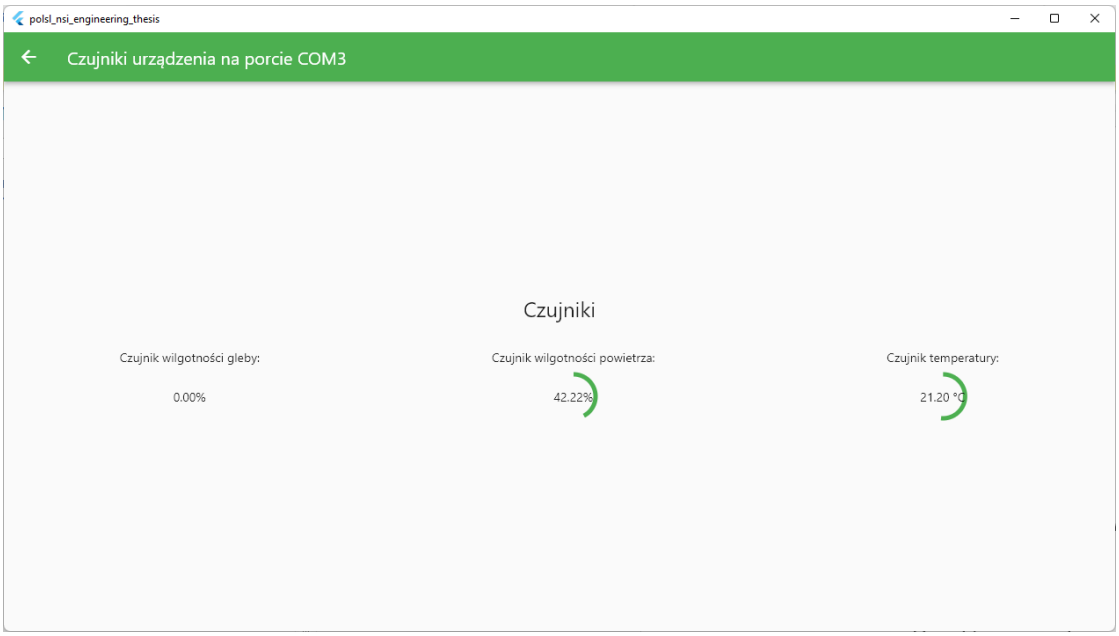
- "Czujniki" - monitorowanie czujników urządzenia,
- "Pompa" - włączanie/wyłączanie pompy,
- "Automatyczna pompa" - włączanie/wyłączanie automatycznego podlewania rośliny, a także ustawienie zakresu wilgotności gleby w jakim pompa ma zostać aktywowana.

W widoku “Czujniki” możliwe jest monitorowanie podłączonych czujników (Rysunek 4.14). Dostępne są 3 okrągłe wskaźniki:

- dla czujnika wilgotności gleby,
- dla czujnika wilgotności powietrza,
- dla czujnika temperatury.

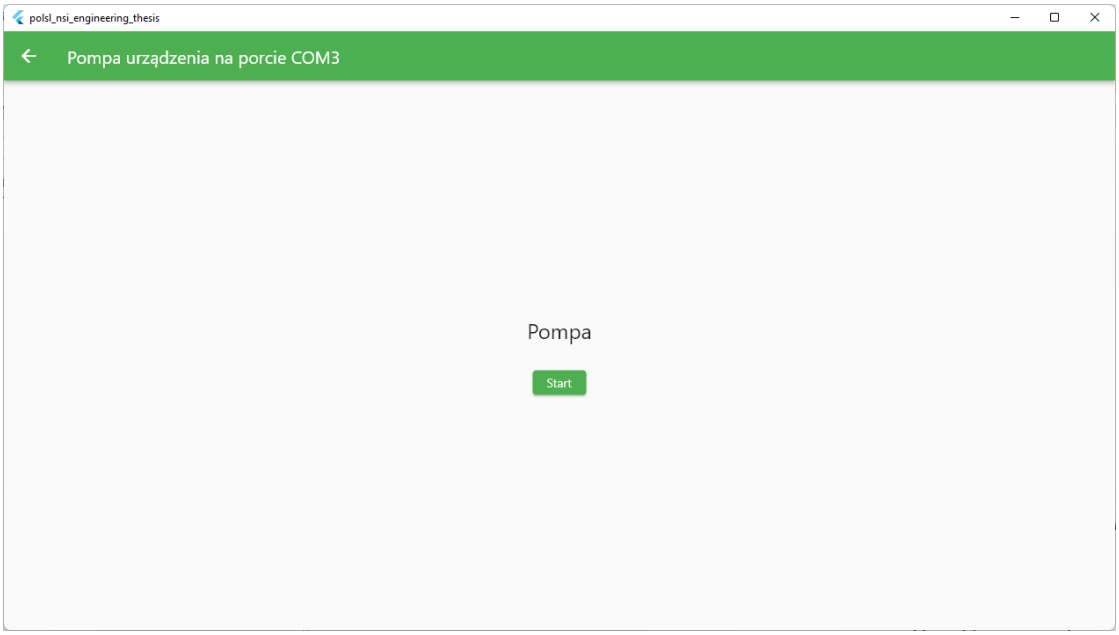
Każdy z nich, poza temperaturą, wskazuje procentowe dane sczytane z czujnika z dokładnością do 2 miejsc po przecinku. W przypadku czujnika temperatury okrągły wskaźnik reprezentuje procentowe dane aktualnej temperatury względem minimalnej i maksymalnej wartości jakie mogą być sczytane używając tego czujnika, a liczba w środku podaje aktualną temperaturę otoczenia w stopniach Celsiusza.

W przypadku niedostępności któregoś z czujników pojawia się tekst “N/A”, a okrągły wskaźnik kręci się w kółko (Domyślne zachowanie “CircularProgressIndicator” w Flutter’ze gdy wartość jest równa 0).



Rysunek 4.14: Widok czujników urządzenia.

Kolejnym widokiem jest widok nazwany “Pompa” i posiada on jeden przycisk, który pozwala na manualne włączenie i wyłączenie pompy (Rysunek 4.15). Dodatkowo po kliknięciu przycisku “Start” zmienia on nazwę na “Stop”, a także zmienia kolor na czerwony, a po ponownym kliknięciu tekst zmienia się z powrotem na “Start”, a kolor na domyślny aplikacji.



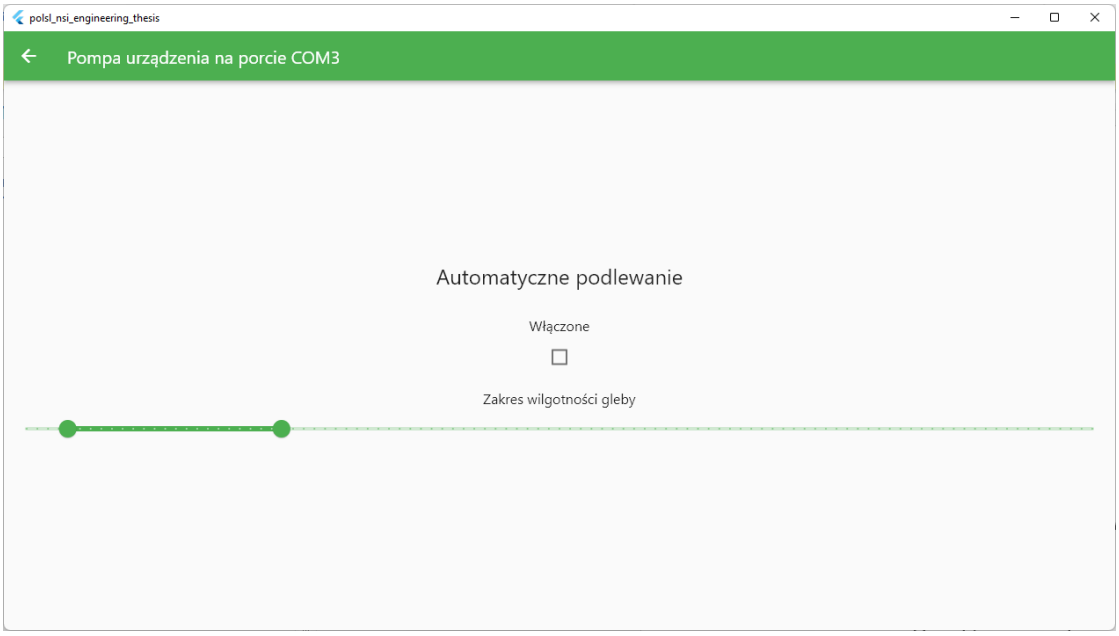
Rysunek 4.15: Widok pompy.

Ostatnim widokiem jest widok nazwany “Automatyczne podlewanie” (Rysunek 4.16) i posiada on dwie opcje:

- włączenie/wyłączenie funkcji automatycznego podlewania,
- ustawienie zakresu wilgotności gleby w jakiej pompa ma zostać włączona.

Cała funkcjonalność zaczyna działać dopiero po włączenia jej używając checkbox’a. Poniżej znajdujący się suwak z zakresem pozwala wyznaczyć procentową, początkową i końcową wartość czujnika wilgotności gleby, po której osiągnięciu powinna uruchomić się pompa. Wartości te są w zakresie 1-100 i suwakiem można

poruszać o 1 jednostkę. Podane ograniczenia zakresu, zaczynające się od 1 wynikają z powodu uniknięcia sytuacji, w której pompa zaczyna działać po wyciągnięciu czujnika wilgotności gleby z rośliny.



Rysunek 4.16: Widok automatycznego podlewania.

4.7 Administracja systemu

Administratorem systemu zostaje końcowy użytkownik korzystający z danego systemu. Został on stworzony w taki sposób, aby użytkownik mógł w prosty i przejrzysty sposób monitorować stan wilgotności gleby i czynników na około rośliny oraz podlewać manualnie i automatycznie roślinę.

4.8 Kwestie bezpieczeństwa

Od kwestii fizycznej projektu należy zwrócić szczególną uwagę na ustawienie urządzenia i zbiornika z wodą, w którym zanurzona jest pompa, aby nie przewrócić

całego zestawu, pomyłkowo zahaczając o na przykład rurkę z pompy, co może spowodować zalanie niżej znajdujących się rzeczy.

Kolejną rzeczą na jaką warto zwrócić uwagę jest monitorowanie poziomu wody w zbiorniku gdzie jest umieszczona pompa. Zbyt mała ilość może spalić mechanizm pompy i spowodować permanentne uszkodzenie pompy.

Od kwestii programowej, całość zarządzania urządzeniem dzieje się poprzez użycie przewodu microUSB, dzięki czemu pozostaje wyeliminowane wykorzystanie urządzenia przez niepowołane osoby z zewnątrz, ponieważ do skorzystania z urządzenia jest potrzebny do niego fizyczny dostęp.

4.9 Przykład działania

Przykładowy scenariusz działania aplikacji wygląda następująco:

Użytkownik otrzymuje wcześniej przygotowany i złożony zestaw oraz archiwum zip z zbudowaną aplikacją desktop'ową. Po rozpakowaniu zestawu użytkownik wkłada czujnik wilgotności do rośliny, przygotowuje źródło wody i wkłada do niego pompę używając dostępnych w pompie przyssawek, oraz podpina urządzenie do źródła zasilania. Następnie rozpakowuje paczkę zip z aplikacją desktop'ową, podłącza komputer do urządzenia kablem microUSB i korzysta z aplikacji, na przykład sprawdzając status czujników i ustawiając automatyczne podlewanie rośliny. Po wykonaniu wszystkich czynności, użytkownik odłącza kabel. Gdy użytkownik chce zakończyć działanie urządzenia, odłącza je od źródła zasilania.

Alternatywnie użytkownik otrzymuje zestaw EcoDuino do samodzielnego złożenia oraz otrzymuje kod programu na płytkę, a także kod aplikacji desktop'owej i instrukcję instalacji i aktywacji. Użytkownik składa urządzenie przy użyciu informacji z punktu 4.4 i 4.5. Po przygotowaniu do użytku zestawu, użytkownik może rozpocząć pracę tak jak zostało to opisane w poprzednim scenariuszu.

Rozdział 5

Specyfikacja wewnętrzna

Poniższy rozdział obejmuje tematykę przedstawienia idei projektu, przedstawienia architektury sytemu jaka została użyta, wykonany został przegląd najważniejszych klas, modułów, komponentów i bibliotek. Dodatkowo zostały opisane szczegóły implementacji ważniejszych algorytmów oraz ważniejszych fragmentów kodu, zastosowane wzorce projektowe i diagramy UML klas.

5.1 Przedstawienie idei

Ideą było stworzenie systemu pozwalającego na szybkie podłączenie zestawu do rośliny oraz zaprogramowanie aplikacji do jej monitorowania i zarządzania z prostym i przyjaznym użytkownikowi interfejsem pozwalający użytkownikowi na najlepszy UX (ang. *User eXperience*).

5.2 Architektura systemu

Architekturą wykorzystaną w systemie jest zmodyfikowana, uproszczona pod projekt wersja protokołu HTTP. Pojedyncze żądanie wygląda jak na kodzie 5.1.

```
1 METODA /routing?parametr1=wartosc&parametr2=wartosc
```

Rysunek 5.1: Schemat pojedynczego żądania.

- METODA - jedna z 2 dostępnych w systemie metod:
  - GET - służy do pobierania danych z urządzenia,
  - POST - służy do zapisu danych na urządzeniu lub wykonaniu na nim jakiejś akcji,
- /routing - ścieżka do akcji jaka ma zostać wykonana, dostępne w systemie ścieżki to:
  - /moisture - działa tylko z metodą GET, zwraca dane z czujnika wilgotności gleby, zwracane dane:
    - value - wartość odczytana z czujnika,
    - min - minimalna wartość czujnika (0),
    - max - maksymalna wartość czujnika (950),
    - ranges - tablica z oznaczeniami poszczególnych zakresów, pojedynczy zakres składa się z 3 elementów: "from" - wartość początku zakresu, "to" - wartość końca zakresu, "title" - tytuł, oznaczenie danego zakresu. Zwracane są 3 elementy z zakresami i oznaczeniami takimi jak prezentuje dokumentacja [5]:
      - 0-300 - "dry soil",
      - 301-700 - "humid soil",
      - 701-950 - "in water",
  - /temperature - działa tylko z metodą GET, zwraca dane z czujnika temperatury, zwracane dane:
    - value - wartość odczytana z czujnika,
    - min - minimalna wartość czujnika (-20),
    - max - maksymalna wartość czujnika (60),
  - /humidity - działa tylko z metodą GET, zwraca dane z czujnika wilgotności powietrza, zwracane dane:
    - value - wartość odczytana z czujnika,
    - min - minimalna wartość czujnika (5),

    - max - maksymalna wartość czujnika (95),
  - /pump - działa tylko z metodą GET, zwraca zapisane dane na temat pompy, zwracane dane:
    - working - aktualny stan pompy (jest włączona/wyłączona),
    - auto - aktualny stan automatycznego podlewania (jest włączona-/wyłączona funkcja),
    - auto_min - minimalna wartość automatycznego podlewania,
    - auto_max - maksymalna wartość automatycznego podlewania,
  - /pump/start - działa tylko z metodą POST, uruchamia pompę,
  - /pump/stop - działa tylko z metodą POST, zatrzymuje pompę,
  - /pump/auto-pump - działa tylko z metodą POST, ustawia automatyczne podlewanie, wymagane parametry:
    - auto - ustawia automatyczne podlewanie, wartość 0 wyłącza, każda inna wartość włącza automatyczne podlewanie,
  - /pump/auto-pump-range - działa tylko z metodą POST, ustawia zakres w jakim automatyczne podlewanie ma działać, wymagane parametry:
    - min - minimalna wartość czujnika wilgotności gleby, po osiągnięciu jakiej ma zostać uruchomiona pompa,
    - max - maksymalna wartość czujnika wilgotności gleby, po osiągnięciu jakiej ma zostać uruchomiona pompa.

Każde żądanie oddzielone jest od siebie znakiem nowej lini (\n). Żądanie może zostać przerwane poprzez wstawienie w dowolnym miejscu znaku "record separator" z numerem 30 w tabli ASCII [9].

Zwrotka jaka jest zwracana po osiągnięciu, któregoś z dostępnych routing'ów jest przygotowana w standardzie JSON i zawiera:

- code - kod błędu, użyte w systemie to 200 (polecenie wykonano prawidłowo) i 500 (wystąpił nieoczekiwany błąd),
- status - wiadomość opisująca status aktualnego żądania,

- data - dane zwracane przez żądanie, jest to część opcjonalna,

W przypadku napotkania błędu system zwraca JSON z kodem 500 i odpowiednio opisanym statusem.

Przykłady są zaprezentowane w kodach 5.2, 5.3, 5.4, 5.5.

```
1 Request:
2 GET /moisture
3
4 Response:
5 {"code":200,"status":"OK","data":{"value":"10","min":"0","max":"950",
    "ranges":[{"from":"0","to":"300","title":"dry soil"},{"from":"301"
    ,"to":"700","title":"humid soil"},{"from":"701","to":"950","title"
    :"in water"}]}}
```

Rysunek 5.2: Przykład 1 żądania

```
1 Request:
2 GET /pump
3
4 Response:
5 {"code":200,"status":"OK","data":{"working":"1","auto":"1","auto_min"
    :"1","auto_max":"30"}}
```

Rysunek 5.3: Przykład 2 żądania

```
1 Request:
2 POST /pump/auto-pump-range?min=10&max=50
3
4 Response:
5 {"code":200,"status":"OK"}
```

Rysunek 5.4: Przykład 3 żądania

```
1 Request:
2 GET /test
3
4 Response:
5 {"code":500,"status":"Invalid route."}
```

Rysunek 5.5: Przykład 4 żądania

Ze względu na ograniczenia sprzętowe, wszystkie żądania kolejkują się i wykonują jedno po drugim, tak jak jest to wykonane w zamyśle kolejki FIFO (ang. *First In First Out*).

5.3 Komponenty, moduły, biblioteki, przegląd ważniejszych klas

Program na płytce EcoDuino składa się z 2 klas “HttpServer” i “Pump”.

Klasa “HttpServer” służy do obsługi pojedynczego rządania. Zawiera ona publiczne metody:

- void readRequest() - metoda służąca do pobrania całego, pojedynczego request’a,
- void processRequest() - metoda służąca do obsłużenia rządania i przygotowująca zwrotkę,
- void sendResponse() - metoda posyłająca response jeśli istnieje.

Pozostałe metody w tej klasie, są to metody typu “protected” i służą one głównie rozbiciu powyższych funkcji na mniejsze fragmenty. Przykładowo znajdują się tam metody takie jak:

- String getMethod() - metoda pobierająca za pomocą regex’a metodę request’u,
- String getRoute() - metoda pobierająca za pomocą regex’a routing request’u,

- void processGet() - metoda obsługująca request'y metody "GET",
- void processPost() - metoda obsługująca request'y metody "POST",
- bool checkDHT() - metoda sprawdzająca dostępność czujnika DHT,
- bool waitForDHT(int maxTimeout) - metoda czekająca na dostępność czujnika DHT z maksymalnym czasem oczekiwania.
- String getQueryString() - metoda zwracająca query z request'a,
- String getNextQueryParam() - metoda ucinająca query o kolejny parametr i zwracająca go.

Drugą klasą na płytce jest klasa "Pump" zajmująca się obsługą pompy wraz z automatyczny podlewaniem roślin. Posiada ona 10 klas publicznych i 1 metodę typu "protected". W skład publicznego interfejsu wchodzą metody:

- void pumpOn(int duration = 0) - metoda włączająca pompę na określony czas w milisekundach lub do czasu wyłączenia jeśli wartość jest równa 0,
- void pumpOff() - metoda wyłączająca pompę,
- void performAuto() - metoda wykonująca automatyczne podlewanie jeśli jest to możliwe,
- bool getWorking(), bool getAuto(), double getAutoMin(), double getAutoMax() - metody zwracające odpowiednio działanie pompy, czy jest ustawione automatyczne podlewanie rośliny, jaka jest minimalna i maksymalna wartość czujnika wilgotności gleby, aby pompa mogła zostać włączona przy automatycznym podlewaniu,
- void setAuto(bool _auto), void setAutoMin(double autoMin), void setAutoMax(double autoMax) - metody ustawiające odpowiednio czy jest ustawione automatyczne podlewanie rośliny, minimalną i maksymalną wartość czujnika wilgotności gleby, aby pompa mogła zostać włączona przy automatycznym podlewaniu.

W sekcji "protected" jest jedna metoda checkMoisture() sprawdzająca czy czujnik wilgotności gleby znajduje się w ramach minimalnej i maksymalnej wartości dla automatycznego podlewania.

Aplikacja desktop'owa napisana jest w Flutte'rze, który przyjmuje ideologię posiadania każdej klasy jako osobnego widget'u, dlatego na każdy widok ekranu zostały napisane osobne widget'y. Dzięki takiemu podejściu jest możliwe w łatwy sposób jest możliwe wyodrębnić każdy widok i uruchomić go z dowolnego momentu życia aplikacji. W niniejszego pracy znalazły się następujące widget'y:

- Application - główny widget aplikacji, zostaje on uruchomiony na samym początku włączenia i ustawia on podstawowe cechy aplikacji, na przykład tytuł, główny kolor i tym podobne,
- HomepageWidget - widget strony głównej aplikacji, posiadający 2 przyciski, jeden prowadzący do listy urządzeń, a drugi do informacji o autorze,
- AboutAuthorWidget - widget zawierający informacje o autorze i promotorze,
- DevicesListWidget - widget wyświetlający listę podłączonych urządzeń, dodatkowo w górnym pasku aplikacji po prawej stronie jest przycisk do odświeżenia listy,
- DeviceWidget - widget wyświetlający opcje dla danego urządzenia, dostępne opcje to "czujniki", "pompa" i "automatyczne podlewanie",
- SensorsWidget - widget wyświetlający dane z 3 czujników takie jak wilgotność gleby, wilgotność i temperatura powietrza. Dane wyświetlane są procentowo, oprócz temperatury, która jest wyświetlana w stopniach Celsiusza. Każda wartość jest otoczona w "CircularProgressIndicator" wyświetlający procentową wartość względem minimalnej i maksymalnej wartości. Jeśli czujnik jest niedostępny, zostaje wyświetlony tekst "N/A" tzn. z angielskiego "Not available",
- PumpWidget - widget pozwalający włączyć i wyłączyć pompę, jeśli pompa jest włączona to przycisk zmienia kolor na czerwony, a gdy pompa jest wyłączona to zmienia się na domyślny,

- AutoPumpWidget - widget pozwalający na ustawienie automatycznego podlewania rośliny. Znajduje się na nim checkbox do włączenia danej funkcjonalności oraz "RangeSlider" do ustawienia minimalnej i maksymalnej wartości, po osiągnięciu której czujnik wilgotności gleby uruchomi pompę.

Oprócz widget'ów zostały także utworzone klasy reprezentujące sensory i pompę na poszczególnych widokach. Dane pobrane z płytki są pobierane jako JSON, a następnie zapisane do obiektów tych klas, dzięki czemu zarządzanie tymi danymi jest łatwiejsze i bezpieczniejsze. Do tych klas należy 1 abstrakcyjna "Sensor" z wspólnymi polami i metodami, oraz dziedziczące po niej klasy takie jak "Humidity", "Moisture", "Temperature" i "Pump".

Ostatnimi plikami w projekcie są 2 pliki w folderze "helper" posiadające pomocnicze klasy i stałe. W pliku "helper/const.dart" zawarta jest stała "CANCEL_CHAR" ukazująca znak zatrzymania request'u, natomiast w pliku "helper/curl.dart" występuje klasa "Curl". Nazwa klasy jest inspirowana Linux'owym narzędziem "curl", a także występującymi funkcjami "curl" w języku PHP. Narzędzia te pozwalały na transportowanie danych w sieci przy użyciu któregoś z protokołów. Stworzona w projekcie klasa "Curl" także służy do transportowania danych między aplikacją, a płytką w zestawie EcoDuino. Posiada ona 3 metody:

- Future<String> createCurl(dynamic data) async - asynchroniczna metoda tworząca osobny wątek, w którym są wykonywane dalsze czynności,

- static Future<void> dataLoader(SendPort sendPort) async - statyczna asynchroniczna metoda wykorzystywana przez metodę "createCurl" podczas tworzenia osobnego wątku. Służy ona do wysyłania rządania do płytki z zestawu EcoDuino i oczekuje na zwrócony wynik, a następnie przekazuje go z powrotem,

- Future sendReceive(SendPort port, data) - metoda służąca jako łącznik pomiędzy metodami "createCurl" i "dataLoader". Wysyła ona dane do utworzonego wątku i zwraca po tym jak otrzyma zwrotkę z "dataLoader".

5.4 Przegląd ważniejszych algorytmów

Jednym z ciekawszych algorytmów zastosowanych w projekcie jest funkcja automatycznego podlewania rośliny. Występuje on w metodzie "performAuto" klasy "Pump" programu na płytkę z zestawu EcoDuino (Kod 5.6). Algorytm w pierwszej kolejności sprawdza w pętli czy funkcja automatycznego podlewania rośliny została włączona oraz czy dane z czujnika wilgotności gleby znajdują się w odpowiednim zakresie w metodzie "checkMoisture", poprzez pobranie wartości z czujnika, obliczenie procentowej wartości oraz porównania jej z zapisanym minimum i maksimum (Kod 5.7). Jeśli procentowa wartość mieści się w zakresie minimum i maksimum to procedura może wykonać następny krok. Po sprawdzeniu wykonane jest uruchomienie pompy na 1 sekundę i ponowne sprawdzenie. Całość wykonywana jest w pętli do momentu, aż któryś z warunków nie zostanie spełniony.

```
1 void performAuto() {
2   while (_auto && checkMoisture()) {
3     pumpOn(1000);
4   }
5 }
```

Rysunek 5.6: Algorytm automatycznego podlewania rośliny.

```
1 bool checkMoisture() {
2   int moisture = analogRead(MOISTURE_PIN);
3   double maxMoisture = 950.0;
4
5   double percentage = (moisture / maxMoisture) * 100;
6
7   return (percentage >= autoMin && percentage <= autoMax);
8 }
```

Rysunek 5.7: Algorytm sprawdzający czy wartość z czujnika wilgotności gleby znajduje się w zakresie wyznaczonym pod automatyczne podlewanie.

Innym ciekawym algorytmem użytym w pracy jest algorytm pobrania następnego parametru z request'a na płytce (Kod 5.8). Jest on realizowany w metodzie

"getNextQueryParam" klasy "HttpServer" i jako pierwszy argument jest przesyłany ciąg znaków zawierający parametry rządania. Podany string w pierwszej kolejności sprawdzany jest pod względem tego czy nie jest pusty, a następnie jeśli wykazuje wartość dodatnią to pobierana jest pozycja znaku "&" oddzielająca kolejne parametry. Jeśli nie występuje ten znak to cały ciąg znaków można uznać jako parametr, jednak w sytuacji jeśli znak "&" wystąpił, to należy pobrać ciąg znaków, aż do "&" i zapisać do tymczasowej zmiennej, która zostanie zwrócona, a pierwotny ciąg skrócić zaczynając od 1 pozycji po wystąpieniu znaku "&".

```
1  String getNextQueryParam(String &queryParams) {
2    String result;
3
4    if (queryParams.length()) {
5      int ampersandPosition = queryParams.indexOf('&');
6      if (ampersandPosition > 0) {
7        result = queryParams.substring(0, ampersandPosition);
8        queryParams = queryParams.substring(ampersandPosition + 1);
9      } else {
10       result = queryParams;
11       queryParams = "";
12     }
13   }
14
15   return result;
16 }
```

Rysunek 5.8: Algorytm pobrania kolejnego parametru z request'a.

Następnym ciekawym algorytmem jest metoda "_setTimer" w "SensorsWidget" w aplikacji desktop'owej (Kod 5.9, 5.10, 5.11). Tworzy ona obiekt klasy "Timer" uruchamiający co 2 sekundy przekazaną jako parametr funkcję. Funkcja ta uruchamia 3 wykonywane po sobie request'y do płytki, w celu uzyskania danych z poszczególnych czujników. Po wykonaniu pojedynczego request'a w pierwszej kolejności zostają zapisane dane, a potem zostaje uruchomiony kolejny request. Podczas zapisu danych sczytanych z response'a, zostają one zdekodowane z ciągu znaków do JSON'a, następnie jest sprawdzony kod błędu. Jeśli kod błędu jest równy 200 to dane zostają zapisane do obiektu odpowiedniej klasy dziedziczącej po klasie "Sensor", a jeśli kod błędu jest różny od 200, to czujnik na widoku zostaje wyłączony.

```
1 void _setTimer() {
2   var callback = (Timer timer) {
3     // ...
4   };
5
6   _timer = Timer.periodic(new Duration(seconds: 2), callback);
7
8   callback(_timer);
9 }
```

Rysunek 5.9: Ustawienie obiektu klasy "Timer" na 2 sekundy.

```
var callback = (Timer timer) {
  var temperatureOnValue = (response) {
    // ...
  };

  var humidityOnValue = (response) {
    // ...
  };

  var moistureOnValue = (response) {
    // ...
  };

  _curl.createCurl({
    "request": "GET /moisture\n",
    "device": widget.device
  }).then((response) {
    moistureOnValue(response);

    _curl.createCurl({
      "request": "GET /humidity\n",
      "device": widget.device
    }).then((response) {
      humidityOnValue(response);

      _curl.createCurl({
        "request": "GET /temperature\n",
        "device": widget.device
      }).then(temperatureOnValue);
    });
  });
};
```

Rysunek 5.10: Zawartość funkcji “callback” przekazanej jako argument dla obiektu “Timer”.

```
var moistureOnValue = (response) {
  var json = jsonDecode(response);
  if (json['code'] == 200) {
    setState(() {
      _moisture.aviable = true;
      _moisture.value = double.parse(json['data']['value']);
      _moisture.min = double.parse(json['data']['min']);
      _moisture.max = double.parse(json['data']['max']);

      _moisture.ranges.clear();
      for (var item in json['data']['ranges']) {
        Map<String, String> map = new Map<String, String>();
        item.forEach((final String key, final value) {
          map.putIfAbsent(key, () => value);
        });
        _moisture.ranges.add(map);
      }
    });
  } else {
    setState(() {
      _moisture.aviable = false;
    });
  }
};
```

Rysunek 5.11: Zawartość przykładowej funkcji obsługującej response.

5.5 Szczegóły implementacji ważniejszych fragmentów

Jednym z najważniejszych algorytmów w projekcie był algorytm z jakim przebiegała komunikacja między płytką, a aplikacją (Kod 5.12). Na płytce została uruchomiona pętla sprawdzająca dostępność podłączonego komputera i obsługa poszczególnych request'ów.

```
1 void loop() {
2   pump.performAuto();
3   if (Serial.available()) {
4     HttpServer server;
5     server.readRequest();
6     server.processRequest();
7     server.sendResponse();
8   }
9 }
```

Rysunek 5.12: Główna pętla programu na płytce.

Pojedynczy request w pierwszej kolejności jest pobierany, aż do momentu wystąpienia nowej lini lub wcześniej ustalonego znaku zakończenia (Kod 5.13).

```
1 void readRequest() {
2   this->request = "";
3
4   char c = NULL;
5   do {
6     c = Serial.read();
7
8     if (c == '\n') {
9       break;
10    }
11
12    if (c == CANCEL_CHAR) {
13      this->request = "";
14 ␣␣␣␣␣␣this->response␣=␣'\n';
15 ␣␣␣␣␣␣break;
16 ␣␣␣␣}
17
18 ␣␣␣␣this->request␣+=␣c;
19 ␣␣}␣while␣(true);
20 }
21 ␣␣␣
```

Rysunek 5.13: Wczytywanie request'a.

Następna jest metoda obsługująca dane rządania. Wymagane jest, żeby rządania nie było puste, zawierało prawidłową metodę i routing (Kod 5.14).

```
1  void processRequest() {
2    if (!this->request.length()) {
3      return;
4    }
5
6    String method = this->getMethod();
7    String route = this->getRoute();
8
9    method.toUpperCase();
10
11   if (!this->checkMehtod(method)) {
12     this->response = String("{\"code\":500,\"status\":\"Check method
         failure.\"}");
13     return;
14   }
15
16   if (!this->checkRoute(route)) {
17     this->response = String("{\"code\":500,\"status\":\"Check route
         failure.\"}");
18     return;
19   }
20
21   if (method == "GET") {
22     return this->processGet(route);
23   } else if (method == "POST") {
24     return this->processPost(route);
25   }
26 }
```

Rysunek 5.14: Obsługa request'a.

Ostatnią czynnością jest wysłanie, jeśli istnieje, wcześniej przygotowanego response'a, na podstawie wyników dla danego request'a (Kod 5.15).

```
1 void sendResponse() {
2   if (this->response.length()) {
3     Serial.println(response);
4   }
5 }
```

Rysunek 5.15: Wysłanie response'a.

Od strony aplikacji w Flutter'ze przygotowana jest klasa, która po uruchomieniu metody "createCurl" tworzy osobny wątek do obsługi komunikacji między płytką a aplikacją w celu uniknięcia zatrzymania całego programu podczas odczytywania danych z płytki (Kod 5.16). W pierwszej kolejności jest utworzony obiekt klasy "ReceivePort" oraz nowy wątek. Następnie "Isolate" jako pierwszy obiekt zwraca swój obiekt klasy "SendPort" służący do przekazywania danych. Kolejnym krokiem jest wysłanie danych do wątku przez metodę "sendReceive" (Kod 5.17) i poczekanie wykonanie operacji w metodzie "dataLoader" (Kod 5.18) i zwrócenie wyniku.

```
1  Future<String> createCurl(dynamic data) async {
2    ReceivePort receivePort = ReceivePort();
3    await Isolate.spawn(dataLoader, receivePort.sendPort);
4
5    SendPort sendPort = await receivePort.first;
6
7    String response = await sendReceive(sendPort, data);
8
9    return response;
10 }
```

Rysunek 5.16: Metoda createCurl.

```
1 Future sendReceive(SendPort port, data) {
2   ReceivePort response = ReceivePort();
3   port.send([response.sendPort, data['request'], data['device']]);
4   return response.first;
5 }
```

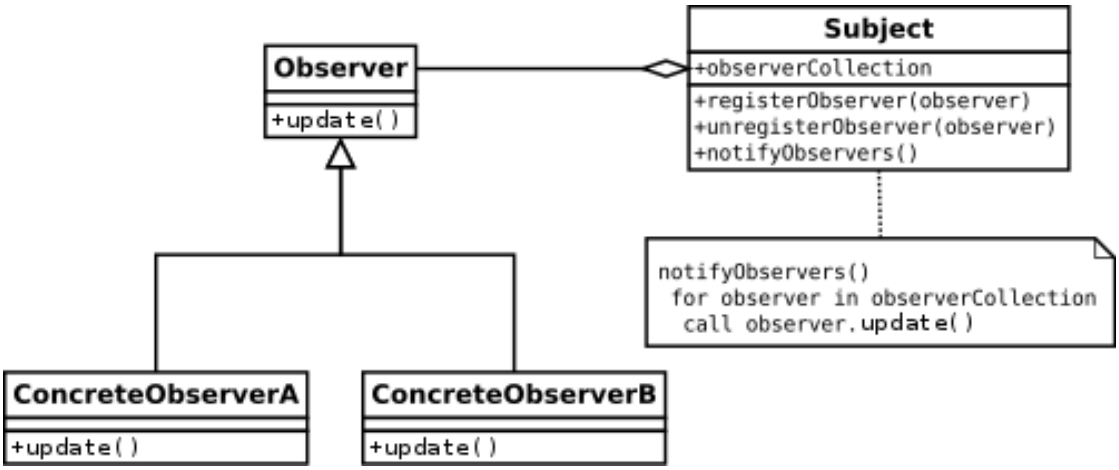
Rysunek 5.17: Metoda sendReceive.

```
1  static Future<void> dataLoader(SendPort sendPort) async {
2    ReceivePort port = ReceivePort();
3
4    sendPort.send(port.sendPort);
5
6    await for (var data in port) {
7      SendPort replyTo = data[0];
8      String request = data[1];
9      String device = data[2];
10     SerialPort serialPort = new SerialPort(device);
11
12     while (serialPort.isOpen) {}
13     serialPort.open(mode: SerialPortMode.readWrite);
14
15     String response = "";
16
17     if (!request.endsWith('\n')) {
18       request += '\n';
19     }
20
21     List<int> list = request.codeUnits;
22     Uint8List bytes = Uint8List.fromList(list);
23
24     serialPort.write(bytes);
25
26     Uint8List l;
27     var c;
28     do {
29       l = serialPort.read(1);
30       if (l.isNotEmpty) {
31         c = String.fromCharCode(l[0]);
32
33         if (c == '\n') {
34           break;
35         }
36
37         response += c;
38       }
39     } while (true);
40
41     serialPort.close();
42
43     replyTo.send(response);
44   }
45 }
```

Rysunek 5.18: Metoda dataLoader.

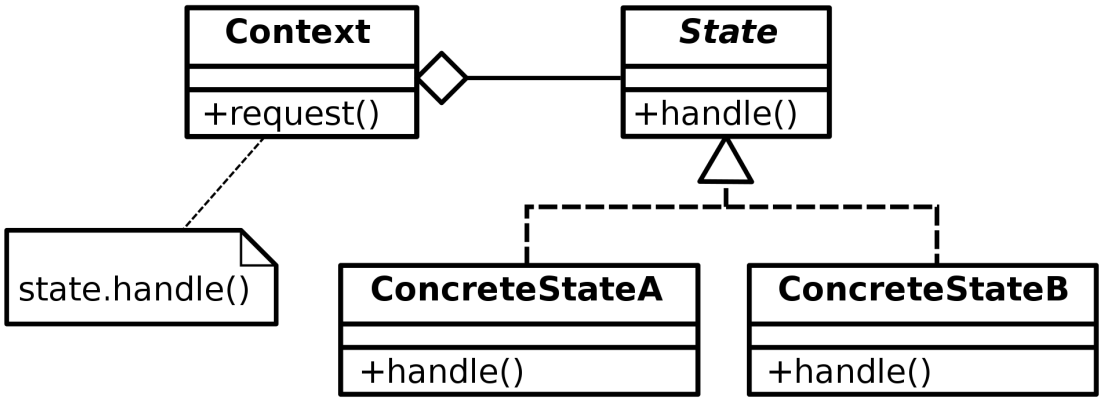
5.6 Zastosowane wzorce projektowe

Flutter w swoim zestawie narzędzi udostępnia wykorzystanie wzorca "Obserwator" (Rysunek 5.19), który pozwala na zmianę wartości obiektu, przy jednoczesnej aktualizacji miejsc w których został on wykorzystany, poprzez powiadomienie ich o zmianie. W Flutter'ze służy do tego metoda "setState(callback)", która przyjmuje jako argument funkcję, która zostaje automatycznie uruchomiona. W przekazanej funkcji zostają nadane wartości wszystkim polom, które potrzebują zostać zaktualizowane.



Rysunek 5.19: Diagram UML wzorca "Obserwator" [6]

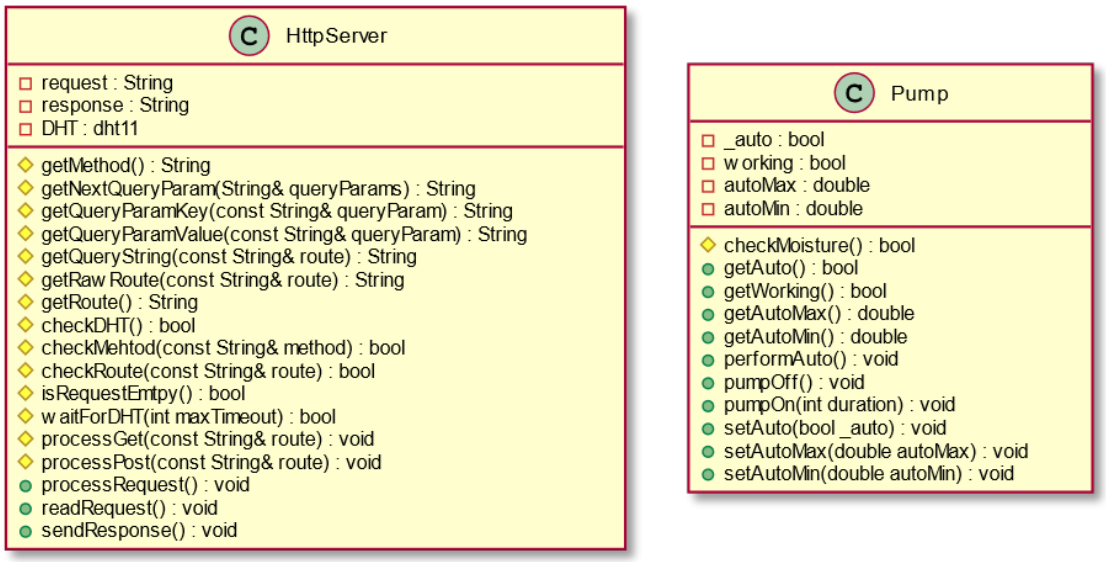
W projekcie został użyty także wzorzec “Stan” (Rysunek 5.20). Pozwala on na ustawienie jakiegoś stanu aplikacji i odmiennych zachowań względem tego jak jest ustawiony. Przykładem w projekcie jest automatyczne podlewanie na płytce EcoDuino, które włączą i wyłącza daną funkcjonalność, w zależności od stanu zmiennej. Innym przykładem jest przycisk “Start”/“Stop” pompy w aplikacji desktop’owej, który w zależności od stanu, zmienia tekst i kolor.



Rysunek 5.20: Diagram klas UML programu na płytkę EcoDuino.

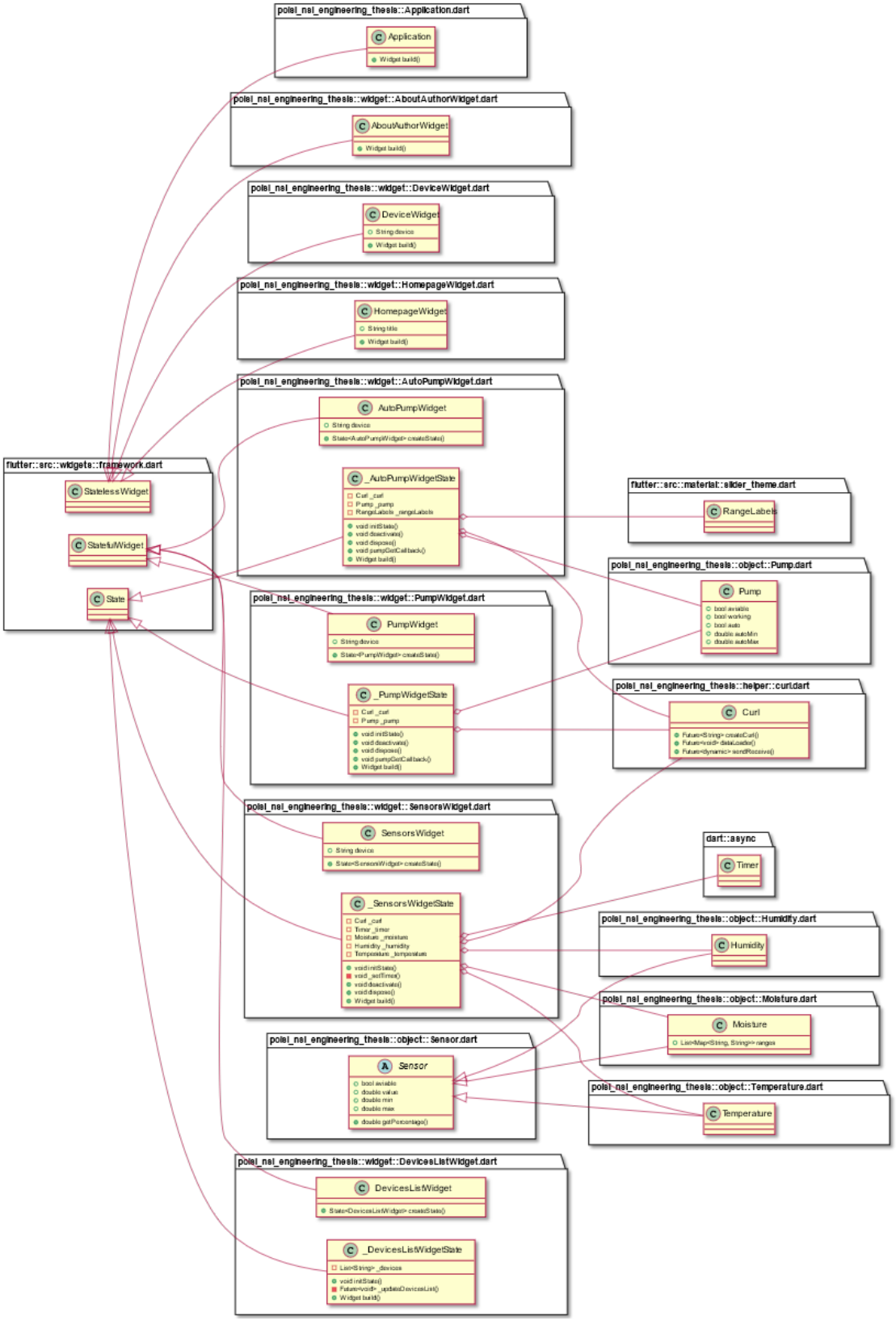
5.7 Diagramy UML

Na rysunku 5.21 ukazany jest diagram klas programu na płytkę z zestawu EcoDuino.



Rysunek 5.21: Diagram UML wzorca “Stan” [7]

Natomiast na rysunku 5.22 znajduje się diagram klas aplikacji desktop’owej.



Rysunek 5.22: Diagram klas UML aplikacji desktop'owej.

Rozdział 6

Weryfikacja i walidacja

W niniejszym rozdziale zostaną przedstawione sposoby weryfikacji i walidacji działania oprogramowania, przypadki testowe, zakres testowania, znalezione błędy oraz sposób ich wyeliminowania, a także wynik jednego z finalnych testów.

6.1 Sposób testowania w ramach pracy

Testowanie aplikacji przebiegało poszczególnymi etapami podczas procesu programowania.

Program na płytce z zestawu EcoDuino został przetestowany manualnie przy użyciu Serial Port Monitor'a.

W aplikacji desktop'owej do testowania zostały użyte testy manualne podczas kolejny części pisania aplikacji, ale także zostało napisanych kilka testów jednostkowych.

6.2 Organizacja eksperymentów

Do testowania programu na płytce z zestawu EcoDuino użyty został Serial Port Monitor, w którym wpisywane były poszczególne rządania i sprawdzane odpowiadające im zwrotki. Jeśli wszystko pasowało, dane z czujników były sczytane pomyślnie, albo pompa zostawała uruchamiana i zatrzymywana, test przechodził. W przeciwnym wypadku, gdy wystąpił błąd i coś nie działało, problem zostawał

sprawdzany i naprawiany.

Podczas pisania aplikacji desktop'owej w Flutter'ze poszczególne części programu były testowane na bieżąco. Wykorzystywane w tym celu zostały testy manualne, które były najprostsze do wykonania oraz pozwalały na przegląd działania wszystkich funkcji w sytuacjach takich jakie może napotkać końcowy użytkownik. Oprócz testów manualnych zostały przygotowane testy jednostkowe sprawdzające poprawność wyświetlania kilku przykładowych widget'ów.

Testy manualne zostały przeprowadzone w dwóch środowiskach:

- deweloperskim - podczas tworzenia oprogramowania, użyte do testów zostało pudełko z wodą, do której została włożona pompa, a czujniki zostały przetestowane odpowiednio nawilżając je wodą z pudełka, czy też dmuchając w przypadku czujnika wilgotności powietrza i temperatury,

- produkcyjnym - finalne testy przy użyciu prawdziwej rośliny i pudełka z wodą.

6.3 Przypadki testowe, zakres testowania

Najważniejszymi zakresami jakie zostały przetestowane były odpowiednio system wykonywania request'ów i response'ów na płytce EcoDuino, poszczególne rządania, działanie czujników, pompy i funkcji automatycznego podlewania, a także całości programu działającego na płytce.

Aplikacja została przetestowana pod kątem wyświetlania i działania każdego z dostępnych widget'ów. Dodatkową uwagę zwrócono na widget z danymi sczytywanymi z czujników urządzenia. Wszystkie rządania musiały zostać wykonywane asynchronicznie w taki sposób, aby użytkownik nie doświadczył zawieszenia aplikacji spowodowanych oczekiwaniem na odpowiedź z urządzenia.

Inną częścią aplikacji, która musiała zostać dokładniej przetestowana była funkcja automatycznego podlewania rośliny. Podczas powrotu do poprzedniego widget'u i ponownego wejścia na widget obsługujący automatyczne podlewanie, dane powinny zostać ponownie pobrane, a nie polegać na tych co były zapisane przy poprzednim wejściu.

6.4 Wykryte i usunięte błędy

Podczas testowania zostało wykrytych parę błędów, które zostały następnie naprawione. Zostaną one przedstawione poniżej.

Największym problemem jaki został wykryty był spadek wydajności aplikacji na widget'cie z czujnikami.

Na początku występował problem z strumieniem do odczytu danych z urządzenia, który był tworzony jako nowy obiekt przy każdym request'cie. Przez to, że obiekt ten nasłuchiwał na zmiany, nie zostawał on usunięty z pamięci, dlatego po kilku wywołaniach funkcji, zostało utworzonych zbyt duża ilość takich obiektów i osiągano limit dostępnych zasobów. Problem ten został naprawiony poprzez zamianę obiektu stream na funkcję sczytującą po 1 znaku, aż do napotkania znaku kończącego response.

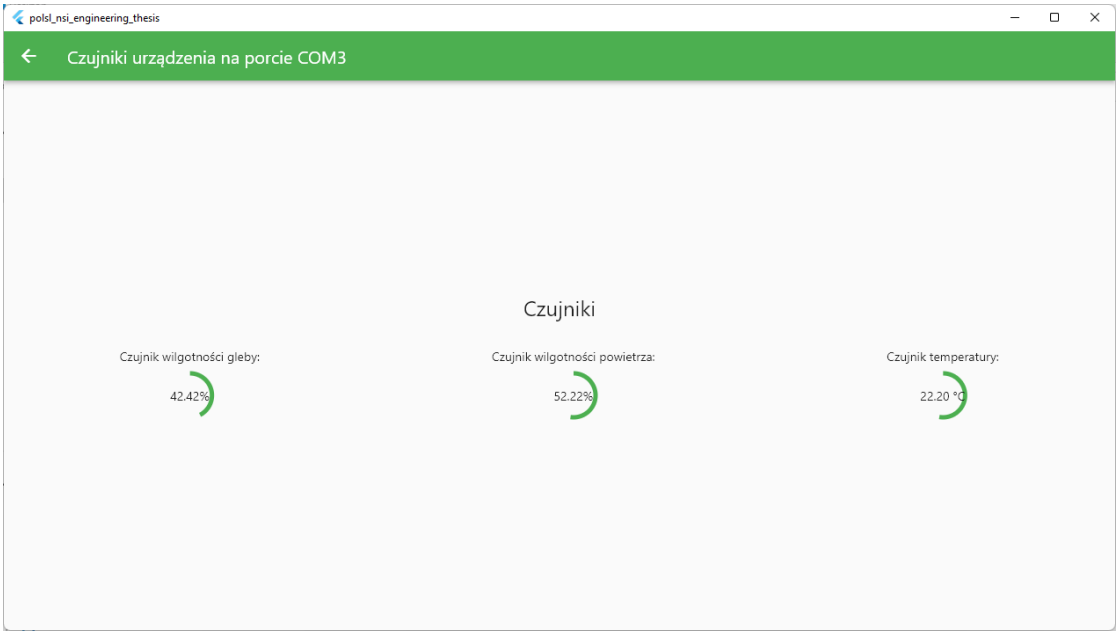
Niestety nie naprawiło to w pełni problemu z wydajnością aplikacji na tym widget'cie. Każde rządania mimo tego, że zostało uruchomione asynchronicznie, wykonywało się w obrębie tego samego wątku, czego konsekwencją było wstrzymanie programu przy każdym odpytaniu. Rozwiązaniem było przeniesienie wywołań do osobnych, wyizolowanych wątków, dzięki czemu, problem z zatrzymywaniem się programu po wykonywaniu rządania został wyeliminowany.

Innym błędem jaki został zauważony podczas testów było ciągłe odpytywanie płytki o dane po wyjściu z widoku z czujnikami. Problem został zauważony podczas wyświetlania żądań i odpowiedzi używając polecenia "print()". Mimo tego, że został opuszczony widok z czujnikami, request'y nadal były wykonywane. Powodem takiego zachowania był ciągłe, nieprzerwane działanie funkcji "periodic()" klasy "Timer". Rozwiązaniem tego problemu było zakończenie działania Timer'a podczas opuszczania widget'u przy użyciu metody "cancel()" klasy "Timer".

6.5 Opcjonalnie wyniki badań eksperymentalnych

Jeden z finalnych testów został wykonany na prawdziwej roślinie Zamioculcas zamiolistnym (Rysunek 6.1). Dane na temat wilgotności gleby jakie zostały sczytane to około 42% (Rysunek 6.2), co nie należało do najlepszych, ale też najgorszych wyników.

Rysunek 6.1: Test na prawdziwej roślinie.



Rysunek 6.2: Wyniki testu na prawdziwej roślinie.

Rozdział 7

Podsumowanie i wnioski

Projekt niniejszej pracy, który zakładał stworzenie systemu do monitorowania wilgotności gleby roślin doniczkowych, monitorowania temperatury i wilgotności otoczenia, nawadniania gleby oraz ustawienia automatycznego nawadniania w zależności od wilgotności gleby został zakończony z powodzeniem. Temat został odpowiednio przeanalizowany, zostały także wybrane właściwe do realizacji narzędzia i technologie, tak aby w finalnym efekcie został stworzony poprawnie działający system.

Podczas realizacji tworzenia systemu, zostało napotkanych kilka problemów, które zostały rozwiązane w czasie programowania. Największym problemem był wspomniany w podrozdziale 6.4 problem związany z wydajnością aplikacji desktop'owej na widget'cie z czujnikami, który powodował trudności, a nawet nie możliwość korzystania z aplikacji. Innym problemem, także opisanym w podrozdziałem 6.4 był niewidoczny dla użytkownika problem z brakiem zatrzymania Timer'a po wyjściu z widget'u z czujnikami, co powodowało niekończące się odpytywanie urządzenia o nowe dane, nawet jeśli nie było się na danym widget'cie.

Projekt ten posiada także potencjał na rozbudowę, poprzez użycie innych środków komunikacji takich jak Bluetooth lub WiFi oraz odpowiedniego ich zabezpieczenia. Rozbudowa ta pozwoliłaby końcowemu użytkownikowi na wygodniejszy sposób korzystania z urządzenia w niedużej odległości, bez konieczności podłączania się pod urządzenie.

Dana praca została wykonana w swoich założeniach i działa prawidłowo. Po-

zwoliła na wykonanie ciekawego projektu, który może zostać użyty, przez każdego człowieka potrzebującego systemu pomagającemu w opiece nad rośliną.

Bibliografia

[1] EcoDuino_-_An_Auto_Plant_Kit_SKU___KIT0003-DFRobot. https://wiki.dfrobot.com/ecoduino_-_an_auto_plant_kit_sku___kit0003. dostęp 20.01.2022.

[2] Windows install | Flutter. https://docs.flutter.dev/get-started/install/windows. dostęp 11.02.2022.

[3] Windows install | Flutter System requirements. https://docs.flutter.dev/get-started/install/windows#system-requirements. dostęp 11.02.2022.

[4] DHT 11 Library. https://www.dfrobot.com.cn//images/upload/file/201709141149593byvtx.zip. dostęp 12.01.2022.

[5] Moisture_Sensor___SKU_SEN0114_-DFRobot. https://wiki.dfrobot.com/moisture_sensor___sku_sen0114_. dostęp 20.01.2022.

[6] Observer pattern Wikipedia. https://en.wikipedia.org/wiki/observer_pattern. dostęp 06.02.2022.

[7] State pattern Wikipedia. https://en.wikipedia.org/wiki/state_pattern. dostęp 13.11.2021.

[8] Portel Schematics. https://image.dfrobot.com/image/data/kit0003/new/freedostęp 20.01.2022.

[9] ASCII Table. https://upload.wikimedia.org/wikipedia/commons/1/1b/ascii-table-wide.svg. dostęp 14.02.2022.

Dodatki

Spis skrótów i symboli

HTTP	Hypertext Transfer Protocol
REST	Representational state transfer
API	Application Programming Interface
JSON	JavaScript Object Notation
UX	User eXperience
FIFO	First In First Out
ASCII	American Standard Code for Information Interchange

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,
- film pokazujący działanie opracowanego oprogramowania oraz zaprojektowanego i wykonanego urządzenia,
- prezentacja,

Spis rysunków