

UNIWERSYTET GDAŃSKI
Wydział Matematyki, Fizyki i Informatyki

Maciej Marzec

nr albumu: 231056

Rafał Kulaszewicz

nr albumu: 219083

Ewelina Mazur

nr albumu: 235359

Inteligentny Dom dla każdego

Praca licencjacka na kierunku:

INFORMATYKA

Promotor:

dr Włodzimierz Bzyl

Gdańsk 2018

Streszczenie

W projekcie „Inteligentny Dom” wykorzystaliśmy układy scalone ESP8266¹. Zostały one wmontowane w listwę zasilającą, a dzięki możliwości zaprogramowania ich w języku C oraz GPIO², umożliwiają otwarcie lub zamknięcie przepływu prądu. Układy poprzez moduł Wi-Fi łączą się z serwerem, do którego wysyłają informacje o aktualnym stanie urządzenia oraz odbierają sygnały zarządzające. Użytkownik może włączać lub wyłączać przepływ prądu korzystając z aplikacji mobilnej bądź strony internetowej. W obu przypadkach posłużyliśmy się językiem Python, lecz używając różnych bibliotek oraz frameworków: Twisted-Kivy na system Android, Flask dla strony internetowej.

Program wymaga od użytkownika rejestracji oraz każdorazowego logowania. Aby odizolować się od niechcianego działania osób trzecich, autoryzacja odbywa się po stronie serwera, poprzez rozszerzenie Flask – SQLAlchemy³. Po zalogowaniu, przed pierwszym zastosowaniem urządzenia, należy zarejestrować układ ESP podając jego adres IP. Komunikację zaimplementowaliśmy przez urllib, z kolei kontakt z modemem 3G jest możliwy poprzez serial port.

Schemat działania aplikacji został przedstawiony na diagramie 1.9.

Słowa kluczowe

Python3, Python2, Twisted, Kivy, Flask, SQLAlchemy, smart home, mobile, web, układ scalony

¹<http://en.wikipedia.org/wiki/ESP8266>

²wyjście – wejście ogólnego przeznaczenia

³<http://flask-sqlalchemy.pocoo.org/2.3/>

Spis treści

Wprowadzenie	6
1. Nasza wizja Inteligentnego Domu	7
2. Mobile app	14
2.1. Kivy – czym jest?	14
2.2. Dlaczego został wybrany?	14
2.3. Kivy ScreenManager	15
2.3.1. Komunikacja widoku z funkcją	15
2.3.2. Przykładowy screen	16
2.4. Layout w Kivy	17
2.4.1. BoxLayout	17
2.4.2. GridLayout	19
2.5. Twisted	20
2.5.1. Kivy z Twisted	21
3. Web app	23
3.1. Python – Flask – co to takiego?	23
3.2. Zastosowanie w aplikacji	23
3.2.1. Flask – SQLAlchemy	24
3.2.2. Flask – LoginManager	26
3.3. Autoryzacja SMS	28
3.4. Tworzenie stron internetowych	29
4. Hardware	33
4.1. ESP8266	33
4.2. Raspberry pi 3 – jako serwer aplikacji	34
4.3. Komunikacja	34
4.4. Bezpieczeństwo w projektach ”Smart Home”	37

<i>Spis treści</i>	5
--------------------	---

Zakończenie	39
------------------------------	----

Bibliografia	40
-------------------------------	----

Spis rysunków	42
--------------------------------	----

Oświadczenie	43
-------------------------------	----

Wprowadzenie

Kiedys pomysł stworzenia instalacji oraz systemu umożliwiającego sterowanie urządzeniami w budynku był mało realny. Mogliśmy jedynie sobie wyobrażać o ile nasze życie mogłoby być wygodniejsze. Obecnie, coraz częściej możemy usłyszeć o projektach kreowanych od samych podstaw pod „futurystyczne” obiekty.¹ Założeniami zastosowania nowoczesnej technologii jest głównie poprawa naszego komfortu i bezpieczeństwa, które są celem naszego projektu. Wyobraźmy sobie sytuację, kiedy wracamy zmęczeni po wyczerpującym dniu w pracy. Za pomocą naszego pilota – w tym wypadku smart-phone’a – przed wejściem do mieszkania włączymy oświetlenie, telewizję lub muzykę i usiadzimy od razu wygodnie w fotelu. Komercyjne rozwiązania takie jak DEIMIC² czy też Samsung Smart Home³ posiadają ogrom możliwości. Przykładowo pozwala ustawić konkretną temperaturę w wyznaczonej strefie obiektu, uzbrajać alarm, sterować klimatyzacją oraz wiele innych. Nasz projekt zespołowy skupia się na dwóch płaszczyznach. Stworzyliśmy aplikację mobilną oraz webową, aby użytkownik nie był ograniczony jedynie do telefonu komórkowego. Zaimplementowaliśmy podstawowe funkcje wyróżniające ten typ aplikacji, które zostaną opisane i poparte przykładami w dalszej części pracy.

¹<https://www.mgprojekt.com.pl/blog/inteligentny-dom/dn.2018-03-22>

²<http://deimic.pl/system-deimic/aplikacja-sterowania-domem>

³<https://play.google.com/store/apps/details?id=com.samsung.smarthome>

ROZDZIAŁ 1

Nasza wizja Inteligentnego Domu

Komercyjne rozwiązania wyglądają atrakcyjnie, ale ich cena i koszt instalacji, bywają zbyt wysokie. Ponadto im więcej sprzętów i czujników mamy do dyspozycji, tym bardziej zawiła staje się obsługa programu. Dlatego opracowaliśmy własne rozwiązanie, którego prostotę osiągnęliśmy implementując jedną funkcję, czyli włączanie i wyłączanie urządzeń. Aby nasz dom stał się bardziej „smart” nie potrzebujemy nowych sprzętów, ani dodatkowej instalacji. Należące do IoT układy scalone wmontowaliśmy w listwę zasilającą i poprzez połączenie z aplikacją możemy zatrzymać przepływ prądu. Tak naprawdę nie sterujemy wcale samym urządzeniem, a jedynie decydujemy o czasie dostarczania mu zasilania. Nie jest to doskonałe rozwiązanie, lecz jest tańsze i mniej inwazyjne w porównaniu obecnych na rynku ofert. W ten sposób możemy zdalnie włączać i wyłączać każde urządzenie podpięte do takiej listwy.



Rysunek 1.1: Zdjęcie złożonego zestawu

Do działania aplikacji potrzebujemy przynajmniej jednej listwy z układem ESP i dostępu do komputera. Podłączamy urządzenie do prądu, a na komputerze wystarczy wpisać w przeglądarce adres `http://192.168.1.186:8090`, by znaleźć się na stronie głównej panelu obsługi naszego urządzenia.



Rysunek 1.2: Strona główna

Jeśli jesteśmy nowymi użytkownikami, najpierw trzeba się zarejestrować. W tym celu należy podać swój login, adres e-mail, hasło oraz numer telefonu. Na podany numer zostanie wysłany czterocyfrowy, losowy kod aktywacyjny, którym potwierdzimy swoją tożsamość i będziemy mogli zalogować się do panelu sterowania.

The registration form for 'Inteligentny Dom' features a dark background with a glowing title. At the top, there are two buttons: 'Strona domowa' (blue) and 'Zaloguj' (red). The form includes four input fields: 'Login', 'Adres e-mail', 'Hasło', and 'Telefon +48000000000'. A green 'Zarejestruj' button is positioned below the fields.

Strona domowa Zaloguj

Inteligentny Dom

Login

Adres e-mail

Hasło

Telefon +48000000000

Zarejestruj

Rysunek 1.3: Rejestracja

The login form for 'Inteligentny Dom' features a dark background with a glowing title. At the top, there are two buttons: 'Strona domowa' (blue) and 'Zarejestruj' (red). The form includes two input fields: 'Adres e-mail' and 'Hasło'. Below the fields are two green buttons: 'Zaloguj' and 'Zapomniałeś hasła?'.

Strona domowa Zarejestruj

Inteligentny Dom

Adres e-mail

Hasło

Zaloguj

Zapomniałeś hasła?

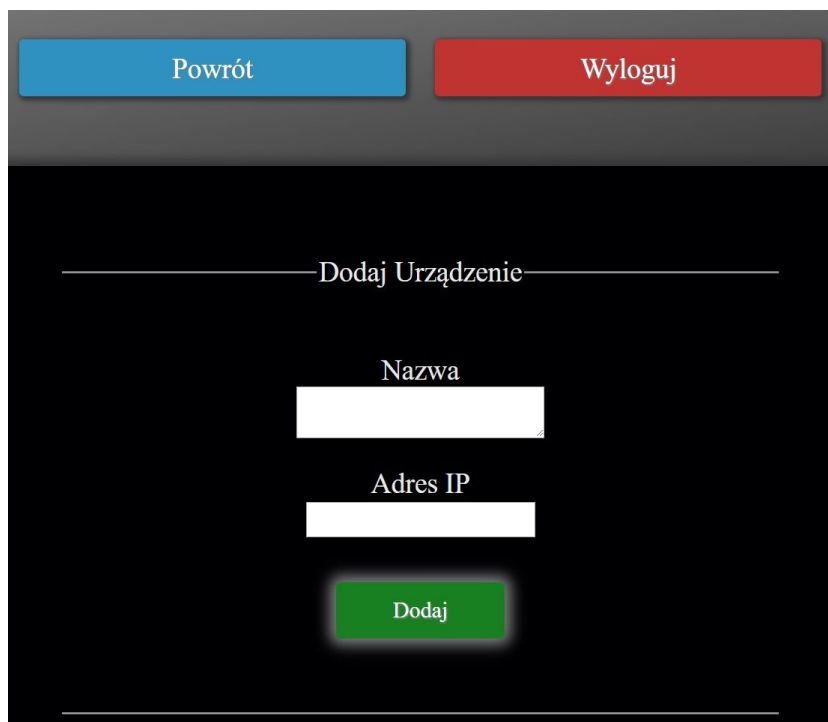
Rysunek 1.4: Logowanie

W przypadku gdy nasze hasło nie działa bądź go zapomnieliśmy, możemy wykorzystać funkcję „Zapomniałeś hasła?”. Po wpisaniu adresu e-mail, wykorzystanego podczas rejestracji, ponownie otrzymamy czterocyfrowy kod, który umożliwi nam przejście do naszego panelu. Po zalogowaniu możemy zmienić hasło na nowe w zakładce „Zmień hasło”.



Rysunek 1.5: Zmiana hasła

Początkowo panel sterowania będzie pusty. Aby dodać do widoku układ, nad którym chcemy sprawować kontrolę, trzeba kliknąć przycisk „Dodaj urządzenie”, a kolejnej stronie, podać wygodną dla nas nazwę (np. kuchnia) oraz adres IP, który został przekazany wraz z urządzeniem. W przypadku braku połączenia, wyświetli się błąd. W takiej sytuacji należy sprawdzić czy listwa z układem ESP otrzymuje zasilanie z gniazdka i czy podłączony do niej sprzęt działa prawidłowo. Następnie proszę powtórzyć próbę dodania urządzenia.



Powrót Wyloguj

Dodaj Urządzenie

Nazwa

Adres IP

Dodaj

Rysunek 1.6: Dodawanie urządzenia



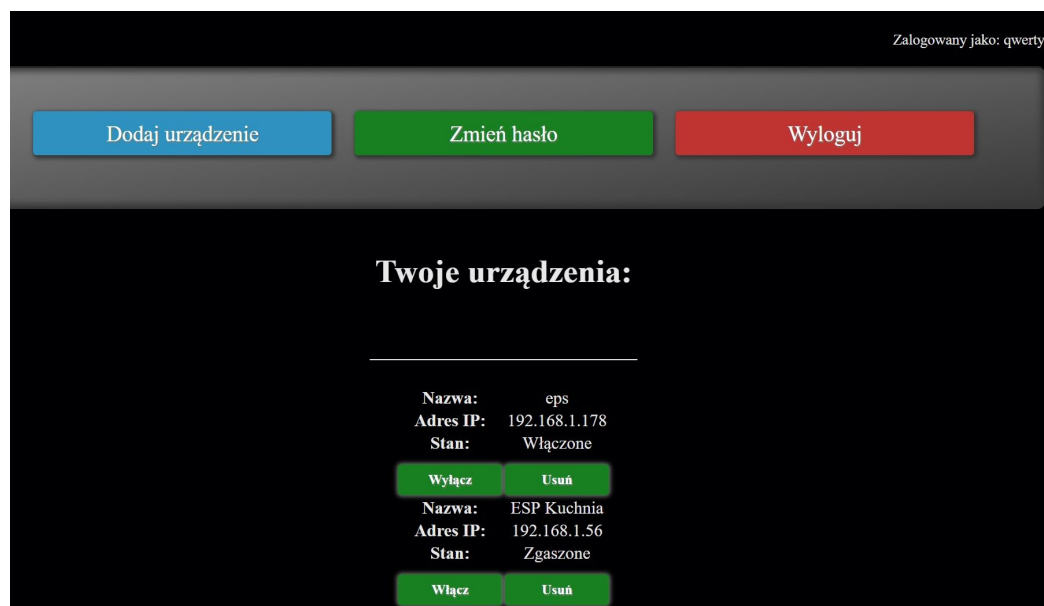
Zalogowany jako: qwerty

Powrót

Nie znaleziono urządzenia.

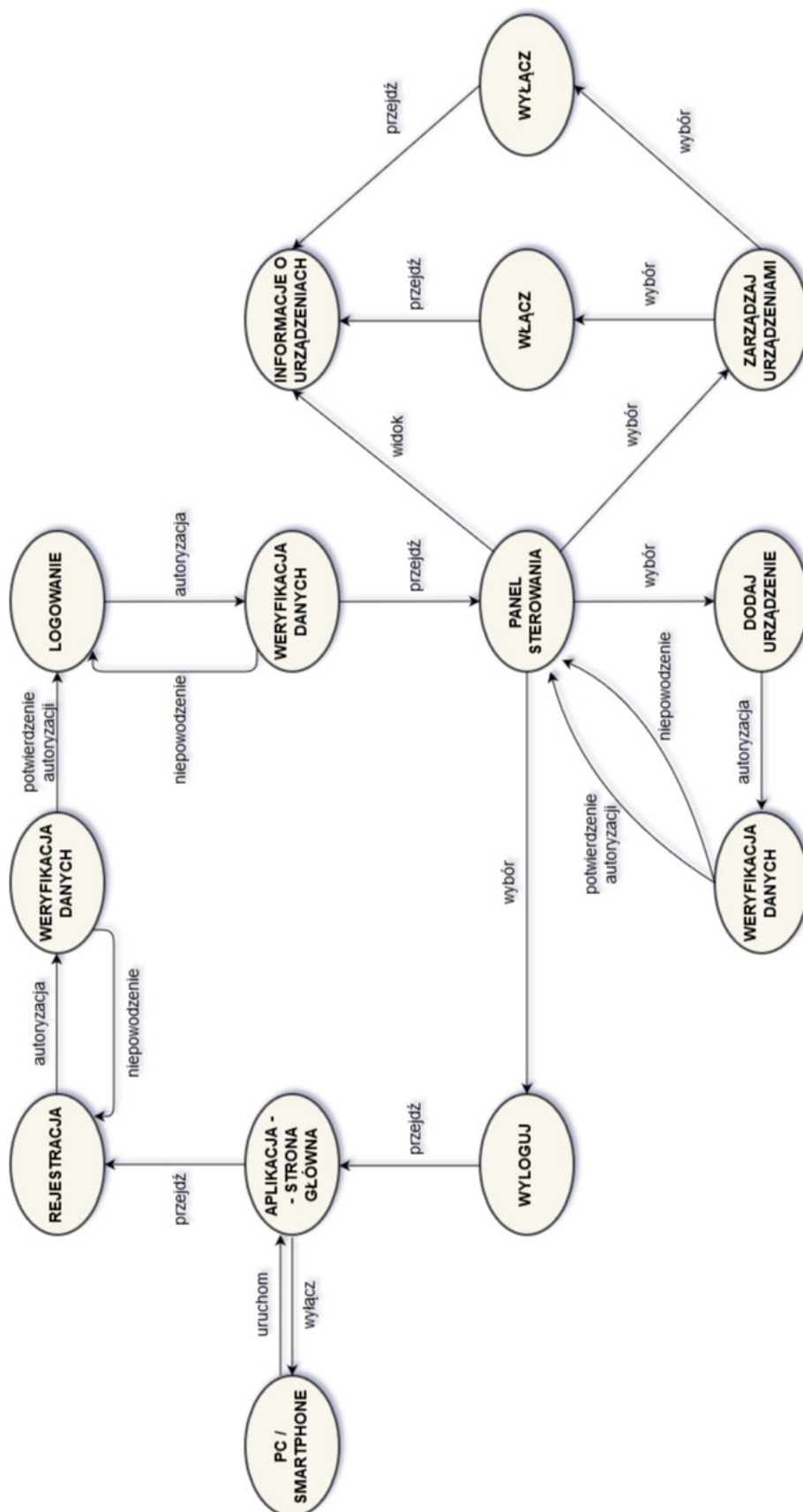
Rysunek 1.7: Komunikat błędu

Po przejściu wszystkich kroków, w panelu sterowania zobaczymy listę z dodanymi przez nas układami. Odczytamy tu podane wcześniej informacje o nazwie, adresie IP a także o stanie urządzenia. Gdy stan określony jest jako „Włączone”, poprzez kliknięcie przycisku „Wyłącz”, odcinamy dopływ prądu i wyłączamy nasz sprzęt, analogicznie w odwrotnym przypadku. W momencie gdy chcemy zmienić nazwę urządzenia lub po prostu usunąć je z listy, wystarczy kliknąć przycisk „Usuń”. Po otrzymaniu potwierdzenia przestanie ono być widoczne w panelu. W każdej chwili urządzenie można dodać ponownie.



Rysunek 1.8: Panel sterowania

Schemat działania aplikacji został przedstawiony na diagramie DFD:



Rysunek 1.9: Diagram DFD – opracowanie własne

ROZDZIAŁ 2

Mobile app

2.1. Kivy – czym jest?

Jest to biblioteka stworzona z myślą o interfejsach graficznych. Aplikację w której kivy zostało wykorzystane możemy uruchomić na Windowsie, Linuksie, MacOS, Androidzie oraz iOS. Istnieją dwie metody tworzenia GUI, z poziomu Pythona oraz Kvang. Obie metody można ze sobą dowolnie łączyć. Sam Framework istnieje od niedawna, jest typem open source, wciąż rozwijanym i darmowym. Wydają go twórcy, którzy na codzień zajmują się właśnie dystrybucją oprogramowania na urządzenia mobilne. Zostali oni również zauważeni przez Python Software Foundation, którzy wsparli ich dotacją w 2012 roku, za port do Pythona3. Biblioteka podlega warunkom MIT.

2.2. Dlaczego został wybrany?

Jednym z przedmiotów podczas realizowania programu nauczania studiów I stopnia informatyki był właśnie Python. Ze względu na czytelność i klarowność jego kodu został wybrany na język przewodni w naszym projekcie. Jako, że Kivy^[1] jest tworzone w Pythonie, a to główna zaleta odnośnie alternatywnych rozwiązań, zdecydowaliśmy się właśnie na ten framework. Również zaletami jest łatwe i niestandardowe tworzenie wyglądu. Dokumentacja jest napisana w sposób czytelny, zrozumiały dla osób mających styczność pierwszy raz z Kivy. Problemem przede wszystkim może być to, że ta technologia jest dość „świeża”, co za tym idzie szukanie pomocy na forach kończy się praktycznie zerowym odzewem. Jeszcze jednym zauważalnym minusem okazuje się być narzędzie do „deploy’owania” – Buildozer, który działa poprawnie tylko na linuxie.

2.3. Kivy ScreenManager

Menadżer ekranu¹ jest to widżet posiadający funkcję zarządzania wieloma ekranami. Domyślnie, wyświetla tylko jeden ekran na raz i używa funkcji `TransitionBase` do przejścia do następnego widoku. Wiele z nich jest obsługiwana w oparciu o zmianę współrzędnych ekranu, pozwala również na wykonywanie niestandardowych animacji za pomocą shaderów. Poruszanie się po naszej aplikacji oparliśmy właśnie o `ScreenManager`. Wykorzystaliśmy do tego przyciski(ang. `button`), za którymi kryje się funkcja odpowiadająca za komunikowanie się z widokiem napisanym w pliku `kv`.

```
def login{self}:  
    self.root.current = 'login'
```

Przykładowo prosta funkcja, która odwołuje się do naszego w tym wypadku podanego w cudzysłowie ID – `login`. Jeśli podane przez nas ID w funkcji istnieje w pliku `kv`, `ScreenManager` przeniesie nas w aplikacji do docelowego widoku.

Screen:

```
name: 'login'
```

Screen oznacza w tym wypadku rozpoczęcie nowego widoku i przypisania mu identyfikatora. W naszym wypadku jest to `login`, aby w późniejszym rozbudowaniu aplikacji kod był przejrzysty i intuicyjny.

2.3.1. Komunikacja widoku z funkcją

Button:

```
text: 'Login'  
on_press: app.login()  
background_normal: 'button_normal.png'  
background_down: 'button_down.png'
```

¹<https://kivy.org/docs/api-kivy.uix.screenmanager.html>

W naszym widoku zdefiniowaliśmy przycisk. Posiada on pole typu text, gotową funkcję on_press zaczerpniętą z frameworku, oraz wygląd zaimportowany z plików png. On_press, jak sama nazwa wskazuje, po wciśnięciu wykona pożądane przez nas działanie. W tym wypadku, jeśli istnieje app.login zostanie wywołana funkcja.

2.3.2. Przykładowy screen

```
Screen :
name: 'login '
GridLayout :
    label :
        id: username_label
        text: 'Nazwa użytkownika: '
        TextInput :
            id: username_input
    label :
        id: password_label
        text: 'Hasło: '
        TextInput :
            id: password_input
        password: True
    Button :
        text: "Zaloguj"
        on/_press: app.zaloguj()
        background_normal: 'button_normal.png'
        background_down: 'button_down.png'
    Button :
        text: "Cofnij"
        on_press: app.cofnij()
        background_normal: 'red_button_normal.png'
        background_down: 'red_button_down.png'
```


W pełni zdefiniowany

2.4. Layout w Kivy

W kivy tworzenie layout'u jest dość specyficzne. Jest kilka możliwości ułożenie naszych widżetów w określony sposób. Wyróżniamy:

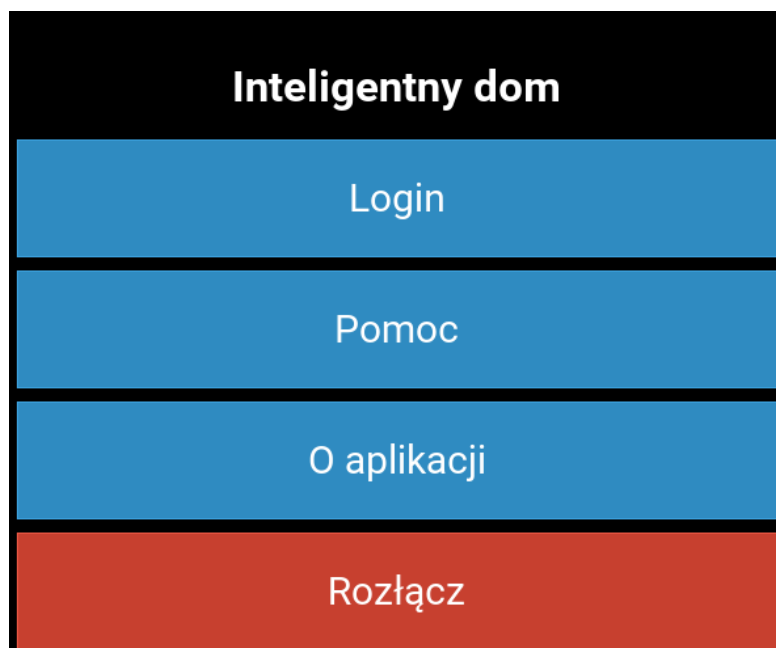
- Anchor Layout mogą być zaczepione do górnego, dolnego, lewego, prawego boku lub do środka.
- BoxLayout: ułożenie w orientacji pionowej lub poziomej.
- FloatLayout: nieograniczony układ.
- RelativeLayout: są rozmieszczane względem układu.
- GridLayout: ułożone w „siatkę” zdefiniowaną przez rzędy i kolumny.
- PageLayout: do tworzenia układów wielostronicowych, umożliwiając w łatwy sposób przewijanie z jednej strony na drugą.
- ScatterLayout: podobnie jak w RelativeLayout, ale można je obracać i skalować.
- StackLayout: ułożenie widżetów od lewej do prawej. W naszej aplikacji wykorzystaliśmy GridLayout oraz BoxLayout.

2.4.1. BoxLayout

Tak wygląda nasze menu aplikacji mobilnej po podłączeniu się do serwera za pomocą frameworku Twisted. Jest ono zdefiniowane w wyżej wymienionym BoxLayout. Został ułożony on w orientacji pionowej. Definiuje się to w następujący sposób:

BoxLayout :

```
orientation : 'vertical'
```



Rysunek 2.1: Menu w BoxLayout

Ważną kwestią w tworzeniu layout'u jest zaprogramowanie jego parametrów. W tym wypadku najważniejszym będzie ustawienie odstępów między odstępami. Czyli:

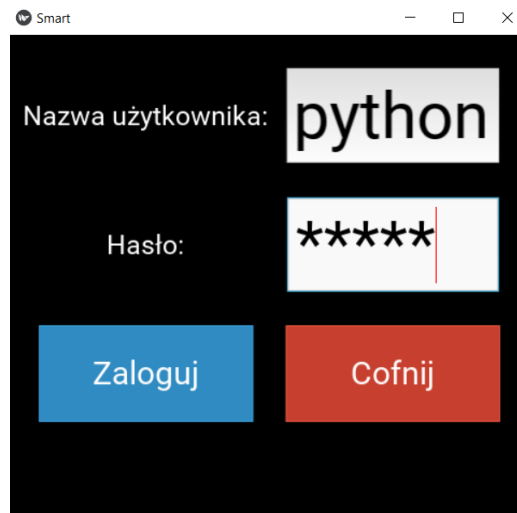
```
<BoxLayout >:  
    padding: 10  
    spacing: 10
```

Przy czym padding jest wypełnieniem pozycji pomiędzy dwoma polami, a spacing odstępami podanymi w pikselach.

2.4.2. GridLayout

```
<GridLayout>:
    rows: 4
    cols: 2
    padding: 30
    spacing: 30
    row_default_height: 90
    row_force_default: True
```

W ten sposób definiujemy GridLayout w pliku Kv. Rows odpowiada za maksymalną ilość wierszy, a cols kolumn. Jeśli w tworzeniu naszego widoku po użyciu właśnie zdefiniowanego layout'u użyjemy więcej pól – przykładowo przycisków czy tabel – nie zostaną one uwzględnione. Dodatkowo została użyta funkcja `row_force_default`, która jeśli jest `True`, nakazuje ignorować narzuconą wysokość oraz `size_hint_y`, przez co używa domyślnej wysokości wiersza.

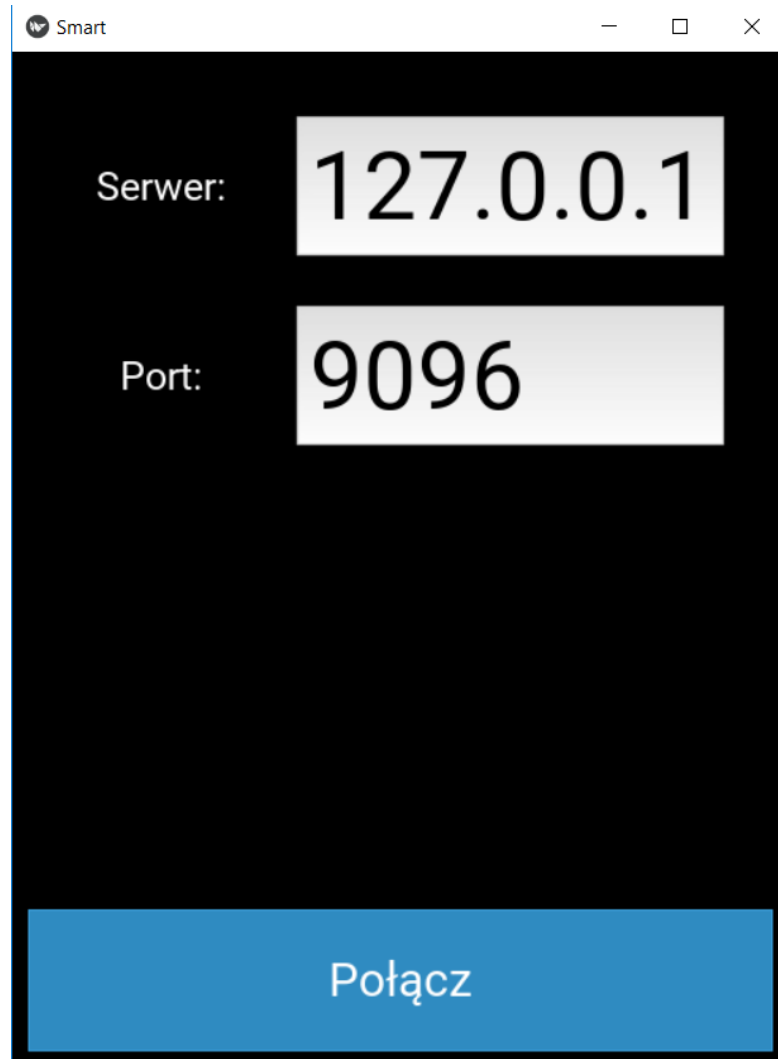


Rysunek 2.2: Ekran logowania w GridLayout

2.5. Twisted

Naszym pierwszym krokiem przy tworzeniu aplikacji mobilnej było stworzenie połączenia między nią a naszym urządzeniem sterującym. Do tego potrzebowaliśmy frameworku sieciowego. Twisted[2] obejmuje obsługę popularnych protokołów sieciowych, takich jak na przykład HTTP i SMTP. Oszczędza bardzo wiele pracy programiście, ze względu na dużą ilość funkcji, klas oraz korzysta z introspekcji Pythona. Do testów został stworzony również serwer oparty o tą technologię. Największą zaletą był brak wymaganej konfiguracji, przez co stworzenie go, nie wymagało nadmiaru pracy.

2.5.1. Kivy z Twisted



Rysunek 2.3: Widok łączenia się z serwerem

ROZDZIAŁ 3

Web app

3.1. Python – Flask – co to takiego?

„Python jest dynamicznie typowanym językiem interpretowanym wysokiego poziomu”[3]. Oznacza to, że Python jest wykonywany na bieżąco, linijka po linijce. Nie jest on kompilowany, a jedynie wczytywany i wykonywany podczas uruchomienia przez interpreter języka. Języki skryptowe są mniej wydajne od języków kompilowanych, jednakże są dość proste w użyciu, przez co dobrze sprawdzają się podczas pisania niewielkich aplikacji. Podstawowymi framework’ami¹ Python’a są Django oraz Flask[4]. Naszym założeniem było stworzenie prostej i szybkiej aplikacji, dlatego zdecydowaliśmy się na wykorzystanie Flask’a. Jest to micro-framework obsługujący rozszerzenia, które mogą dodawać funkcje do aplikacji, takie jak sprawdzanie poprawności formularzy, obsługa wysyłania czy otwarte technologie uwierzytelniania. Ponadto jest kompatybilny z Google App Engine, zawiera zintegrowaną obsługę testów jednostkowych, obsługę plików cookie, a przede wszystkim posiada serwer programistyczny i debugger. Dzięki temu nie wymaga żadnych dodatkowych narzędzi. Dzięki swoim funkcjom, Flask upraszcza projektowanie stron www.

3.2. Zastosowanie w aplikacji

Serce naszej aplikacji, główny kod i schemat działania, zawarty jest w czterech podstawowych plikach: `run.py`, `model.py`, `views.py` oraz `__init__.py`. Pierwszy plik określa ścieżkę do serwera, drugi odpowiada za komunikację z bazą danych a trzeci za wyświetlanie i zachowanie stron

¹<https://pl.wikipedia.org/wiki/Framework>

internetowych. Plikiem, bez którego działanie aplikacji nie byłoby możliwe jest `__init__.py`. Plik ten odpowiada za import klas bądź funkcji do poziomu pakietu, aby można je było wygodnie importować w późniejszym etapie. Flask używa nazwy „import”, aby wiedzieć, gdzie szukać zasobów, szablonów, plików statycznych, folderu instancji, itp. Komendy:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager
```

umożliwiają nam korzystanie z samego Flask’a oraz zaimportowanie rozszerzeń SQLAlchemy oraz LoginManager, a polecenia takie jak

```
from www.model import baza, login_manager
from www import views
```

odpowiadają za wskazanie lokalizacji powiązanych plików, z kolei

```
app = Flask(__name__)
```

wskazuje na miejsce przechowywania zasobów aplikacji.

3.2.1. Flask – SQLAlchemy

SQLAlchemy jest stworzonym dla Pythona narzędziem, który wraz z ORM² oferuje pełną funkcjonalność języka SQL bez konieczności samodzielnego pisania skomplikowanych zapytań. Flask–SQLAlchemy z kolei jest niczym innym jak rozszerzeniem dodającym usługę SQL do programu.

W naszym przypadku do pełnego i bezpiecznego działania aplikacji, konieczne było stworzenie tabel przechowujących dane logujących się osób oraz informacje o podłączanych urządzeniach. Aby nasza baza mogła funkcjonować w pliku `__init__.py` utworzyliśmy jej instancję

```
baza.init_app(app)
app.static_folder = 'static'
baza.app = app
baza.create_all()
```

by w pliku `model.py` móc zadeklarować tabele „User” oraz „Esp”

²Object Relational Mapper


```
class User(baza.Model, UserMixin):
    id = baza.Column('id', baza.Integer, primary_key=True)
    imie = baza.Column('imie', baza.String)
    email = baza.Column('email', baza.String, unique = True)
    haslo = baza.Column('haslo', baza.String)
    telefon = baza.Column('telefon', baza.String)
    aktywne = baza.Column('aktywne', baza.Boolean, default=False)
    pin = baza.Column('pin', baza.Integer)
    def __init__(self, imie, email, haslo, telefon, aktywne, pin):
        self.imie = imie
        self.email = email
        self.haslo = self.hash_password(haslo)
        self.telefon = telefon
        self.aktywne = aktywne
        self.pin = pin
    def __repr__(self):
        return ('<User %r>' % self.email)

    def hash_password(self, password):
        return pwd_context.encrypt(password)

    def verify_password(self, password):
        return pwd_context.verify(password, self.haslo)

class Esp(baza.Model):
    id = baza.Column('id', baza.Integer, primary_key=True)
    nazwa = baza.Column('nazwa', baza.String)
    ip = baza.Column('ip', baza.String)
    stan = baza.Column('stan', baza.Boolean, default=False)
    def __init__(self, nazwa, ip, stan):
        self.nazwa = nazwa
        self.ip = ip
        self.stan = stan
```

Jak widać nie stosowaliśmy tu klasycznego zapisu języka SQL, a każda tabela powstała jako klasa, w której deklarujemy poszczególne kolumny i ich właściwości. Funkcje "def __init__" służą przypisaniu odpowiednich wartości, z kolei "def hash_password" oraz "def verify_password" są wbudo-

wanymi funkcjami Python’a, wykorzystanymi do podniesienia poziomu bezpieczeństwa poprzez zaszyfrowanie hasła wpisywanego przez użytkownika, oraz jego weryfikację podczas logowania. Wszelkie dane pobierane od użytkownika, przesyłane są za pomocą metod „GET” i „POST”. W pliku view.py, poprzez tzw. dekoratory deklarujemy powiązanie funkcji z adresem URL. Na przykład w przypadku próby zarejestrowania się, kliknięcie przycisku „Zarejestruj” wywoła funkcję „def register”.

```
@app.route('/register', methods = ['GET', 'POST'])
def register():
    if request.method == 'POST':
        ...
        uzytkownik = User(imie=request.form['imie'], email=
            request.form['email'], haslo=request.form['haslo'],
            telefon=telefon, aktywne = 0, pin=pin )
        ...
        return redirect(url_for('autoryzacja'))
    return render_template('signup.html')
```

Funkcja ta w przypadku poprawnego przekazania danych, generuje losowy kod weryfikacyjny, zapisuje informacje uzyskane do użytkownika do bazy danych poprzez zmienną „uzytkownik”, następnie wywołuje funkcję oraz stronę odpowiedzialną za autoryzację kodu aktywacyjnego i w przypadku wykonania wszystkich zadań z powodzeniem, przekierowuje nas na stronę logowania. Jak widać samo zapytanie SQL przekazujące informacje odbywa się przez „request.form” i nie zajmuje więcej niż jedną linię kodu, dzięki czemu możemy skupić się na budowaniu innych funkcji.

3.2.2. Flask – LoginManager

Flask – LoginManager^[5] jest narzędziem umożliwiającym zarządzanie sesją użytkownika poprzez obsługę zadań takich jak logowanie, wylogowywanie czy zapamiętywanie identyfikatora użytkownika w sesji. Aby móc korzystać z tych funkcji należy najpierw stworzyć model użytkownika o właściwościach:

```
@property
def is_authenticated(self):
    return True
@property
def is_active(self):
    return True
def is_anonymous(self):
    return False
def get_id(self):
    return str(self.email)
```

Odpowiadają one za weryfikację czy klient podał prawidłowe poświadczenia, czy jego konto jest aktywne oraz czy bieżący użytkownik jest anonimowy. Metoda `get_id` bierze pod uwagę instancję klasy `User` 3.2.1 i zwraca unikalny numer dla tego obiektu. Identyfikator użytkownika jest brany pod uwagę w funkcji „`user_loader`”:

```
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

Ma ona za zadanie przekazać programowi z jakim użytkownikiem będzie pracował i jakie dane ma wyświetlić.

Proces logowania odbywa się z kolei w funkcji „`login`”:

```
@app.route('/login', methods = ['GET', 'POST'])
def login():
    if request.method == 'POST':
        haslo = request.form['haslo']
        mail = request.form['email']
        user=User.query.filter(User.email==mail).first()
        if user:
            if user.verify_password(haslo) == True:
                if user.aktywne == 1:
                    login_user(user, remember=True)
                    return redirect(url_for('list_all'))
                return redirect(url_for('autoryzacja'))
            return render_template('error.html', error = 'Haslo
niepoprawne.')
```

```
return render_template('error.html', error = 'Nie ma  
takiego konta.')  
return render_template('signin.html')
```

Adres e-mail oraz hasło przekazywane są za pomocą metod „GET” i „POST”. W pierwszej kolejności polecenie „User.query.filter(User.email==mail).first()” wyszukuje w bazie użytkownika o podanym adresie e-mail, by kolejno, zweryfikować poprawność wprowadzonego hasła, sprawdzić czy konto użytkownika zostało aktywowane oraz by pobrać i zapamiętać jego identyfikator dla bieżącej sesji. W przypadku jakiegokolwiek błędu otrzymamy komunikat bądź zostaniemy przekierowani na odpowiednią stronę.

Przed wywołaniem każdej funkcji, która ma za zadanie wyświetlać lub przekazywać prywatne informacje, skorzystaliśmy z dekoratora „login_required”. Ma on za zadanie zapewnić poprawność wyświetlanych informacji dla danego użytkownika oraz uniemożliwić dostęp osobom niepowołanym. Na przykład przed funkcją przekierowującą do strony panelu sterowania:

```
@app.route('/wyswietl')  
@login_required  
def list_all():  
    esp=Esp.query.all()  
    return render_template('showDevices.html', esp = esp)
```

3.3. Autoryzacja SMS

Użytkownik, który stworzył konto w naszej aplikacji, przy pierwszym logowaniu musi wykonać jednorazową autoryzację konta za pomocą kodu PIN, który otrzymał w formie SMS’a³. Kod PIN generowany jest wbudowaną funkcją Pythona „random”:

```
pin = random.randint(1000,9999)
```

Kod zostaje przypisany użytkownikowi i zapisany w bazie danych.

Wysyłanie wiadomości tekstowej na wskazany przy rejestracji nr telefonu odbywa się przez modem GSM oraz funkcję „wyslijsms”:

³ang. *Short Message Service*

```
def wyslij_sms(number, pin):
    text = pin
    number = number
    s = serial.Serial('/dev/ttyUSB2', 115200, timeout=1)
    s.write(bytes('AT\r', 'UTF-8'))
    time.sleep(1)
    msg=s.read(64)
    print(msg)
    s.write(bytes('AT+CMGF=1\r', 'UTF-8'))
    time.sleep(1)
    msg=s.read(64)
    print(msg)
    s.write(bytes('AT+CMGS="%s"\r' % number, 'UTF-8'))
    #Send message:
    time.sleep(1)
    msg=s.read(64)
    print(msg)
    s.write(bytes('%s\x1a' % text, 'UTF-8'))
    msg=s.read(64)
    print(msg)
    s.close()
    return 0
```

Komunikacja aplikacji i modemu GSM odbywa się przez komendy AT⁴ oraz bibliotekę PySerial 3.4[6]⁵

3.4. Tworzenie stron internetowych

Przy projektowaniu aplikacji internetowej posłużyliśmy się językiem HTML[7]⁶ służącym do umieszczania treści na stronie oraz CSS[8]⁷ odpowiadającym za formatowanie jej wyglądu. Oba języki są proste w użyciu, a co najważniejsze, dzięki swojej popularności wspierane większość przeglądarek internetowych. Jedyną wadą jest różne działanie arkuszy stylów na po-

⁴<https://sonnguyen.ws/send-sms-from-raspberry-pi-with-usb-3g/>

⁵<http://pyserial.readthedocs.io/en/latest/pyserial.html>

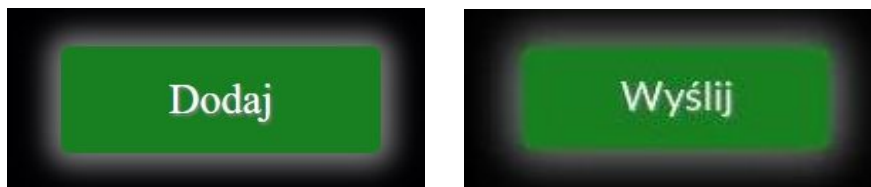
⁶ang. *HyperText Markup Language*

⁷ang. *Cascading Style Sheets*

szczególnych przeglądarkach, jednakże zasady ich działania są proste i przy zachowaniu odpowiednich reguł, nie musimy obawiać się niepoprawnego wyświetlania treści naszej strony. W sieci istnieje wiele gotowych (darmowych bądź płatnych) arkuszy, które możemy pobrać i zaimportować do swojego projektu, dzięki temu oszczędzimy swój czas, a strona będzie wyglądać bardziej profesjonalnie.

W przypadku naszego projektu korzystanie z gotowych szablonów nie było konieczne. Z założenia strona wyglądem miała być zbliżona do aplikacji mobilnych, które cechuje dość prosty design w kontrastowych barwach, aby dobrze prezentowały się na małym wyświetlaczu telefonu.

Podstawą zasadą przy tworzeniu stron HTML jest trzymanie się ogólnych norm przyjętych przez W3C⁸. Przy dbaniu o przejrzystość kodu, poprawnym stosowaniu znaczników oraz atrybutów, dopisanie do naszej strony arkusza stylu nie powinno sprawić większych problemów. Za powiązanie języka CSS z HTML'em odpowiedzialne są tzw. selektory, którymi mogą być elementy HTML oraz atrybuty „id” oraz „class”. Przykładem w naszej pracy może być określenie wyglądu przycisków „Dodaj” bądź „Wyślij”.



Rysunek 3.1: Przyciski na stronach ”Dodaj urządzenie” oraz ”Zmień hasło”

```
<div>
    <input class="button" type = "submit" value = "Wyslij" />
</div>

<div>
    <input class="button" type = "submit" value = "Dodaj" />
</div>
```

⁸https://pl.wikipedia.org/wiki/World_Wide_Web_Consortium

```
.button {  
    margin-top: 20px;  
    height: 50px;  
    width: 150px;  
    cursor: pointer;  
    background-color: #188021;  
    border: 2px solid #188021;  
    color: white;  
    font-size: 20px;  
    font-weight: 400;  
    border-radius: 4px;  
    box-shadow: 0px 0px 20px 6px grey;  
    text-shadow: 1px 1px 1px grey;  
}  
.button:hover {  
    color: #188021;  
    background-color: #E8FFE9;  
}
```

Mamy tutaj przykład dwóch przycisków opisanych atrybutami „value”, „type” i „class”. Pierwszy określa etykietę przycisku, drugi rodzaj wstawianego pola, a trzeci jest odwołaniem do pliku .css. Selektor „button” w naszym arkuszy stylu określa m.in. rozmiar przycisku (height, width), wygląd czcionki (color, font-size, font-weight, text-shadow), tła (background-color) czy nawet zaokrąglenia rogów (border-radius). Klasa „button:hover” z kolei odpowiada za zachowanie przycisku po najejchaniu na niego myszką, w tym przypadku jego kolor i czcionka ulegną zmianie na przeciwne.

Obecnie treść przeglądarek internetowych możemy wyświetlać nie tylko na komputerach ale i telewizorach, projektorach, czytnikach e-book’ów oraz telefonach. W zależności od rozmiaru ekranu treść naszej strony może się nie zmieścić lub zostać rozciągnięta. Dlatego wraz z wprowadzeniem nowej wersji CSS3 zostały rozbudowane reguły @media. Dzięki nim możemy zmienić wygląd naszej strony w zależności na jakim urządzeniu zostanie wyświetlona. Nasza strona została dostosowana do małych ekranów telefonów (poniżej 730px) tak, aby elementy na niej zawarte dostosowały swoją pozycję.

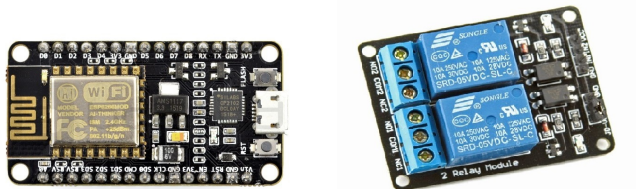
```
@media only screen and (min-width: 730px) {  
  .footer {  
    padding-right: 0;  
    padding-left: 0;  
  }  
  .header {  
    margin-bottom: 30px;  
  }  
  .jumbotron {  
    border-bottom: 0;  
  }  
}
```


ROZDZIAŁ 4

Hardware

4.1. ESP8266

W naszym projekcie wykorzystaliśmy moduł WiFi oparty na układzie ESP8266-12E^[9]¹ jako sterownik przekaźnika.



Rysunek 4.1: Moduł ESP8266 i Przełącznik 2-kanalowy

Esp Nodemcu v2 to programowalny mikrokontroler posiadający 32-bitowy CPU, 4MB pamięci Flash oraz 16 wyprowadzeń GPIO. Komunikacja odbywa się przez WiFi 802.11 b/g/n na częstotliwości 2,4 GHz. Do programowania układu wykorzystałem język "C" i środowisko Arduino IDE². Programator UART, który jest wlutowany w płytke ułatwia wygrywanie programów przez standardowy kabel USB.

ESP8266 został zaprogramowany do działania w trybie klienta WiFi w naszej lokalnej sieci. Dane dla naszej sieci podajemy przy programowaniu urządzenia:

```
char* ssid = "NawaSieci";  
char* pass = "1HasoSieci";
```

```
void setup() {
```

¹<http://www.kloppenborg.net/images/blog/esp8266/esp8266-esp12e-specs.pdf>

²<https://www.arduino.cc/en/main/software>

```
WiFi.begin(ssid, pass);  
}
```

Od tej chwili sterownik jest podłączony do routera.

Przełącznik, który włącza lub wyłącza wysokie napięcie wyposażony jest w optoizolację³ zapewniając odseparowanie napięcia 230V a ESP8266.

4.2. Raspberry pi 3 – jako serwer aplikacji

Mikrokomputer wyposażony w czterordzeniowy procesor ARM, 1 GB RAM, porty USB oraz HDMI. Cało pod kontrolą Linuxa specjalnie przygotowany dla Raspberry Pi 3[10]⁴. Zasilany przez 5V ładowarkę np. telefoniczną. Taką konfigurację posiada nasz serwer, który obsługuje aplikację webową.



Rysunek 4.2: Raspberry Pi 3

4.3. Komunikacja

Raspberry Pi 3 i Esp8266 komunikują się ze sobą za pomocą pomocą JSON'ów. Gdy klient postawowi włączyć/wyłączyć urządzenie za pomocą ESP wywołuje funkcje na RPI 3 "swiatloon" lub "swiatlooff". Obie funkcje działają w identyczny sposób, jedyną różnicą jest wartość pola "stan1" dla GPIO ESP generowanego JSONa. Funkcja "swiatloon" wygląda następująco:

³<https://pl.wikipedia.org/wiki/Transoptor>

⁴<http://docs-europe.electrocomponents.com/webdocs/14ba/0900766b814ba5fd.pdf>

```
def swiatloon(idd):
    esp = session.query(Esp).filter(Esp.id ==int(idd)).one()
    conn = http.client.HTTPConnection(esp.ip)
    payload = "{\"stan1\":\"1\"}"
    headers = {
        'content-type': "application/json",
    }
    conn.request("POST", "/relay", payload, headers)
    res = conn.getresponse()
    data = res.read()
    return redirect(url_for('list_all'))
```

Pod zmienną `esp` zostaje przypisany obiekt ESP wyfiltrowany po ID. W `payload` przechowujemy stan GPIO do wysłania. Za samo przesłanie JSON'a na odpowiednie IP ESP odpowiada linia kodu `conn.request("POST", "/relay", payload, headers` oraz biblioteka `http.client`.

Na adres 192.168.1.56/relay trafia pakiet danych z serwera RPI 3. ESP również jest serwerem, który przetwarza dane za pomocą funkcji:

```
void setRelay() {

    String data = server.arg("plain");
    StaticJsonBuffer<200> jBuffer;
    JsonObject& jobject = jBuffer.parseObject(data);
    String stan1 = jobject["stan1"];
    int value1;
    String stanJson;
    value1=(stan1.toInt());
    if (value1==1){
        digitalWrite(pin, LOW);
        Serial.println(stan1);
        stanJson= "1";
    }
    else{
        digitalWrite(pin, HIGH);
        Serial.println(stan1);
        stanJson= "0";
    }
    StaticJsonBuffer<200> JSONbuffer;
```

```

JsonObject& JSONEncoder = JSONbuffer.createObject();
String ip = WiFi.localIP().toString();
short int port = 8090;
JSONEncoder["ip"] = (ip);
JSONEncoder["stan"] = stanJson;
    char JSONmessageBuffer[200];
    JSONEncoder.prettyPrintTo(JSONmessageBuffer, sizeof(
        JSONmessageBuffer));
    Serial.println(JSONmessageBuffer);
    HTTPClient http;
    String adres="http://";
    adres+="192.168.1.186";
    adres+=":";
    adres+=port;
    adres+="/readjson";
    Serial.println();
    Serial.println(adres);
    http.begin(adres);
    http.addHeader("Content-Type", "application/json");
    int httpCode = http.POST(JSONmessageBuffer);
    String payload = http.getString();
    Serial.println(httpCode);
    Serial.println(payload);
    http.end();
    server.send(200,"ok");
}

```

Po odczytaniu wartości `stan1` Esp zmienia stan pinu, czyli włącza prąd na danym porcie i uruchamia przekaźnik. Dalszy kod funkcji jest odpowiedzialny za odesłanie do serwera odpowiedzi w formie JSON'a z obecnym stanem urządzenia i jego IP. Dopiero, gdy serwer otrzyma odpowiedź na `/readjson` wraz z numerem ip i stanem goldpinu 8266, aktualizuje bazę danych:

```

@app.route('/readjson', methods=['POST'])
def readjson():
    print(request.is_json)
    content = request.get_json()
    print(content)
    ip = content["ip"]
    stan = content["stan"]

```

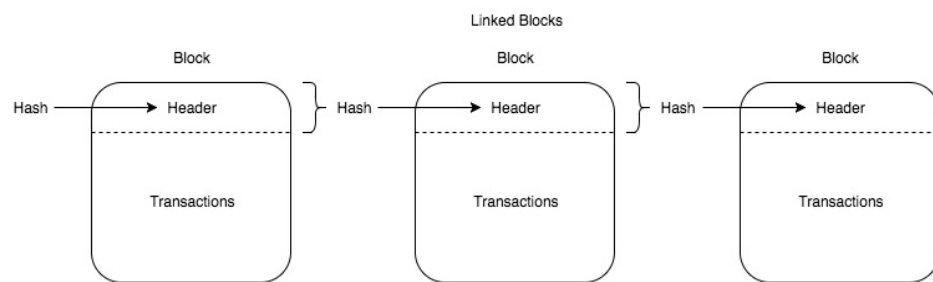
```
print(stan, ip)
esp = session.query(Esp).filter(Esp.ip ==ip).one()
esp.stan = int(stan)
session.commit()
return redirect(url_for('list_all'))
```

4.4. Bezpieczeństwo w projektach "Smart Home"

Sieć Internetu Rzeczy i same urządzenia przetwarzają dane o naszej codziennej aktywności dobowej, np. kiedy włączamy światło. Po dłuższej analizie takich zachowań określić, czy jesteśmy w domu, na jak długo go opuszczamy lub kiedy do niego wracamy. Informacje takie jeśli dostaną się w niepowołane ręce mogą posłużyć do fizycznego włamania do naszego mieszkania.

Każde urządzenie w naszej sieci Internetu Rzeczy może być potencjalnym punktem włamania do infrastruktury IOT i ujawnieniem wrażliwych danych. Poprawą bezpieczeństwa może pomóc nam Blockchain[11][12]. Technologia ta opiera się o sieć peer-to-peer. Czynności odbioru lub nadaniu danych przez urządzenie jest rozgłaszane w sieci przez co klienci rejestrują je w swoich blokach i zapisują w księdze transakcji. Bloki takich informacji zostają zapieczętowane. Do zapieczętowania wykorzystywane są poprzednie bloki, dzięki czemu próba manipulacji na jednym bloku powoduje dezintegrację całości.

Wykorzystanie technologii Blockchain[13] w sieci IOT ogranicza ryzyko włamań przez redukcję centralnych systemów autoryzacji, łatwą weryfikację danych oraz niezmiennosc.



Rysunek 4.3: Pieczętowanie bloków

Zakończenie

Założeniem projektu było stworzenie aplikacji prostej w obsłudze oraz systemu czujników do kontrolowania przepływu prądu. Cele, które założyliś przy projektowaniu sytemu zostały przez nas osiągnięte, jedynie co wymaga poprawy to bezpieczeństwo komunikacji przy wymianie danych między serwerem a czujnikiem oraz jeszcze prostrza konfiguracja połączenia ESP z siecią lokalą.

Bibliografia

- [1] The Kivy Authors. Welcome to kivy. *Kivy Documentation*, 2017. Materiał wykorzystany 29 marca 2018.
- [2] Dave Peticolas. Twisted introduction. *multi-part introduces*, 2018. Materiał wykorzystany 12 kwietnia 2018.
- [3] Centrum Edukacji Obywatelskiej. Szkolenia Python 101. *Python 101*, 2017. Materiał wykorzystany 10 kwietnia 2018.
- [4] Armin Ronache. Welcome to Flask. *Flask web development, one drop at a time*, 2017. Materiał wykorzystany 11 kwietnia 2018.
- [5] Matthew Frazier. Flask-Login 0.4.1 documentation. *Flask-Login Documentation*, 2017. Materiał wykorzystany 25 kwietnia 2018.
- [6] Chris Liechti. pySerial. *readthedocs.io*, 2016. Materiał wykorzystany 10 kwietnia 2018.
- [7] VTech SEO. Advantages of HTML. *VTech SEO, Advantages of an HTML - based website...*, 2011. Materiał wykorzystany 13 kwietnia 2018.
- [8] W3.CSS. CSS Tutorial. *w3schools.com, Learn CSS*, 2018. Materiał wykorzystany 13 kwietnia 2018.
- [9] AI-Thinker. ESP-12E WiFi Module Version1.0. *kloppenborg.net*, 2015. Materiał wykorzystany 10 maja 2018.
- [10] Raspberry pi hardware. *raspberrypi.org*, 2016. Materiał wykorzystany 10 kwietnia 2018.
- [11] Pluralsight. Blockchain Architecture. *Blockchain Architecture*, 2017. Materiał wykorzystany 10 maja 2018.
- [12] Libby Plummer. Blockchain, Internet rzeczy i bezpieczeństwo – Intel. *Intel*, 2018. Materiał wykorzystany 10 maja 2018.

- [13] Khwaja Shaik. Why blockchain and IoT are best friends. *IBM*, 2018.
Materiał wykorzystany 10 maja 2018.

Spis rysunków

1.1. Zdjęcie złożonego zestawu	7
1.2. Strona główna	8
1.3. Rejestracja	9
1.4. Logowanie	9
1.5. Zmiana hasła	10
1.6. Dodawanie urządzenia	11
1.7. Komunikat błędu	11
1.8. Panel sterowania	12
1.9. Diagram DFD – opracowanie własne	13
2.1. Menu w BoxLayout	18
2.2. Ekran logowania w GridLayout	20
2.3. Widok łączenia się z serwerem	22
3.1. Przyciski na stronach "Dodaj urządzenie" oraz "Zmień hasło"	30
4.1. Moduł ESP8266 i Przekaznik 2-kanałowy	33
4.2. Raspberry Pi 3	34
4.3. Pieczętowanie bloków	38

Oświadczenie

Ja, niżej podpisany(a) oświadczam, iż przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób. Określenie indywidualnego wkładu w pracę każdego z członków zespołu:

1. Rafał Kulaszewicz:

- komunikacja z ESP za pomocą JSON,
- routing aplikacji,
- baza danych oraz łączenie z aplikacją,
- wysyłanie wiadomości SMS w celu potwierdzenia rejestracji,
- autoryzacja konta użytkownika,
- dodawanie i walidacja dodawanego urządzenia,
- zaprogramowanie mikrokontrolera ESP8266,
- połączenie przekaźnika z listwą zasilającą oraz ESP8266,

2. Ewelina Mazur:

- rejestracja użytkowników,
- logowanie i wylogowywanie z kont,
- szyfrowanie, resetowanie oraz zmiana hasła,
- wygląd i funkcjonalność aplikacji,

3. Maciej Marzec:

- wygląd aplikacji,
- przechodzenie między widokami,
- statyczna baza danych,

- logowanie do panelu aplikacji,
- łączenie się z serwerem urządzenia,
- zmiana stanów naszego urządzenia.

.....

data

.....

podpis