

Politechnika Warszawska

WYDZIAŁ INŻYNIERII PRODUKCJI



# Praca przejściowa inżynierska

na kierunku Automatykacja i Robotyzacja Procesów Produkcyjnych

**Komunikacja serwera z modułem mikrokontrolera  
ESP8266 oraz czujnika RFID poprzez Wi-Fi.**

**Błażej Chmielewski**

Numer albumu 312565

## Spis treści

1. Cel i zakres pracy .....	3
2. Wiadomości wstępne .....	3
2.1 Mikrokontroler ESP8266.....	3
2.1 Czujnik RFID RC522.....	4
2.2 Serwer.....	5
2.3 Model MVC.....	7
3. Wykorzystane elementy .....	8
3.1 Połączenie elementów z płytką ESP8266 .....	8
4. Oprogramowanie użyte do połączenia klienta z serwerem .....	10
4.1 Oprogramowanie po stronie klienta.....	10
4.2 Oprogramowanie po stronie serwera.....	12
4.2.1 Pom.xml.....	13
4.2.2 application.properties .....	15
4.2.3.1 Klasy encji .....	15
4.2.3.2 Repozytoria.....	18
4.2.3.3 Serwisy.....	19
4.2.3.3 Kontrolery.....	21
4.2.3.4 Widoki HMTL .....	24
5. Praca aplikacji .....	32
6. Podsumowanie, obserwacje, wnioski oraz zalety i wady .....	36
6.1 Podsumowanie .....	36
6.2 Obserwacje .....	36
6.3 Wnioski .....	36
6.4 Zalety i Wady .....	37
6.4.1 Zalety .....	37
6.4.2 Wady.....	37
7. Bibliografia.....	37
8. Spis Ilustracji .....	38

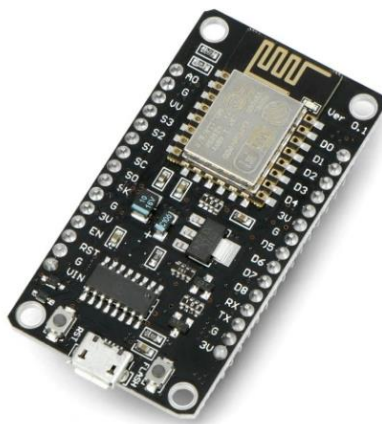
## 1. Cel i zakres pracy

Celem pracy jest praktyczne wykorzystanie protokołu http do przekazywania informacji z płytki mikrokontrolera na serwer. Client wysyła do serwera żądanie POST gdzie przekazuje kod otrzymany z czujnika ówczśnie otrzymany z czujnika RFID i odbicia karty magnetycznej. Zadaniem serwera jest odebranie żądań http od klienta oraz ich przetworzenie. W naszym przypadku został stworzony interfejs użytkownika umożliwiający przypisywanie kodów z karty magnetycznej do konkretnych użytkowników.

## 2. Wiadomości wstępne

### 2.1 Mikrokontroler ESP8266

ESP8266 to niewielki mikrokontroler z wbudowanym modulem Wi-Fi, który jest szeroko wykorzystywany w projektach IoT (Internet of Things) i prototypowaniu elektroniki. Jest produkowany przez firmę Espressif Systems i zyskał popularność ze względu na swoją niską cenę, niskie zużycie energii i możliwość łatwego połączenia z siecią Wi-Fi.



1. Mikrokontroler ESP8266

Oto kilka cech charakterystycznych dla ESP8266:

Wbudowany moduł Wi-Fi: ESP8266 posiada wbudowany moduł Wi-Fi, co pozwala na łatwe łączenie się z siecią bezprzewodową, przesyłanie danych przez Internet i komunikację z innymi urządzeniami w sieci.

- Niski koszt: ESP8266 jest bardzo dostępny cenowo, co sprawia, że jest popularnym wyborem dla projektów DIY i komercyjnych.

- Niska moc: Ten mikrokontroler zużywa niewiele energii, co jest korzystne w zastosowaniach bateryjnych i zasilanych z ogniw słonecznych.
- Wsparcie dla różnych języków programowania: Możesz programować ESP8266 przy użyciu różnych języków, takich jak Arduino (C/C++), MicroPython czy Lua.
- Duża społeczność: ESP8266 ma dużą społeczność programistyczną i wiele dostępnych bibliotek oraz narzędzi, co ułatwia rozwijanie projektów.
- Obsługa GPIO: Posiada wiele pinów GPIO (General-Purpose Input/Output), co pozwala na podłączanie różnych czujników, wyświetlaczy i innych urządzeń.
- Wsparcie dla OTA (Over-The-Air): Wersje ESP8266 pozwalają na aktualizację oprogramowania przez sieć Wi-Fi, co jest przydatne w zastosowaniach, gdzie aktualizacje oprogramowania są konieczne.
- ESP8266 jest wykorzystywany w różnych projektach, takich jak inteligentne domy, monitorowanie środowiska, zdalne sterowanie urządzeniami, automatyzacja, czujniki IoT i wiele innych. Dzięki jego wszechstronności i dostępności, jest to popularny wybór dla wielu twórców projektów elektronicznych.

## 2.1 Czujnik RFID RC522

Czujnik RFID (Radio-Frequency Identification) jest urządzeniem, które służy do identyfikacji i śledzenia przedmiotów, zwierząt lub ludzi za pomocą technologii radiowej. Działa na zasadzie komunikacji bezprzewodowej między czytnikiem (lub skanerem) RFID a tagiem RFID, który jest przymocowany do identyfikowanego obiektu lub osoby. Oto ogólny sposób działania czujnika RFID:



2. Czujnik RFID RC522

- Tag RFID: Tag RFID (czasami nazywany też etykietą lub kartą RFID) to małe urządzenie, które zawiera unikalny identyfikator (numer seryjny) oraz antenę. Tag RFID może być pasywny (bez baterii) lub aktywny (z wbudowaną baterią). Pasywne tagi czerpią energię do działania z sygnału radiowego wysyłanego przez czytnik RFID, podczas gdy aktywne tagi posiadają własne źródło zasilania i mogą działać na większe odległości.
- Czytnik RFID: Czytnik RFID to urządzenie, które generuje sygnał radiowy i wysyła go w kierunku taga RFID. Ten sygnał zasilający pozwala tagowi odpowiedzieć na sygnał i przelać swoje dane identyfikacyjne z powrotem do czytnika.
- Komunikacja radiowa: Gdy tag RFID otrzyma sygnał radiowy od czytnika, odpowiedź taga zawierająca jego unikalny identyfikator jest wysyłana z powrotem do czytnika. Ta komunikacja radiowa odbywa się na określonym zakresie częstotliwości radiowych, który zależy od rodzaju i regionu, w którym jest używany system RFID.
- Odczyt i przetwarzanie danych: Czytnik RFID odbiera dane z taga RFID i przetwarza je, aby zidentyfikować tag i wykorzystać jego dane w systemie. Te dane mogą być przesyłane do komputera lub systemu sterowania w celu dalszego przetwarzania lub śledzenia.
- Aplikacje: Technologia RFID znajduje zastosowanie w wielu dziedzinach, takich jak zarządzanie magazynem, śledzenie produktów w łańcuchu dostaw, kontrole dostępu, zarządzanie zapasami w sklepach, kontrole biletów i wiele innych. Możliwości zastosowań RFID są szerokie i zależą od konkretnych potrzeb i branży.

Czujniki RFID są używane ze względu na swoją niezawodność, szybkość odczytu, zdolność do obsługi wielu tagów jednocześnie oraz bezdotykową naturę identyfikacji, co oznacza, że tagi nie muszą być widoczne ani w zasięgu wzroku, aby być odczytane.

## 2.2 Serwer

W przypadku tego projektu rolę serwera będzie pełnić komputer Acer Nitro5. Urządzenie to jest zdecydowanie wystarczające do autonomicznej pracy jednocześnie pełniąc rolę serwera w tle. Parametry urządzenia:

- Windows 10 Home 64 bitów.
- Procesor AMD Ryzen™ 7 6800H 8-rdzeniowy 3.20 GHz.
- NVIDIA® GeForce RTX™ 3070Ti z 8 GB dedykowanej pamięci.
- 39.6 cm (15.6") QHD (2560 x 1440) 16:9 165 Hz.
- 16 GB, DDR5 SDRAM.
- 1 TB SSD.

Aby odbierać żądania od płytki należało napisać kod wykonujący działania na protokole http. Do tego celu zostały użyte innowacyjne rozwiązania jakie są wykorzystywane w komercyjnych projektach.

Użyte technologie:

- **Java:** Java jest językiem programowania o dużej popularności i rozbudowanych możliwościach. Jest często wykorzystywana do tworzenia aplikacji webowych, a także aplikacji desktopowych i mobilnych. W połączeniu z frameworkiem Spring Boot staje się potężnym narzędziem do tworzenia aplikacji webowych.
- **Spring Boot:** Spring Boot to framework aplikacji Java, który ułatwia tworzenie aplikacji webowych i mikrouslug. Jest on często używany do budowy serwerów RESTful, zarządzania danymi i kontrolerów oraz obsługi żądań HTTP. Spring Boot zapewnia wiele funkcji ułatwiających konfigurację i uruchamianie aplikacji, co pozwala programistom skupić się na implementacji funkcjonalności.
- **Thymeleaf:** Thymeleaf jest silnikiem szablonów HTML przeznaczonym do tworzenia stron internetowych w aplikacjach webowych. Jego główną cechą jest to, że jest to szablon silnika, który umożliwia wbudowywanie dynamicznych danych bezpośrednio w kodzie HTML. Thymeleaf jest często używany w połączeniu z Spring Boot do renderowania dynamicznych widoków w aplikacjach internetowych.
- **MySQL:** MySQL to system zarządzania relacyjnymi bazami danych (RDBMS), który pozwala na składowanie, zarządzanie i dostęp do danych w sposób efektywny.

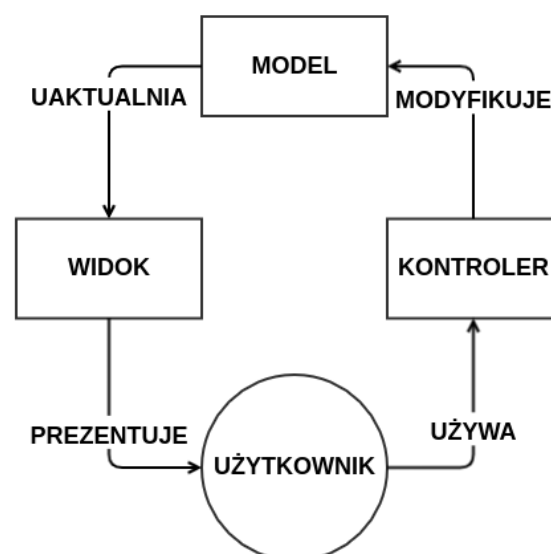
Jak działają razem:

- Spring Boot działa jako serwer aplikacyjny, obsługując żądania HTTP i zarządzając przepływem aplikacji.
- Thymeleaf jest używany do tworzenia widoków HTML w aplikacji. Możesz umieszczać zmienne i wyrażenia Thymeleaf w kodzie HTML, które zostaną dynamicznie wypełnione danymi z kontrolera Spring Boot.
- Kontrolery Spring Boot obsługują żądania HTTP i decydują, jakie dane należy przekazać do widoków Thymeleaf.
- Po przetworzeniu żądania przez kontroler, Thymeleaf generuje HTML na podstawie szablonów HTML zawierających zmienne i wyrażenia Thymeleaf. Te zmienne i wyrażenia są wypełniane danymi z kontrolera.
- Wygenerowany HTML jest następnie przesyłany jako odpowiedź na żądanie HTTP i wyświetlany w przeglądarce internetowej użytkownika.

## 2.3 Model MVC

Model-View-Controller (MVC) to wzorec projektowy stosowany w inżynierii oprogramowania, który pomaga w organizacji i rozdzieleniu różnych aspektów aplikacji, takich jak dane, interfejs użytkownika i logika biznesowa. Wzorec ten ma na celu poprawienie modularności, skalowalności i zarządzalności aplikacji. W MVC występują trzy główne komponenty:

- **Model (M):** Model reprezentuje dane i logikę biznesową aplikacji. Odpowiada za przechowywanie, przetwarzanie i dostarczanie danych do aplikacji. Model jest niezależny od warstwy widoku i kontrolera, co oznacza, że zmiany w modelu nie wpływają bezpośrednio na wygląd ani zachowanie interfejsu użytkownika. Model może zawierać operacje CRUD (Create, Read, Update, Delete) do manipulowania danymi.
- **Widok (V):** Widok jest odpowiedzialny za prezentację danych użytkownikowi i zbieranie od niego interakcji. To element interfejsu użytkownika, który wyświetla dane z modelu i umożliwia użytkownikowi wprowadzanie danych. Widok jest pasywny i nie zawiera logiki biznesowej. W przypadku aplikacji webowych, widok to często strona HTML, która wykorzystuje szablony do wygenerowania treści.
- **Kontroler (C):** Kontroler pełni rolę pośrednika między modelem a widokiem. Odpowiada za obsługę żądań użytkownika, przekazywanie tych żądań do modelu, aktualizację modelu w odpowiedzi na żądania i renderowanie odpowiedniego widoku. Kontroler jest odpowiedzialny za zarządzanie przepływem danych między modelem a widokiem, ale nie zawiera bezpośrednio logiki biznesowej ani związanej z interfejsem użytkownika.



3. Model MVC

Podstawową idea wzorca MVC polega na tym, że każdy z tych trzech komponentów jest oddzielony od pozostałych, co ułatwia rozwijanie, testowanie i utrzymanie aplikacji. Zmiana w jednym z komponentów nie wpływa bezpośrednio na pozostałe. MVC jest szczególnie przydatne w aplikacjach, które wymagają łatwej modyfikowalności interfejsu użytkownika lub które muszą obsługiwać różne rodzaje interfejsów (np. aplikacje webowe i mobilne), ponieważ możesz ponownie wykorzystać ten sam model biznesowy dla różnych widoków i kontrolerów.

Warto również zaznaczyć, że istnieje wiele wariantów wzorca MVC, takie jak Model-View-Presenter (MVP) i Model-View-ViewModel (MVVM), które wprowadzają pewne modyfikacje i dostosowania do konkretnych potrzeb i technologii.

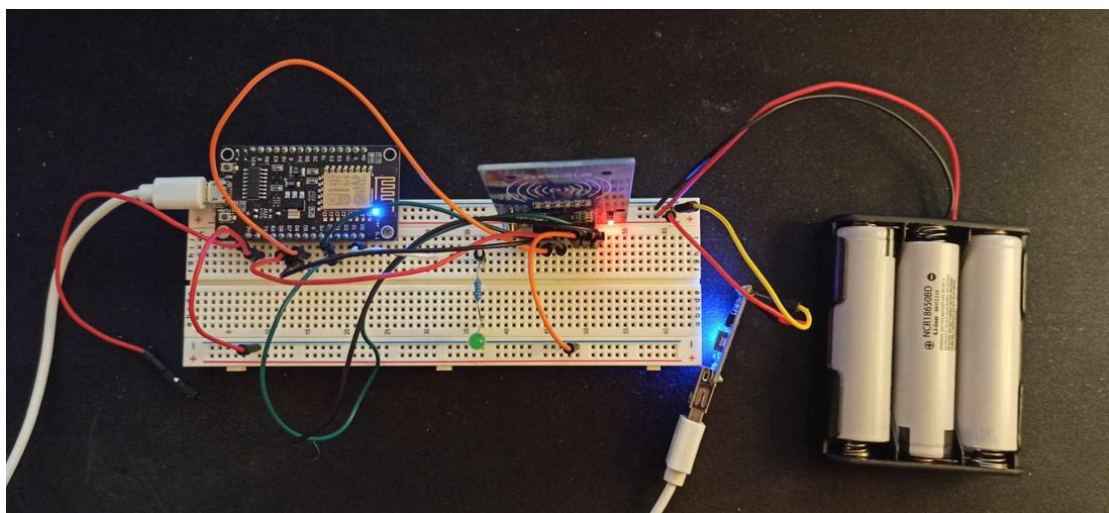
Dla uproszczenia i zwiększenia czytelności kodu w projekcie został również zaimplementowany pośrednik w postaci serwisu. Serwis to nic innego jak zbiór wszystkich metod jakie są wykonywane na obiektach. W klasie kontrolera istnieje tylko instancja serwisu i na potrzeby danych metod są wywoływane odpowiednie akcje z serwisu. Pozwala to zachować czytelność klasy kontrolera ponieważ znajduje się w nim tylko obsługa żądań http a nie ma tam rozbudowanej logiki biznesowej.

### 3. Wykorzystane elementy

#### 3.1 Połączenie elementów z płytą ESP8266

Poniżej znajdują się zdjęcie połączonych układów:

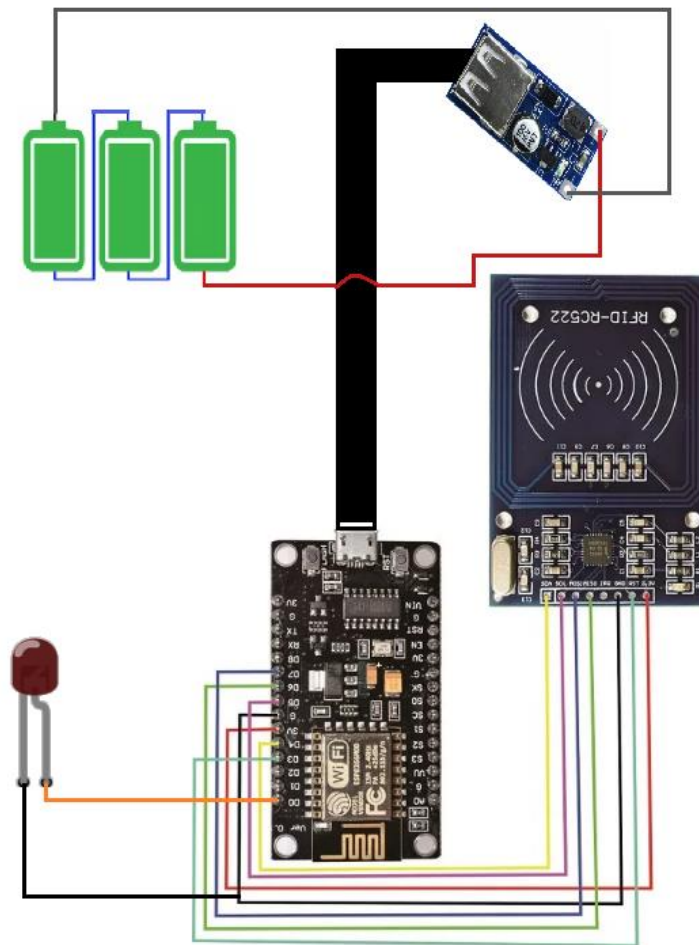
- Płytki ESP8266
- Czujnika RFID RC522
- Diody LED
- Konwertera napięcia (12V --> 5V)
- Zasilanie 3 ogniwami połączonymi szeregowo (razem ~11.4V)



4. Fizyczny połączenie układu płytki



Dla przejrzystości poniżej przedstawię schemat układu.



5. Schemat przedstawiający połączenie układu

W następnej kolejności przedstawię jak prezentują się podpięcie wejść cyfrowych na płytce.

ESP8266	RFID-RC522
D4(GPIO2)	SDA
D5(GPIO14)	SCK
D7(GPIO13)	MOSI
D6(GPIO12)	MISO
-	IRQ
GND	GND
D3	RST
3V(GPIO0)	3.3V
LED	
D0(GPIO16)	LED

6. Połączenie pinów cyfrowych

## 4. Oprogramowanie użyte do połączenia klienta z serwerem

### 4.1 Oprogramowanie po stronie klienta

```
#include <SPI.h>
#include <MFRC522.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

// Piny dla modułu MFRC522
constexpr uint8_t RST_PIN = D3;
constexpr uint8_t SS_PIN = D4;

MFRC522 rfid(SS_PIN, RST_PIN); // Instancja klasy
MFRC522::MIFARE_Key key;
String tag;

// Dane sieci Wi-Fi
const char* ssid = "####"; // podstawić ssid sieci
const char* password = "####"; // podstawić hasło sieci

// Adres serwera HTTP
const char* serverAddress = "http://192.168.0.8:8080/rfid"; // domyślny end point serwera

void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi...");
    }

    SPI.begin(); // Inicjalizacja magistrali SPI
    rfid.PCD_Init(); // Inicjalizacja modułu MFRC522
}

void loop() {
    if (WiFi.status() == WL_CONNECTED) {
        if (!rfid.PICC_IsNewCardPresent())
            return;
        if (rfid.PICC_ReadCardSerial()) {
            for (byte i = 0; i < 4; i++) {
                tag += rfid.uid.uidByte[i];
            }

            Serial.println(tag);
            sendTagToServer(tag);
            tag = "";
            rfid.PICC_HaltA();
            rfid.PCD_StopCrypto1();
        }
    }
}

void sendTagToServer(String tag) {
    HTTPClient http;
    WiFiClient client; // Używamy klienta WiFi jako argumentu w metodzie begin

    http.begin(client, serverAddress);
    http.addHeader("Content-Type", "text/plain");

    int httpResponseCode = http.POST(tag);

    if (httpResponseCode > 0) {
        String response = http.getString();
        Serial.println("Response code: " + String(httpResponseCode));
        Serial.println("Response: " + response);
    } else {
        Serial.println("Error sending POST request");
    }

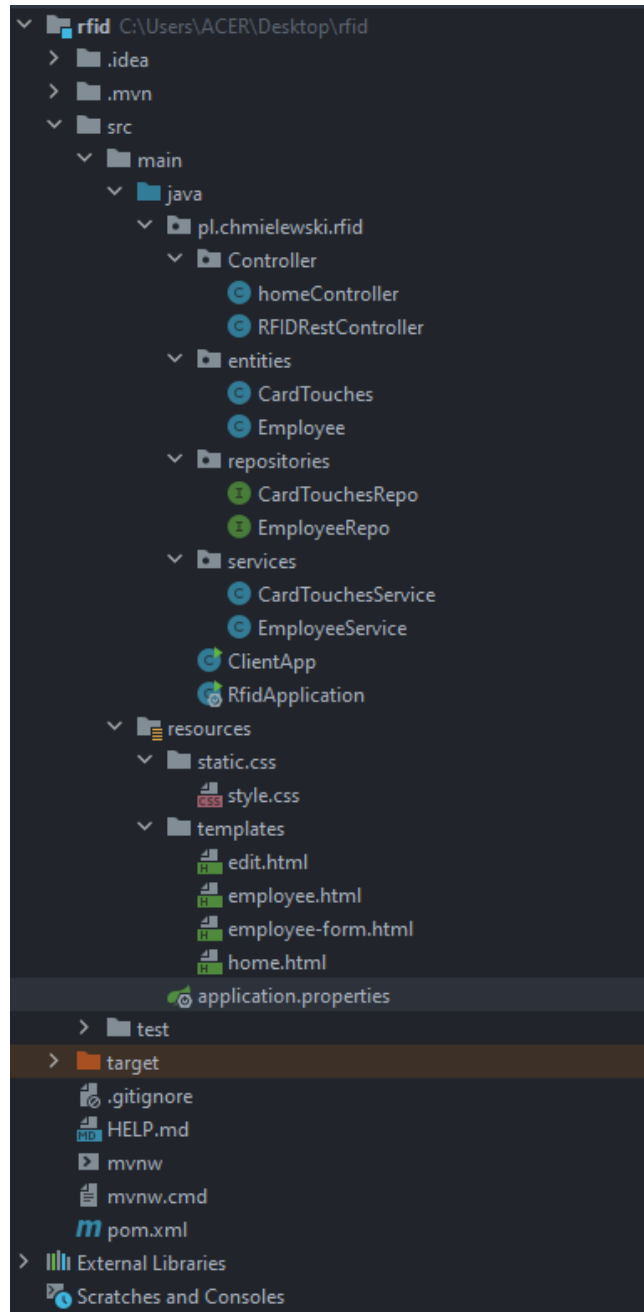
    http.end();
}
```

Poniżej opis poszczególnych czynności jakie się dzieją w kodzie

- W pierwszych kilku liniach kodu importowane są biblioteki, takie jak SPI, MFRC522, ESP8266WiFi i ESP8266HTTPClient, które są potrzebne do obsługi modułu RFID oraz komunikacji przez Wi-Fi.
- Określane są piny, na których podłączone są interfejsy SPI i moduł RFID.
- Inicjalizowana jest instancja klasy MFRC522 oraz obiekt klasy MIFARE\_Key do obsługi modułu RFID.
- Definiowane są dane sieci Wi-Fi (SSID i hasło) oraz adres serwera HTTP, na który będą wysyłane dane RFID.
- W funkcji setup() inicjalizowane są UART (dla komunikacji szeregowej z monitorem szeregowym), połączenie Wi-Fi oraz magistrala SPI i moduł RFID.
- W funkcji loop() program sprawdza, czy ESP8266 jest połączony z siecią Wi-Fi. Następnie monitoruje obecność nowej karty RFID i czyta jej identyfikator, a następnie wysyła go na serwer HTTP.
- Funkcja sendTagToServer() odpowiada za wysłanie identyfikatora na serwer HTTP. Tworzy ona klienta HTTP i wysyła POST request z identyfikatorem do określonego serwera. Odpowiedź od serwera jest wyświetlana w monitorze szeregowym.
- Ten kod jest przykładem zastosowania ESP8266 do odczytywania danych z modułu RFID i przesyłania ich przez Wi-Fi na serwer HTTP, co może być użyteczne w wielu zastosowaniach, takich jak kontrola dostępu, monitorowanie obiektów lub rejestracja czasu pracy.

## 4.2 Oprogramowanie po stronie serwera

Poniżej zostało pokazane drzewko projektu z poszczególnymi klasami, plikami konfiguracyjnymi. W tym miejscu zaznaczam że jako budowniczy projektu wybrałem Maven dlatego drzewko trochę się różni od klasycznego schematu projektu Javy.



7. Drzewo projektu serwera

- 8 Klas
- 2 Repozytoria
- 4 pliki Html
- 1 Plik Css
- Plik konfiguracyjny(application.properties)
- Plik pom.xml(do pobierania zależności)

### 4.2.1 Pom.xml

Plik pom.xml jest kluczowym elementem projektów opartych na narzędziu Apache Maven, które jest popularnym narzędziem do zarządzania zależnościami, budowania projektów i zarządzania cyklem życia projektu w języku Java (oraz innych językach). Plik pom.xml zawiera konfigurację projektu i jest używany przez Maven do automatycznego zarządzania projektami.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3.      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
    4.0.0.xsd">
4.      <modelVersion>4.0.0</modelVersion>
5.      <parent>
6.          <groupId>org.springframework.boot</groupId>
7.          <artifactId>spring-boot-starter-parent</artifactId>
8.          <version>2.7.15</version>
9.          <relativePath/> <!-- lookup parent from repository -->
10.     </parent>
11.     <groupId>pl.chmielewski</groupId>
12.     <artifactId>rfid</artifactId>
13.     <version>0.0.1-SNAPSHOT</version>
14.     <name>rfid</name>
15.     <description>Demo project for Spring Boot</description>
16.     <properties>
17.         <java.version>17</java.version>
18.     </properties>
19.     <dependencies>
20.         <dependency>
21.             <groupId>org.springframework.boot</groupId>
22.             <artifactId>spring-boot-starter-data-jpa</artifactId>
23.         </dependency>
24.         <dependency>
25.             <groupId>org.springframework.boot</groupId>
26.             <artifactId>spring-boot-starter-thymeleaf</artifactId>
27.         </dependency>
28.         <dependency>
29.             <groupId>org.springframework.boot</groupId>
30.             <artifactId>spring-boot-starter-web</artifactId>
31.         </dependency>
32.         <dependency>
33.             <groupId>org.mybatis.spring.boot</groupId>
34.             <artifactId>mybatis-spring-boot-starter</artifactId>
35.             <version>2.3.1</version>
36.         </dependency>
37.
38.     <dependency>
```

```

39.         <groupId>com.mysql</groupId>
40.         <artifactId>mysql-connector-j</artifactId>
41.         <scope>runtime</scope>
42.     </dependency>
43.
44.     <dependency>
45.         <groupId>org.springframework.boot</groupId>
46.         <artifactId>spring-boot-starter-test</artifactId>
47.         <scope>test</scope>
48.     </dependency>
49.     <dependency>
50.         <groupId>org.mybatis.spring.boot</groupId>
51.         <artifactId>mybatis-spring-boot-starter-test</artifactId>
52.         <version>2.3.1</version>
53.         <scope>test</scope>
54.     </dependency>
55. </dependencies>
56.
57.     <build>
58.         <plugins>
59.             <plugin>
60.                 <groupId>org.springframework.boot</groupId>
61.                 <artifactId>spring-boot-maven-plugin</artifactId>
62.             </plugin>
63.         </plugins>
64.     </build>
65.
66. </project>

```

Wgrane zależności :

- Spring Data JPA
- MySQL Driver
- Spring Web
- Thymeleaf

### 4.2.2 application.properties

W tym pliku znajdują się połączenie do bazy danych. Przekazujemy tutaj lokalizację serwera bazy danych, w naszym przypadku jest on na localhost, porcie 3306 oraz domyślny schemat to rfid. Zostały ustawione parametry nazwy użytkownika oraz hasło, wskazana klasa sterownika bazy danych oraz strategia tworzenia bazy w tym przypadku update. Ta strategia pozwala nam nie tracić danych po zresetowaniu aplikacji

```
# Konfiguracja bazy danych MySQL
spring.datasource.url=jdbc:mysql://localhost:3306/rfid
spring.datasource.username=root
spring.datasource.password=####
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto=update
```

### 4.2.3.1 Klasy encji

W języku Java, klasy są jednym z fundamentalnych koncepcji programowania obiektowego. Klasy stanowią szablon lub wzorzec, na podstawie którego tworzone są obiekty, które są instancjami danej klasy. Poniżej przedstawione zostały w pierwszej kolejności klasy encji:

#### Encja „Employee”

Konceptualnie jest to encja przechowująca pola opisujące pracownika. Mamy tutaj pole z imieniem, nazwiskiem, dniem i godziną utworzenia pracownika oraz pole z przypisanym kodem z karty magnetycznej.

```
package pl.chmielewski.rfid.entities;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String firstName;
    private String lastName;
    private String code;
    private String creationDate;

    public Employee() {
    }

    public Long getId() {
        return id;
    }
}
```

```

    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getCreationDate() {
        return creationDate;
    }

    public void setCreationDate(String creationDate) {
        this.creationDate = creationDate;
    }
}

```

## Encja „CardTouches”

W tej encji mamy pola id, pola z przechowywanym kodem z czujnika oraz data i czasem odbicia karty.

```

1 package pl.chmielewski.rfid.entities;
2
3 import org.springframework.lang.Nullable;
4
5 import javax.persistence.Entity;
6 import javax.persistence.GeneratedValue;
7 import javax.persistence.Id;
8
9 @Entity
10 public class CardTouches {
11

```



```

12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private Long id;
15
16     private String code;
17     private String dateTime;
18
19     @Nullable
20     private String employeeId;
21
22
23     public CardTouches() {
24     }
25
26     public Long getId() {
27         return id;
28     }
29
30     public void setId(Long id) {
31         this.id = id;
32     }
33
34     public String getCode() {
35         return code;
36     }
37
38     public void setCode(String code) {
39         this.code = code;
40     }
41
42     public String getDateTime() {
43         return dateTime;
44     }
45
46     public void setDateTime(String dateTime) {
47         this.dateTime = dateTime;
48     }
49
50     public String getEmployeeId() {
51         return employeeId;
52     }
53
54     public void setEmployeeId(String employeeId) {
55         this.employeeId = employeeId;
56     }
57 }

```

Tworzenie encji z frameworkiem springboot z pozwala nam na znaczne przyspieszenie tworzenia tabel w bazie danych. Jedyne co trzeba zrobić to użyć odpowiednich notacji i pewnych metod:

**@Entity** – ta notacja tworzy z klasy encję czyli w danym języku programowania nie jawnie mamy konwertowanie kodu do kodu SQL tworzącego tabele i zarządzającego nimi

**@Id** – Wskazuje nam na klucz główny tabeli (w sql: PRIMATY KEY)

**@GeneratedValue** – Ta notacja pozwala nam określić styl jakim chcemy tworzyć klucz główny. W naszych encjach w obydwu przypadkach mamy strategię IDENTITY czyli klucz będzie tworzony inkrementalnie oraz gdy któryś rekord po drodze zostanie usunięty to ciąg idzie dalej i się nie cofa „zalepiając” lukę w rekordach.

**Gettery i Settery** – Te metody są konieczne do działania programy, springboot na ich podstawie odwołuje się do pól i pozwala przypisać bądź pobrać wartość.

**Bezparametrowy Konstruktor** – Większość frameworków w których tworzymy encje wymaga dostępu do tego konstruktora. Jest używany przez mechanizmy serializacji/deserializacji, a także przy niektórych operacjach refleksji.

#### 4.2.3.2 Repozytoria

Repozytoria są to „arsenały” z metodami bez implementacji gotowe do użycia w danej klasie. Używa się ich w momencie mamy podobne metody lekko różniące się implementacją. W kontekście Java Spring, repozytoria są często używane do dostępu do danych w bazie danych. Spring Framework dostarcza narzędzi do tworzenia repozytoriów w sposób bardziej efektywny i ułatwiający pracę z bazą danych.

Repozytoria w Spring można tworzyć przy użyciu technologii Spring Data, a w szczególności Spring Data JPA i Spring Data JDBC. Oto kilka informacji na ten temat:

#### Repozytorium CardTouchesRepo

```
1 package pl.chmielewski.rfid.repositories;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5 import pl.chmielewski.rfid.entities.CardTouches;
6
7 @Repository
8 public interface CardTouchesRepo extends JpaRepository<CardTouches, Long> {
9 }
```

#### Repozytorium EmployeeRepo

```
1 package pl.chmielewski.rfid.repositories;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5 import pl.chmielewski.rfid.entities.Employee;
6
7 import java.util.Optional;
8
9 @Repository
10 public interface EmployeeRepo extends JpaRepository<Employee, Long> {
11
12     Optional<Employee> findByCode(String code);
13 }
```

Do tworzenia repozytorium wystarczy dodanie nam notacji `@Repository` oraz rozszerzenie istniejącego repozytorium `Jpa` dostarczającego nam gotowe implementacje metod najczęściej używanych na bazach danych, tak zwanych CRUD.

Wygląda to tak że `Jpa` jest gotowym systemem nazewnictwa metod, potrzebnym aby się porozumieć z bazą danych, natomiast przy poszerzeniu `JpaRepository` otrzymujemy również implementacje tych metod w ramach tak zwanego Hibernate.

Hibernate jest to po prostu najpopularniejsza implementacja `Jpa`. Pozwala nam ona na wykonanie intuicyjnych metod na obiektach encji bez jakiegokolwiek pisania kodu SQL. Przykładowo możemy wykonać akcję: `findById(Long id)`, gdzie intuicyjnie rozumiemy że hibernate znajdzie nam element po `id` podspodem wykonując kod: „`SELECT * FROM Table WHERE id = id`”). To rozwiązanie znacznie przyspiesza wykonywanie typowych poleceń CRUD. W tym miejscu można by się zastanowić nad wydajnością samego hibernate’a. Dobrze radzi on sobie w prostych zapytaniach do bazy danych lecz przy wykonywaniu procedur, joinowania i bardziej skomplikowanych operacji zaleca się korzystanie z `JdbcTemplate`, czyli klasy która pozwala nam przekazanie kodu SQL z aplikacji backendowej do serwera bazy danych.

#### 4.2.3.3 Serwisy

Serwisy to klasy z zawartą logiką biznesową. W przypadku projektów Java Springboot opartych na modelu MVC mamy tutaj operacje na bazie danych korzystając z instancji repozytorium utworzonego dla poszczególnych encji. Na owej instancji możemy wykonywać już funkcje jakie nam dostarcza hibernate. Aby klasa była brana pod uwagę w kontenerze springa musi mieć adnotację `@Service`.

Poniżej znajdują się utworzone serwisy z zestawem funkcji potrzebnych w kontrolerze. Funkcje są samo wyjaśniające się także pomijam ich opis.

#### Serwis CardTouchesService

```
1 package pl.chmielewski.rfid.services;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import pl.chmielewski.rfid.entities.CardTouches;
6 import pl.chmielewski.rfid.repositories.CardTouchesRepo;
7
8 import java.time.LocalDateTime;
9 import java.time.format.DateTimeFormatter;
10 import java.util.List;
11
12 @Service
13 public class CardTouchesService {
14
15     private CardTouchesRepo cardTouchesRepo;
16     private EmployeeService employeeService;
17
18     @Autowired
```

```

19     public CardTouchesService(CardTouchesRepo cardTouchesRepo, EmployeeService
20 employeeService) {
21         this.cardTouchesRepo = cardTouchesRepo;
22         this.employeeService = employeeService;
23     }
24
25     public void createNewTouch(String code){
26         CardTouches cardTouches = new CardTouches();
27         cardTouches.setCode(code);
28
29 cardTouches.setEmployeeId(String.valueOf(employeeService.getEmployeeIdByCode(code)));
30         LocalDateTime localDateTime = LocalDateTime.now();
31         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
32         String formattedDateTime = localDateTime.format(formatter);
33         cardTouches.setDateTime(formattedDateTime);
34         cardTouchesRepo.save(cardTouches);
35     }
36
37     public List<CardTouches> getCardTouchesList() {
38         return cardTouchesRepo.findAll();
39     }
40 }

```

## Serwis EmployeeService

```

1 package pl.chmielewski.rfid.services;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import pl.chmielewski.rfid.entities.Employee;
6 import pl.chmielewski.rfid.repositories.EmployeeRepo;
7
8 import java.time.LocalDateTime;
9 import java.time.format.DateTimeFormatter;
10 import java.util.ArrayList;
11 import java.util.List;
12 import java.util.stream.Collectors;
13
14 @Service
15 public class EmployeeService {
16
17     private EmployeeRepo employeeRepo;
18
19     @Autowired
20     public EmployeeService(EmployeeRepo employeeRepo) {
21         this.employeeRepo = employeeRepo;
22     }
23
24     private List<String> codes = new ArrayList<>();
25
26     public EmployeeService() {
27         codes.add("Brak");
28     }
29
30     public Employee getEmployeeById(Long id) {
31         return employeeRepo.findById(id).orElse(null);
32     }
33 }

```

```

32     }
33
34     public Long getEmployeeIdByCode(String code) {
35
36         Employee byCode = employeeRepo.findByCode(code).orElse(null);
37         assert byCode != null;
38         return byCode.getId();
39     }
40
41     public void addNewEmployee(Employee employee) {
42         LocalDateTime localDateTime = LocalDateTime.now();
43         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
44         String formattedDateTime = localDateTime.format(formatter);
45         employee.setCreationDate(formattedDateTime);
46         employeeRepo.save(employee);
47     }
48
49     public void addNewCode(String code) {
50         codes.add(code);
51     }
52
53     public List<String> getAllCodes() {
54         return codes.stream().distinct().collect(Collectors.toList());
55     }
56
57     public void edit(Employee employee) {
58         employeeRepo.save(employee);
59     }
60
61     public List<Employee> getAllEmployees() {
62         return employeeRepo.findAll();
63     }
64 }

```

#### 4.2.3.3 Kontrolery

Do wysyłania w postaci RESTAPI kodów odebranych przez mikrokontroler służy nam klasa kontrolera.

#### Kontroler RFIDRestController

```

1 package pl.chmielewski.rfid.Controller;
2
3 import org.springframework.http.HttpStatus;
4 import org.springframework.http.ResponseEntity;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PostMapping;
7 import org.springframework.web.bind.annotation.RequestBody;
8 import org.springframework.web.bind.annotation.RestController;
9
10 @RestController
11 public class RFIDRestController {
12
13     private String lastReceivedRFID = "";
14
15     private HomeController homeController;

```

```

16
17     public RFIDRestController(pl.chmielewski.rfid.Controller.homeController homeController) {
18         this.homeController = homeController;
19     }
20
21     @PostMapping("/rfid")
22     public ResponseEntity<String> receiveRFIDData(@RequestBody String rfidData) {
23         try {
24             System.out.println("Odebrano kod RFID: " + rfidData);
25
26             lastReceivedRFID = rfidData;
27             homeController.sendData(rfidData);
28             return new ResponseEntity<>("Odebrano kod RFID: " + rfidData, HttpStatus.OK);
29         } catch (Exception e) {
30             return new ResponseEntity<>("Błąd podczas przetwarzania kodu RFID.",
31 HttpStatus.INTERNAL_SERVER_ERROR);
32         }
33     }
34
35     @GetMapping("/lastRFID")
36     public String getHome() {
37         return lastReceivedRFID;
38     }
39 }

```

W przedstawionym kodzie, kontroler REST napisany w Spring Boot obsługuje żądania HTTP POST i GET związane z danymi RFID. Kontroler odbiera dane RFID, zapisuje je i może je przekazać do innych komponentów aplikacji. Ostatnio otrzymany kod RFID jest przechowywany w zmiennej `lastReceivedRFID`, a dostęp do niego można uzyskać poprzez żądanie GET na `/lastRFID`.

### Kontroler homeController

Ten kod to kontroler Spring Boot obsługujący różne żądania HTTP związane z zarządzaniem danymi pracowników i operacjami związanymi z kodami RFID.

```

1 package pl.chmielewski.rfid.Controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.ui.Model;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.ModelAttribute;
8 import org.springframework.web.bind.annotation.PathVariable;
9 import org.springframework.web.bind.annotation.PostMapping;
10 import pl.chmielewski.rfid.entities.Employee;
11 import pl.chmielewski.rfid.services.CardTouchesService;
12 import pl.chmielewski.rfid.services.EmployeeService;
13
14
15 @Controller
16 public class HomeController {
17
18     private EmployeeService employeeService;
19     private CardTouchesService cardTouchesService;

```

```

20
21     @Autowired
22     public HomeController(EmployeeService employeeService, CardTouchesService
23 cardTouchesService) {
24         this.employeeService = employeeService;
25         this.cardTouchesService = cardTouchesService;
26     }
27
28     @GetMapping("/home")
29     public String home(Model model) {
30         model.addAttribute("getCardTouchesList", cardTouchesService.getCardTouchesList());
31         return "home";
32     }
33
34     public void sendData(String lastReceivedRFID) {
35         employeeService.addNewCode(lastReceivedRFID);
36         cardTouchesService.createNewTouch(lastReceivedRFID);
37     }
38
39     @PostMapping("/newEmployee")
40     public String newEmployee(@ModelAttribute Employee employee) {
41         employeeService.addNewEmployee(employee);
42         return "redirect:/employee";
43     }
44
45     @GetMapping("/addEmployee")
46     public String getEmployeeForm(Model model) {
47         model.addAttribute("employee", new Employee());
48         return "employee-form";
49     }
50
51     @GetMapping("/employee")
52     public String getEmployeeList(Model model) {
53         model.addAttribute("employees", employeeService.getAllEmployees());
54         return "employee"; // Nazwa szablonu Thymeleaf do wyświetlenia listy pracowników.
55     }
56
57     @GetMapping("/edit/{id}")
58     public String getEmployeeEdit(@PathVariable Long id, Model model) {
59         model.addAttribute("employee", employeeService.getEmployeeById(id));
60         model.addAttribute("Codes", employeeService.getAllCodes());
61         return "edit";
62     }
63
64     @PostMapping("/edit")
65     public String editEmployee(@ModelAttribute("employee") Employee employee, Model model)
66 {
67         employeeService.edit(employee);
68
69         return "redirect:/employee";
70     }
71 }

```

Kontroler obsługuje operacje związane z pracownikami i kodami RFID w aplikacji.

- Metoda `home` obsługuje żądania GET na `/home` i przekazuje listę dotknięć kart RFID do widoku.
- Metoda `sendData` przyjmuje dane RFID i przekazuje je do serwisów `employeeService` i `cardTouchesService`.
- Metoda `newEmployee` obsługuje żądania POST na `/newEmployee` i dodaje nowego pracownika.
- Metoda `getEmployeeForm` obsługuje żądania GET na `/addEmployee` i zwraca formularz do dodawania pracownika.
- Metoda `getEmployeeList` obsługuje żądania GET na `/employee` i zwraca listę pracowników.
- Metoda `getEmployeeEdit` obsługuje żądania GET na `/edit/{id}` i zwraca formularz edycji pracownika.
- Metoda `editEmployee` obsługuje żądania POST na `/edit` i aktualizuje dane pracownika.

Kod ten wykorzystuje adnotacje Spring do mapowania ścieżek URL na metody obsługi i wstrzykiwania zależności (dependency injection) poprzez adnotacje `@Autowired`.

#### 4.2.3.4 Widoki HTML

Poniżej przedstawienie kodu do widoków HTML. Każdy z nich zawiera odpowiednie znaczniki udostępnione przez silnik szablonów `thymeleaf`. Jest to silnik szablonów HTML dla języka Java, który jest często wykorzystywany w aplikacjach opartych na Spring Framework, w tym Spring Boot. Pozwala on na generowanie widoków HTML w sposób dynamiczny, umożliwiając wstawianie danych z modelu aplikacji bezpośrednio do szablonów HTML. Oto kilka informacji na temat generowania widoków HTML w Thymeleaf:

- **Składnia Thymeleaf:** Wprowadzona została specjalna składnia, która pozwala na umieszczanie atrybutów Thymeleaf w kodzie HTML przy użyciu prefiksu `th`. Na przykład: `th:text`, `th:if`, `th:each`, `th:attr`, itp.
- **Wstawianie Danych z Modelu:** W szablonach Thymeleaf można wstawiać dane z modelu (obiektu przekazywanego z kontrolera) za pomocą atrybutów Thymeleaf. Na przykład, aby wstawić nazwę użytkownika, można użyć `th:text`:

```
<p th:text="${user.name}">Imię użytkownika</p>
```

- **Operacje na pętlach i warunki:** Thymeleaf umożliwia tworzenie warunków i pętli bezpośrednio w kodzie HTML. Możesz użyć `th:if` do warunkowego wyświetlania elementów lub `th:each` do iteracji po kolekcjach danych.

**Warunek :** `<div th:if="${user.isAdmin}">To jest administrator</div>`

**Pętla:** `<ul>`

```
<li th:each="product : ${products}" th:text="${product.name}">Nazwa produktu</li>
```

```
</ul>
```



- Formularze i Przekazywanie Danych: Thymeleaf jest przydatny do generowania formularzy HTML i przekazywania danych z formularzy z powrotem do kontrolerów Spring. Możesz użyć atrybutów Thymeleaf, takich jak `th:field` i `th:value`, do tworzenia dynamicznych formularzy.

```
<form th:action="@{/submitForm}" method="post">
```

```
    <input type="text" th:field="*{username}" />
```

```
    <button type="submit">Submit</button>
```

```
</form>
```

### Oto kody HTML dla poszczególnych widoków:

#### 1.home

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <title>RFID Codes</title>
5     <link rel="stylesheet" type="text/css" href="/css/style.css">
6     <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
7     rel="stylesheet">
8
9 </head>
10 <body>
11     <div class="button-container">
12         <a th:href="@{/employee}"><button>Zobacz liste użytkowników</button></a>
13     </div>
14     <div class="title-container"><h3>Odbicia kart przez pracowników</h3></div>
15
16     <table>
17         <thead>
18             <tr>
19                 <th>ID</th>
20                 <th>Code</th>
21                 <th>DateTime</th>
22                 <th>Employee ID</th>
23             </tr>
24         </thead>
25         <tbody>
26
27             <tr th:each="touch : ${getCardTouchesList}">
28                 <td th:text="${touch.getId()}"></td>
29                 <td th:text="${touch.getCode()}"></td>
30                 <td th:text="${touch.getDateTime()}"></td>
31                 <td th:text="${touch.getEmployeeId()}"></td>
32             </tr>
33         </tbody>
34     </table>
35 </body>
36 </html>
```

## 2. employee-form

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Add New Employee</title>
  <link rel="stylesheet" type="text/css" href="/css/style.css">
  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
rel="stylesheet">

</head>
<body>
  <div class="button-container">
    <a th:href="@{/employee}"><button>Zobacz liste użytkowników</button></a>
  </div>
  <div class="title-container">

    <h3>Add New Employee</h3>
  </div>
<div class="form-container">
  <form action="/newEmployee" method="post" th:object="${employee}">
    <div class="inputs"><label for="firstName">First Name:</label></div>
    <div class="inputs"><input type="text" id="firstName" name="firstName"
th:field="*{firstName}" required/><br/></div>
    <div class="inputs"><label for="lastName">Last Name:</label></div>
    <div class="inputs"><input type="text" id="lastName" name="lastName"
th:field="*{lastName}" required/><br/></div>
    <div class="input-submit-container"><input class="input-submit" type="submit"
value="Add Employee"/></div>
  </form>
</div>
</body>
</html>
```

## 3. employee-form

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Employee List</title>
  <link rel="stylesheet" type="text/css" href="/css/style.css">
  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
rel="stylesheet">

</head>
<body>
<div class="button-container">
  <a th:href="@{/home}"><button style="margin-right: 40px;">Przejdź do listy
logów</button></a>
  <a th:href="@{/addEmployee}"><button>Dodaj użytkownika</button></a>
</div>
  <div class="title-container"><h3>Lista użytkowników</h3>
</div>
<table>
  <thead>
```

```

<tr>
  <th>ID</th>
  <th>First Name</th>
  <th>Last Name</th>
  <th>Card Code</th>
  <th>Creation Date</th>
</tr>
</thead>
<tbody>

<tr th:each="employee : ${employees}">
  <td th:text="${employee.id}"></td>
  <td th:text="${employee.firstName}"></td>
  <td th:text="${employee.lastName}"></td>
  <td th:text="${employee.code}"></td>
  <td>
    <div class="edit-icon-container">
      <span th:text="${employee.creationDate}"></span>
      <a th:href="@{'/edit/' + ${employee.id}}">
        
      </a>
    </div>
  </td>
</tr>
</tbody>
</table>
</body>
</html>

```

## 4. edit

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <link rel="stylesheet" type="text/css" href="/css/style.css">
  <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap"
rel="stylesheet">

</head>
<body>
<div class="button-container">
  <a th:href="@{/employee}"><button>Zobacz liste użytkowników</button></a>
</div>
<table>
  <thead>
    <tr>
      <th>ID</th>

```

```

        <th>First Name</th>
        <th>Last Name</th>
        <th>Card Code</th>
        <th>Creation Date</th>
    </tr>
</thead>
<tbody>
<tr th:object="{employee}">
    <td th:text="{employee.id}"></td>
    <td th:text="{employee.firstName}"></td>
    <td th:text="{employee.lastName}"></td>
    <td th:text="{employee.code}"></td>
    <td th:text="{employee.creationDate}"></td>
</tr>

</tbody>
</table>

<div class="title-container">
    <h3>Edit</h3>
</div>

<div class="form-container">
    <form action="/newEmployee" method="post" th:object="{employee}">
        <input type="hidden" th:field="*{id}" />
        <div class="inputs"><label for="firstName">First Name:</label></div>
        <div class="inputs"><input type="text" id="firstName" name="firstName"
th:field="*{firstName}" required/><br/></div>
        <div class="inputs"><label for="lastName">Last Name:</label></div>
        <div class="inputs"><input type="text" id="lastName" name="lastName"
th:field="*{lastName}" required/><br/></div>
        <div class="inputs"><label for="lastCode">Card Code:</label></div>
        <div class="inputs">
            <label for="code"></label><select id="code" name="code" th:field="*{code}">
                <option th:each="code : ${Codes}" th:value="{code}"
th:text="{code}"></option>
            </select>
        </div>
        <div class="input-submit-container"><input class="input-submit" type="submit"
value="Edit"/></div>
    </form>
</div>
</body>
</html>

```

Oprócz tego poniżej zamieszczam style jakie zostały użyte aby poprawić estetykę całego interfejsu.

```

* {
    margin: 0;
    padding: 0;
}

body {
    background-color: #eee;
    font-family: 'Roboto', sans-serif;
}

```

```

        color: black;
    }

    a {
        cursor: pointer;
        text-decoration: none;
        color: white;
    }

    table {
        width: 60%;
        margin-left: auto;
        margin-right: auto;
        border-collapse: collapse;
        margin-bottom: 20px;
        border: 1px solid black;
    }

    thead {
        background-color: #333;
        color: #fff;
    }

    th {
        padding: 10px;
        text-align: left;
    }

    tr:nth-child(even) {
        background-color: #c6c6c6;
    }

    td {
        padding: 10px;
        text-align: left;
        border-bottom: 1px solid #ddd;
        border-bottom: 1px solid black;
    }

    td:first-child {
        font-weight: bold;
    }

    .button-container {
        display: flex;
        align-items: center;
        justify-content: center;
        padding: 20px;
    }

    button {
        width: auto;
        height: 50px;
        padding: 3px 10px;
        border-radius: 5px;
        font-weight: 700;
    }

```

```

        background-color: #333;
        color: #fff;
    }

    button:hover {
        background-color: #666;
        cursor: pointer;
    }

    .title-container {
        display: flex;
        text-align: center;
        justify-content: center;
        margin-bottom: 10px;
        font-size: 30px;
    }

    .form-container {
        width: 500px;
        height: 300px;
        padding: 10px;
        margin-top: 30px;
        border-radius: 10px;
        font-weight: 400;
        background-color: #c1c1c1;
        margin-right: auto;
        margin-left: auto;
        -webkit-box-shadow: 0px 0px 36px -19px rgba(66, 68, 90, 1);
        -moz-box-shadow: 0px 0px 36px -19px rgba(66, 68, 90, 1);
        box-shadow: 0px 0px 36px -19px rgba(66, 68, 90, 1);
    }

    form {
        width: 500px;
        height: 150px;
        margin-right: auto;
        margin-left: auto;
    }

    .input-submit {
        width: 200px;
        height: 50px;
        margin-top: 20px;
        margin-right: auto;
        margin-left: auto;
        border-radius: 10px;
        background-color: #333;
        color: #fff;
    }

    .input-submit-container {
        width: 500px;
        display: flex;
        align-items: center;
        justify-content: center;
        margin-left: auto;
    }

```

```
        margin-right: auto;
    }

    .input-container {
        display:inline-block;
        align-items: center;
        justify-content: center;
        width: 500px;
        margin-left: auto;
        margin-right: auto;
        margin-top: 20px;
    }

    .inputs {
        width: 500px;
        color: white;
        margin-top: 3px;
        display: flex;
        align-items: center;
        justify-content: center;
    }

    .inputs label {
        font-size: 20px;
    }

    .inputs input {
        font-size: 20px;
        padding: 0 5px;
    }

    .edit-icon-container {
        display: flex;
        align-items: center;
        justify-content: space-between;
    }

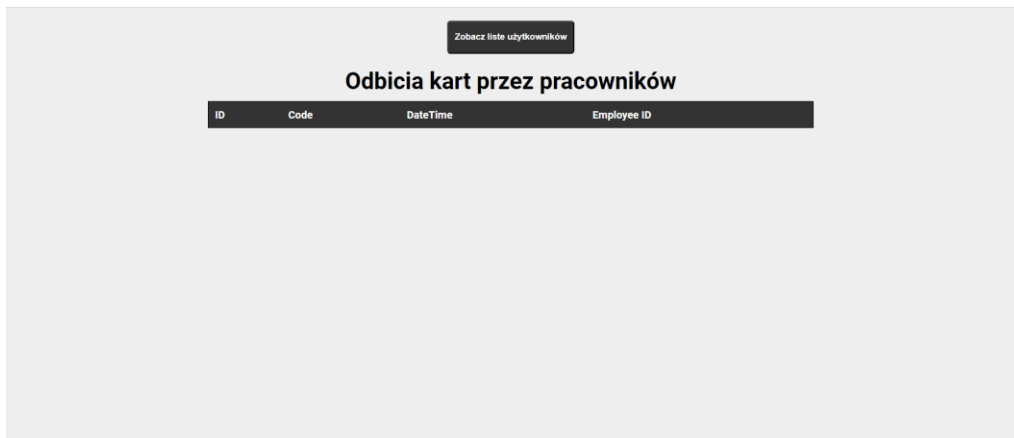
    .edit-icon-container span {
        font-size: 16px;
        margin-right: 10px;
    }

    .edit-icon {
        width: 20px;
        height: 20px;
        cursor: pointer;
    }
```

## 5. Praca aplikacji

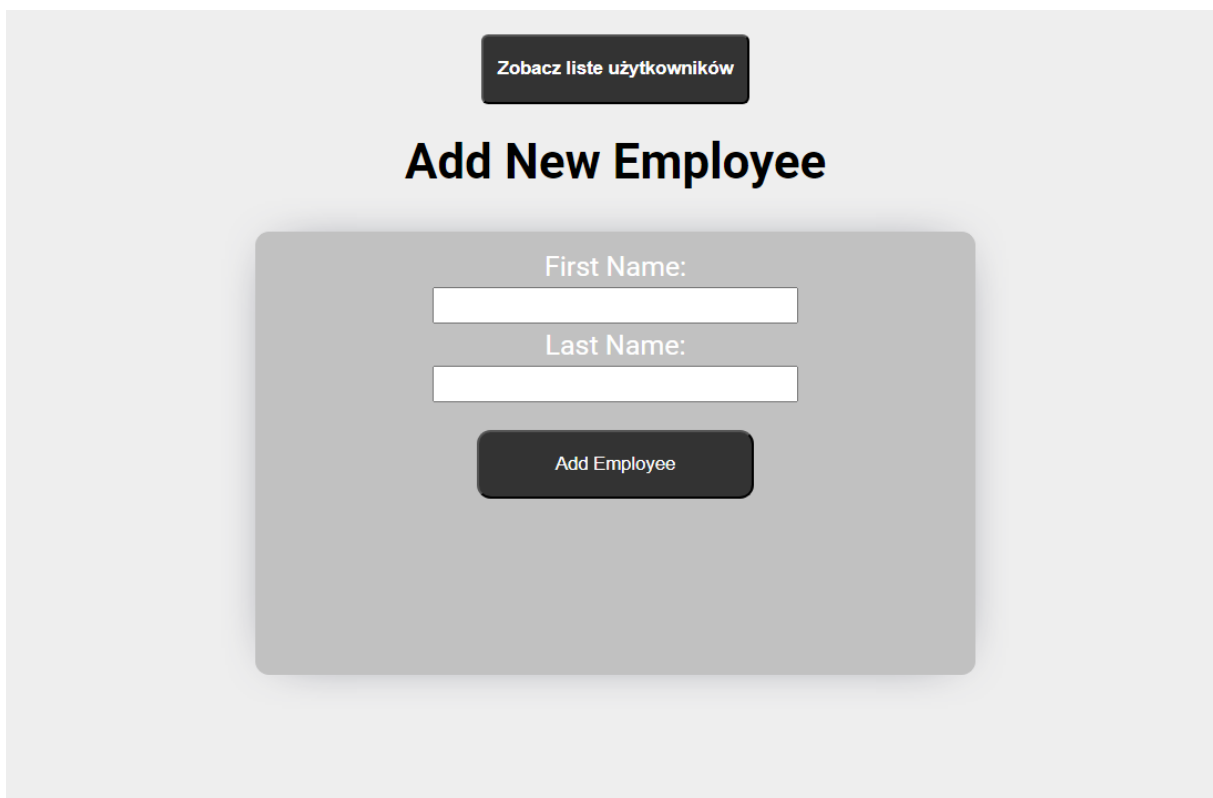
Aplikacja działa w takiej formie że zbliżona karta do czujnika powoduje chwilowe zamigotanie zielonej lampki LED oraz przekazanie kodu z karty na serwer.

Po wejściu na endpoint /home ukazuje nam się widok pustej tabeli. Tabela ta zwraca tylko logi użytkowników którzy są zarejestrowani w aplikacji.



8. Widok pustej tabeli logów

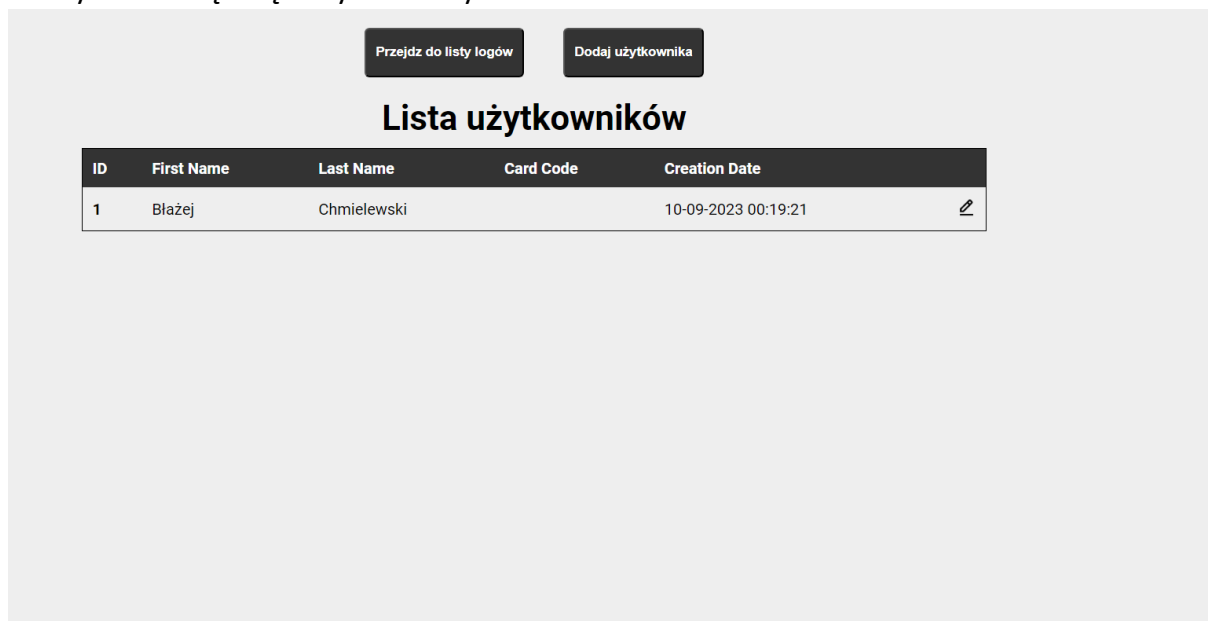
Przejdźmy zatem do formularza dodawania użytkowników

The screenshot displays a web form titled 'Add New Employee'. At the top, there is a button labeled 'Zobacz liste użytkowników'. The main heading 'Add New Employee' is prominently displayed. Below the heading, the form is contained within a light gray box. It features two input fields: 'First Name:' and 'Last Name:'. At the bottom of the form box is a button labeled 'Add Employee'.

9. Widok formularza tworzenia użytkownika

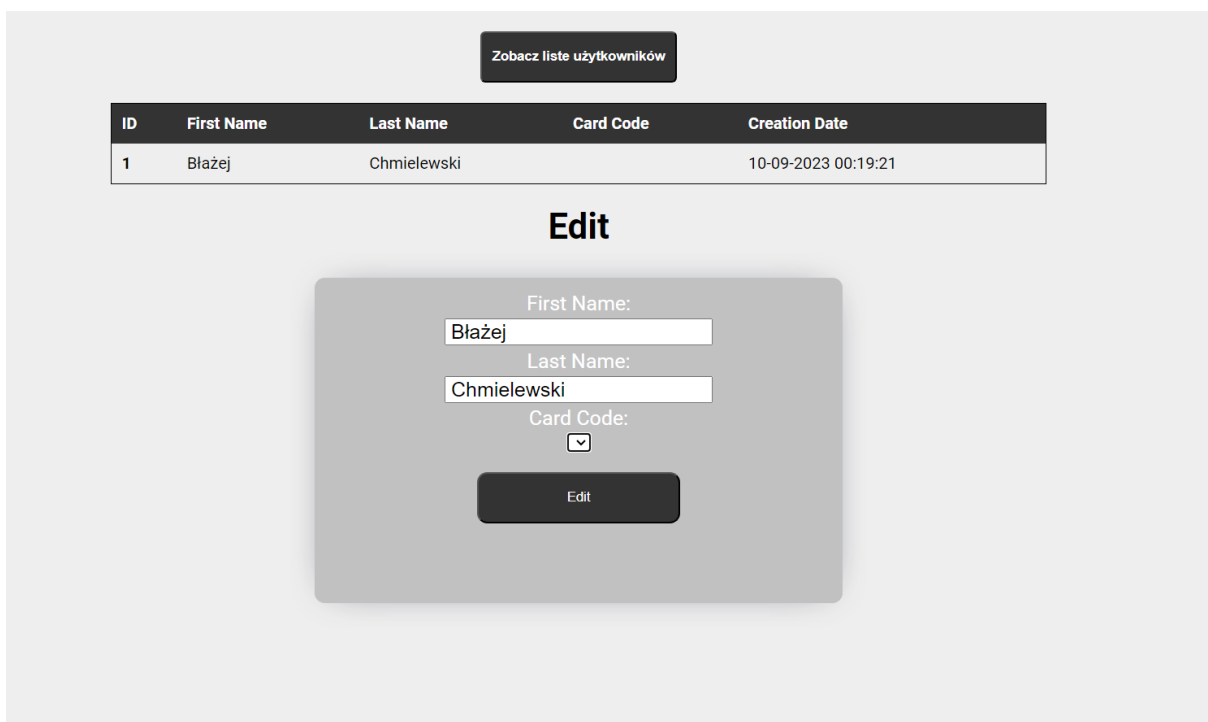


Po utworzeniu użytkownika zostajemy przekierowani na endpoint /employee gdzie mamy zwróconą listę wszystkich użytkowników.



10. Lista użytkowników z pierwszym użytkownikiem

Należy teraz zedytować użytkownika przypisując mu kartę



11. Formularz edycji bez możliwości wybrania karty

Przechodzimy do edycji i mamy tutaj możliwość zmiany pozostałych parametrów ale domyślnie imię oraz nazwisko zostają tak samo. Z listy rozwijalnej możemy

W aplikacji serwera możemy zobaczyć odpowiedni log :



Zobacz listę użytkowników

ID	First Name	Last Name	Card Code	Creation Date
1	Błażej	Chmielewski		10-09-2023 00:19:21

## Edit

First Name:

Last Name:

Card Code:

Edit

### 13. Widok formularza edycji z możliwością wybrania karty

Teraz odbijamy się jeszcze raz i na liście logów kart jest nasz użytkownik z datą odbicia karty oraz przekazanym jego id w celu anonimizacji.

Zobacz liste użytkowników

Odbicia kart przez pracowników

ID	Code	DateTime	Employee ID
1	5110983154	10-09-2023 00:25:11	1

14. Widok listy logów(1)

Teraz zostaną dodani 2 użytkownicy, przypisane zostaną im karty i wykonane kilka logów

Zobacz liste użytkowników




Odbicia kart przez pracowników

ID	Code	DateTime	Employee ID
1	5110983154	10-09-2023 00:25:11	1
2	14778170170	10-09-2023 00:28:07	2
3	14778170170	10-09-2023 00:28:25	2
4	5110983154	10-09-2023 00:30:43	1
5	5110983154	10-09-2023 00:30:59	1
6	14778170170	10-09-2023 00:31:04	2
7	5113816420	10-09-2023 00:32:25	3
8	5113816420	10-09-2023 00:32:34	3

15. Widok listy logów(8)

[Przejdź do listy logów](#)[Dodaj użytkownika](#)

## Lista użytkowników

ID	First Name	Last Name	Card Code	Creation Date	
1	Błażej	Chmielewski	5110983154	10-09-2023 00:25:05	
2	Jan	Kowalski	14778170170	10-09-2023 00:26:48	
3	Adam	Nowak	5113816420	10-09-2023 00:32:21	

16. Widok listy użytkowników

## 6. Podsumowanie, obserwacje, wnioski oraz zalety i wady

### 6.1 Podsumowanie

Protokół http dostarcza nam ciekawej możliwości bezprzewodowej komunikacji. Wykorzystanie serwera z graficznym interfejsem w przeglądarce znacząco usprawnia proces monitorowania i administrowania tego typu systemów. Uważam że cały projekt jest kompleksowym rozwiązaniem łączącym w sobie szeroki wachlarz nowoczesnych narzędzi. Cały system jest prosty, wydajny, skalowalny oraz otwarty na budowę modułową

### 6.2 Obserwacje

- Zaobserwowano że mikrokontroler pracujący w sieci nie komunikuje się z serwerem jeżeli została wykorzystana sieć 5GHz, dopiero zmiana na sieć 2.4GHz pozwoliła na komunikację klient serwer.
- Brak wykorzystania konwertera napięcia może uszkodzić płytkę. Napięcie zasilania z ogniwami o łącznym napięciu  $\sim 11.4\text{ V}$  wymaga konwertowania do 5V. Warto zaznaczyć że przy wgrywaniu programu i połączeniu komputera z płytką, dostarczone zewnętrzne zasilanie o takim napięciu może uszkodzić również komputer, na przykład spalić port USB w komputerze.
- Przy zetknięciu karty do czujnika gdy nie ma przypisanego użytkownika sprawia że dioda sygnalizująca mruga bardzo słabo. Dopiero po przypisaniu użytkownika do karty i odbiciu go ponownie, można zaobserwować prawidłowe zaświecenie diody LED.

### 6.3 Wnioski

- Dopiero po zweryfikowaniu zaprojektowanego modelu mikrokontrolera zauważono że nie obsługuje on sieci 5G. Jest to podyktowane tym że fale o większej częstotliwości a co za tym idzie mniejszej długości, nie przenoszą sygnału tak dobrze jak krótsze fale
- Szybkie zgaszenie diody LED może być spowodowane tym że gdy użytkownik nie jest przypisany to program mimo wszystko stara się wyszukać

użytkownika i robi to na tyle długo że zaburza to kolejność wykonania metod. Dioda świeci przez pewien okres i gaśnie szybko bo przerywa jej działanie opóźniająca metoda z wyszukiwaniem użytkownika. Jest to klasyczny przykład problemu wynikającego z asynchronicznej pracy układów.

## 6.4 Zalety i Wady

### 6.4.1 Zalety

- Budowa modułowa, otwartość na dołożenie nowych modułów mikrokontrolera z czujnikami.
- Zapisy w bazie danych przechowywane są w sposób trwały, to znaczy że po planowanych bądź nie planowanych zamknięciu aplikacji, dane ciągle są dostępne
- Rozwiązanie jest tanie i ma minimalną ilość elementów
- Rozwiązanie ma wmontowane zasilanie bateriami. Pozwala nam to na uniezależnienie się od źródeł prądowych przewodowych oraz nie wygodnych przyłączy i nie pasujących logistycznie gniazd elektrycznych.

### 6.4.2 Wady

- Zasilanie jest jednocześnie wadą i zaletą. Mimo wygody w montowaniu niestety zmusza użytkownika do tego aby ładować baterie. Z obliczeń wynika że 3 baterie są w stanie zaspokoić prądowo prace utrzymania samego mikrokontrolera na 30 godzin.
- Przesyłanie danych protokołem http jest ryzykowne z punktu widzenia zabezpieczeń. Wystarczy że jakiś użytkownik będzie podpięty w naszą sieć i będzie mógł on śledzić ruch w niej i podglądać przesyłane przez nas kody. Jest dużo prostych narzędzi które są darmowe i mogą w tym pomagać na przykład : Wireshark. Warto rozważyć wprowadzenie protokołu https do ochrony danych użytkowników.
- Brak zamontowanego urządzenia sygnalizującego dźwiękowego. Sprawia że dla użytkownika fizycznego nie zawsze jest oczywiste kiedy następuje moment zaczytania przez program jego karty i zmusza go do przyglądania się diodzie co jest nie wygodne.

## 7. Bibliografia

- <https://miliohm.com/rc522-rfid-reader-with-nodemcu/>
- <https://bykowski.pl/kurs-spring-boot-2-tworzenie-efektywnych-aplikacji-internetowych-zapisy/>
- <http://hilite.me/>
- <https://forum.supla.org/viewtopic.php?t=4329&start=40>
- „Java: podstawy” Cay S. Horstmann
- „Java. Techniki zaawansowane. Wydanie VIII” Cay S. Horstmann, Gary Cornell
- „Springboot-livebook” A.P Bykowski

## 8. Spis Ilustracji

1. Mikrokontroler ESP8266.....	3
2. Czujnik RFID RC522 .....	4
3.Model MVC.....	7
4. Fizyczny połączenie układu płytki .....	8
5. Schemat przedstawiający połączenie układu .....	9
6. Połączenie pinów cyfrowych .....	9
7. Drzewo projektu serwera .....	12
8. Widok pustej tabeli logów .....	32
9. Widok formularza tworzenia użytkownika .....	32
10. Lista użytkowników z pierwszym użytkownikiem.....	33
11. Formularz edycji bez możliwości wybrania karty .....	33
12. Zrzut ekranu przekazanego do aplikacji kodu z karty .....	34
13. Widok formularza edycji z możliwością wybrania karty.....	34
14. Widok listy logów(1) .....	35
15. Widok listy logów(8) .....	35
16. Widok listy użytkowników .....	36