Presence Registration System

System rejestracji obecności na zajęciach

Opis aplikacji desktopowej realizowanej w ramach przedmiotu
[PODSTAWY TELEINFORMATYKI]

Kamil Zieliński

Konrad Karkos

Marcin Stasiak

Adrian Staśkiewicz

Poznań, 19 czerwca 2019r.

1. Spis treści

2. Wstęp

Niniejszy dokument ma na celu przedstawienie szczegółowych założeń projektu realizowanego w ramach zajęć projektowych z przedmiotu Podstawy Teleinformatyki. Głównym jego celem jest stworzenie aplikacji desktopowej oraz niezbędnych narzędzi po stronie serwera w celu zapewnienia pełnej funkcjonalności oraz prawidłowego działania systemu rejestrującego obecność podczas zajęć. Projekt wraz z dokumentacją ma stanowić formę zaliczenia przedmiotu. W skład zespołu projektowego realizującego system o roboczej nazwie Presence Registration System weszli Adrian Staśkiewicz, Kamil Zieliński, Marcin Stasiak oraz Konrad Karkos.

Temat został wybrany, ze względu na możliwość nauki połączenia aplikacji napisanej w języku Java z urządzeniami peryferyjnymi, w tym przypadku z czytnikiem kart. Istotnym elementem, który był brany pod uwagę podczas wyboru tematu, była chęć opanowania samego odczytu danych z kart typu smardcard. Niewątpliwie ważnym czynnikiem była także wizja pracy nad ciekawym i nieszablonowym zadaniem.

Dostęp autoryzowany w wielu firmach odbywa się poprzez korzystanie z czytników. Model **Manhattan Czytnik SMART CARD** jest pierwszym krokiem do zrozumienia jak działają nośniki danych typu inteligentne karty. Warto dodać, że najbardziej popularnym standardem RFID (Radio-frequency identification) jest Mifare, standard ten wykorzystywany jest między innymi w kartach bankomatowych.

Wraz z rozwojem technologii, odkryto nowe sposoby identyfikacji osób za pomocą biometrii. Najpopularniejszym, używanym od wielu lat, jest ten bazujący na odciskach palców. Czytniki biometryczne stają się coraz bardziej popularne. W wielu instytucjach identyfikacja na podstawie cech biometrycznych sprawdza się bardzo dobrze. Do najpopularniejszych technik należą:

- sprawdzanie wzór tęczówki oka
- sprawdzanie wzoru linii papilarnych

Korzystanie z inteligentnych kart jest mniej bezpieczne niż korzystanie z wymienionych wyżej technologii, chociażby ze względu na ryzyko utraty, bądź

uszkodzenia karty, a w najgorszym przypadku na dostęp do karty przez osoby nieupoważnione.

3. Opis aplikacji

Presence Registration System to oprogramowanie mające na celu usprawnienie działań podejmowanych w ramach sprawdzania obecności na zajęciach. System umożliwia przechowywanie wprowadzanych przez jego użytkowników informacji, które są magazynowane w scentralizowanej bazie danych. Dostęp do znajdującej się w niej zawartości możliwy jest przy użyciu dedykowanej aplikacji desktopowej, która za pośrednictwem interfejsu graficznego w łatwy i intuicyjny sposób pozwala na szeroko pojęte zarządzanie dostępnymi w systemie zasobami.

4. Przeznaczenie

System rejestracji obecności na zajęciach to narzędzie przeznaczone dla pracowników placówek edukacyjnych, którzy chcą ujednolicić i przyspieszyć proces sprawdzania obecności podczas zajęć, a dzięki dodatkowym funkcjom oprogramowania móc analizować i przetwarzać gromadzone w systemie dane.

Podstawowa, kliencka część oprogramowania Presence Registration System to aplikacja desktopowa. Została przystosowana do uruchomienia w każdym środowisku z dostępem do internetu, na którym zainstalowano pakiet Java Development Kit w wersji 10. Wyższe wersje tego pakietu nie zawierają kluczowych bibliotek, bez których aplikacja PRS może nie działać prawidłowo lub w ogóle się nie uruchamiać. Ze względu na zewnętrzną, scentralizowaną bazę danych są to jedyne wymagania, które należy spełnić by móc w pełni korzystać z oprogramowania.

5. Opis prac

Poniższa tabela zawiera opis wszystkich prac związanych z systemem rejestracji obecności na zajęciach z uwzględnieniem członków zespołu odpowiedzialnych za dane zadanie oraz ewentualnymi uwagami do jego realizacji.

Opis zadania	Osoby realizujące zadanie	Uwagi
Projekt architektury systemu	Konrad Karkos, Kamil Zieliński, Adrian Staśkiewicz, Marcin Stasiak	opracowanie wstępnej wizji systemu i wylistowanie potencjalnych narzędzi pomocnych w implementacji
Projekt bazy danych	Konrad Karkos	opracowanie tabel oraz relacji między nimi
Obsługa czytnika kart	Konrad Karkos, Kamil Zieliński	
Implementacja rdzenia systemu	Adrian Staśkiewicz	dotyczy zaprojektowanej architektury systemu
Obsługa zapytań RESTowych po stronie serwera	Konrad Karkos, Kamil Zieliński, Adrian Staśkiewicz, Marcin Stasiak	
Komunikacja z bazą danych	Konrad Karkos, Kamil Zieliński, Adrian Staśkiewicz, Marcin Stasiak	
Implementacja funkcjonalności rejestrowania obecności	Konrad Karkos, Kamil Zieliński	

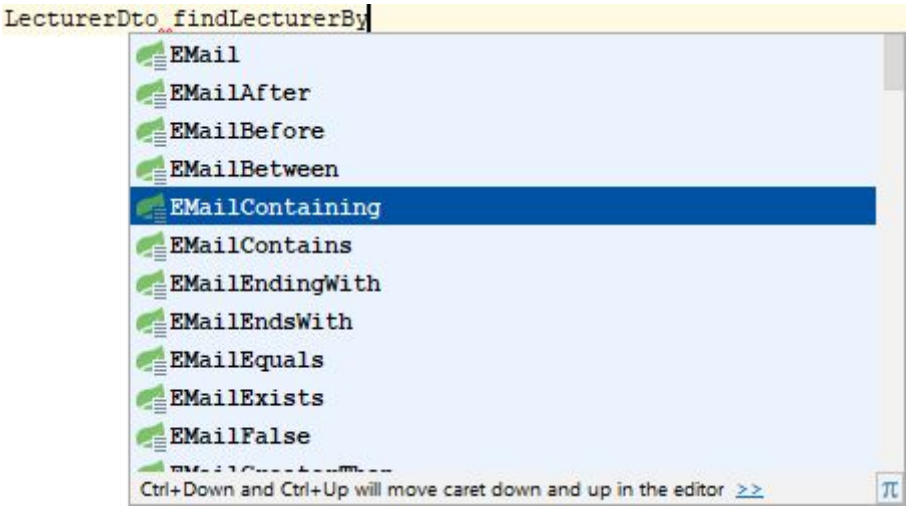
Opis zadania	Osoby realizujące zadanie	Uwagi
Implementacja funkcjonalności przeglądania archiwalnych list obecności	Kamil Zieliński, Marcin Stasiak	
Implementacja funkcjonalności przeglądania szczegółów dotyczących archiwalnej listy obecności	Kamil Zieliński, Marcin Stasiak	
Implementacja funkcjonalności przeglądania statystyk studentów	Marcin Stasiak, Adrian Staśkiewicz	
Projekt interfejsu graficznego	Marcin Stasiak	
Stylowanie interfejsu graficznego	Adrian Staśkiewicz	

6. Technologie

W celu zapewnienia jak największej stabilności systemu do jego implementacji zostały wybrane optymalne narzędzia i technologie, które gwarantują nie tylko płynność działania oprogramowania ale także zapewniają prostotę kodu i łatwość jego refaktoryzacji.

- Maven
    - Narzędzie automatyzujące pracę z projektem
- Spring Framework (Spring Boot)
    - Narzędzie pomocnicze, które umożliwia łatwą konfigurację serwera RESTowego
    - Dodatkowo dzięki modułowi Spring Data, możemy w prosty sposób pisać zapytania do bazy danych. Przykładowo:

      `Card findCardById(String id);` - pozwala nam znaleźć kartę o danym ID. Są to metody generowane automatycznie, jeżeli będziemy trzymali się poprawnego nazewnictwa funkcji.



      Na powyższym obrazku IntelliJ podpowiada szereg możliwych rozwiązań dla obecnej nazwy funkcji. Parametry, które są podane przy wywołaniu muszą być zgodnie z kolejnością parametrów w nazwie funkcji. Przykładowo: jeżeli chcemy sprawdzić czy w danym dniu, o danej godzinie w danej sali był student, to nazwa funkcji będzie wyglądała następująco:

```
PresenceOnLecture findByPresenceDateAndHourTimeRoomAndStudent(Date date, String hour, String room, Student student);
```

W powyższym przykładzie argumenty są podane zgodnie z kolejnością tak jak w nazwie funkcji. Nazwy argumentów, nie muszą być takie same.

- Hibernate (JPA)

    - Narzędzie służące do mapowania obiektowo-relacyjnego, które pozwala nam na zamienienie poszczególnych tabel w bazie relacyjnej, na obiekty.
- QueryDSL
    - Narzędzie, które pomaga w stworzeniu zapytań do bazy danych.
- Java Smart Card I/O
    - Biblioteka dzięki, której można pobrać informacje z czytnika kart. Jest ona dosyć starą, nieaktualizowaną biblioteką, ale jedyną dostępną w języku Java
- Microsoft SQL Server Management Studio
    - Baza danych.
- Lombok
    - Narzędzie, które usprawnia pisanie kodu w języku Java. Dzięki niemu kod jest bardziej czytelny
    - Przykładowy kod:

```
package entities;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import javax.persistence.*;

@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "CARD_0005")

public class Card {
   @Id
```

```
    @Column(name = "id")
    private String id;
    @OneToOne()
    @JoinColumn(name = "student_id")
    private Student student;
}
```

Za pomocą Loomboka (adnotacje: @Setter, @Getter, @NoArgsConstructor, @AllArgsConstructor), nie trzeba w kodzie tworzyć Getterów, Setterów dla zmiennych w powyższym obiekcie. Również automatycznie tworzone są dwa konstruktory: bezargumentowy, oraz zawierający wszystkie argumenty. Dzięki takiemu rozwiązaniu, można łatwo uniknąć nadmiaru kodu.

- JavaFX
    - Biblioteka graficzna Javy, która począwszy od 11 wersji Javy przestała być wspierana przez twórców.
- Git (GitHub)
    - Utworzony kod, umieszczany jest na repozytorium githubowym. W łatwy sposób można kontrolować kod oraz w razie potrzeby cofnąć commity, które zaburzyły poprawne działanie aplikacji.
- SonarLint
    - Jest pluginem, który sprawdza jakość kodu. Pomaga on np. przy literówkach, powtarzalności w kodzie. Wpływa znacząco na czytelność kodu.

7. Architektura

Aplikacja oparta jest na architekturze systemu komputerowego klient-serwer. W pewnym uproszczeniu, sposób komunikacji można przedstawić za pomocą określonych zadań, w których:

- serwer zapewnia usługi dla użytkowników;
- użytkownicy wysyłają zapytania do serwera.

Serwer stanowi oprogramowanie obsługujące zapytania RESTowe wysyłane z aplikacji desktopowych uruchamianych na urządzeniach klienckich. Dostęp do

scentralizowanej bazy danych odbywa się więc za pośrednictwem aplikacji desktopowej, która odpytuje serwer. Ten z kolei wykonuje operacje zgodnie z zaimplementowanymi metodami, wykonując działania na bazie danych i wysyła odpowiedni pakiet w ramach odpowiedzi do aplikacji użytkownika. Warto zaznaczyć, że dla każdego użytkownika wydzielona jest odrębna pula danych i mogą oni pobierać informacje o rekordach, które są im przypisane.

Fizyczna struktura systemu przedstawia się następująco:

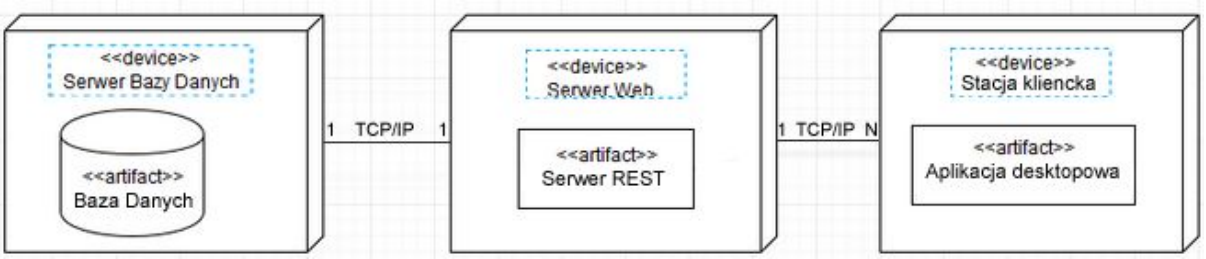


Diagram wdrożenia

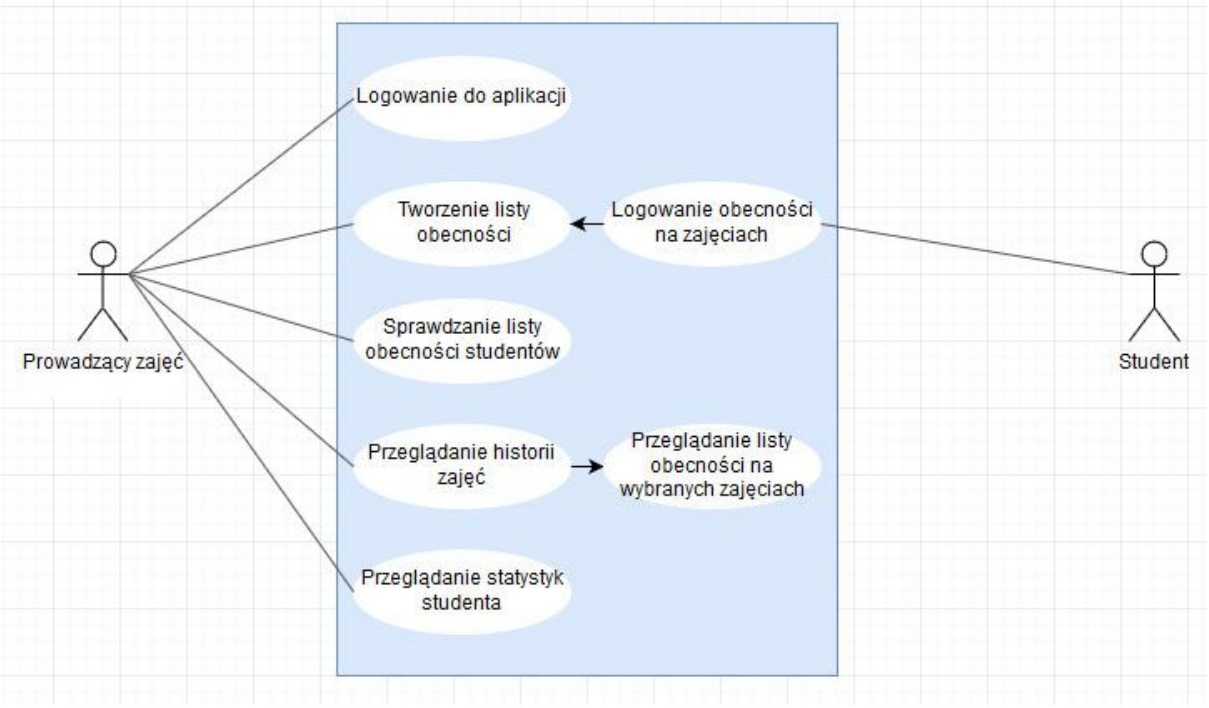


Diagram przypadków użycia

Powyższy diagram jednoznacznie wskazuje funkcjonalności przypisane poszczególnym grupom użytkowników. Dla przykładu, student może jedynie

rejestrować swoją obecność na zajęciach. Dzieje się to za pośrednictwem czytnika kart. Więcej informacji na temat sprawdzania obecności znajduje się w rozdziale Instrukcja użytkowania aplikacji. Dla użytkowników przypisanych do grupy prowadzących aplikacja udostępnia najszerszy wachlarz funkcjonalności. Korzystający z aplikacji mogą tworzyć nowe listy obecności, sprawdzać szczegóły archiwalnych list oraz rejestrować bieżące obecności. Dla tej grupy użytkowników dostępna jest także pełna funkcjonalność związana ze statystykami studentów. Użytkownik może sprawdzić ile razy dany student pojawił się na zajęciach, a także ile razy opuścił daną jednostkę lekcyjną.

8. Funkcjonalności

- Logowanie się do aplikacji jako prowadzący - separacja danych,
- Sprawdzenie listy obecności na zajęciach,
- Rejestrowanie spóźnień,
- Przeglądanie historii zajęć,
- Przeglądanie listy studentów obecnych na wybranych zajęciach,
- Przeglądanie statystyk studenta
    - ilość obecności wybranego studenta na wskazanych zajęciach;
    - ilość nieobecności wskazanego studenta na wybranych zajęciach;
    - całkowita ilość zajęć odbytych przez danego prowadzącego
    - graf kołowy przedstawiający stosunek ilości obecności do nieobecności

Wymagania pozafunkcjonalne:

1. Zgodnie z założeniami projektu, komunikacja między systemem a użytkownikiem została zrealizowana za pośrednictwem interfejsu graficznego. Ma to na celu zapewnienie intuicyjności oraz ułatwienie w nawigacji po aplikacji. Interfejs graficzny ma maksymalnie uprościć korzystanie z dostępnych funkcjonalności i dodatkowo zabezpieczyć użytkownika przed niewłaściwym wykonaniem operacji.
2. Aplikacja powinna działać niezawodnie, przy czym powinna być przystosowana do obsługi możliwych błędów. Modułowość

oprogramowania ma umożliwić bezproblemowe korzystanie z każdej części systemu z osobna, tak, aby błędy w jednej części programu nie wpływały na inne, niepowiązane ze sobą komponenty.

9. Błędy i trudności w implementacji

Pierwszą napotkaną trudnością było ustalenie hierarchii plików znajdujących się na naszych kartach. Politechnika nie udostępnia na swoich stronach szczegółowych informacji co do sposobu programowania i przechowywania danych na kartach. Została podjęta decyzja o skorzystaniu z wykładu, z innej uczelni, na temat programowania kart, który prezentował sposób przechowywania danych.

Drugim kamieniem milowym tego projektu było skonstruowanie odpowiedniej komendy APDU (smart card application protocol data unit), która pozwalałaby na odczyt danych znajdujących się w plikach karty. Po zdekompilowaniu zewnętrznego oprogramowania odczytującego podstawowe dane karty oraz przejrzeniu dostępnych dokumentacji, udało się utworzyć komendę odczytującą tylko i wyłącznie numer karty.

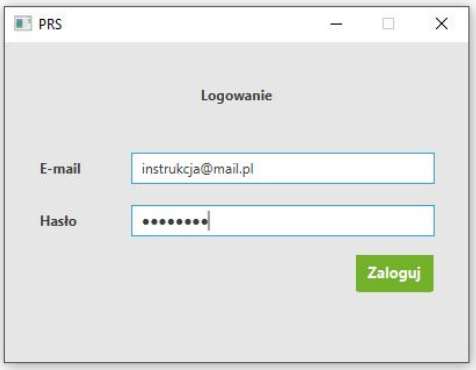
Kolejnym napotkanym problemem było utworzenie wątku nasłuchującego aktywności czytnika. Problem ten sprwaiał największą trudność podczas prób komunikacji między wątkiem pobocznym obsługującym czytnik, a wakiem głównym odpowiedzialnym za logikę biznesowa klienckiej części (aplikacji desktopowej).

10. Instrukcja użytkowania aplikacji

Poniższa sekcja zawiera opis istotnych czynności wykonywanych przez użytkownika w trakcie działania programu. Instrukcja ma za zadanie wyszczególnienie i wyjaśnienie wszystkich kroków, które należy wykonać w celu prawidłowego korzystania z omawianych funkcjonalności.

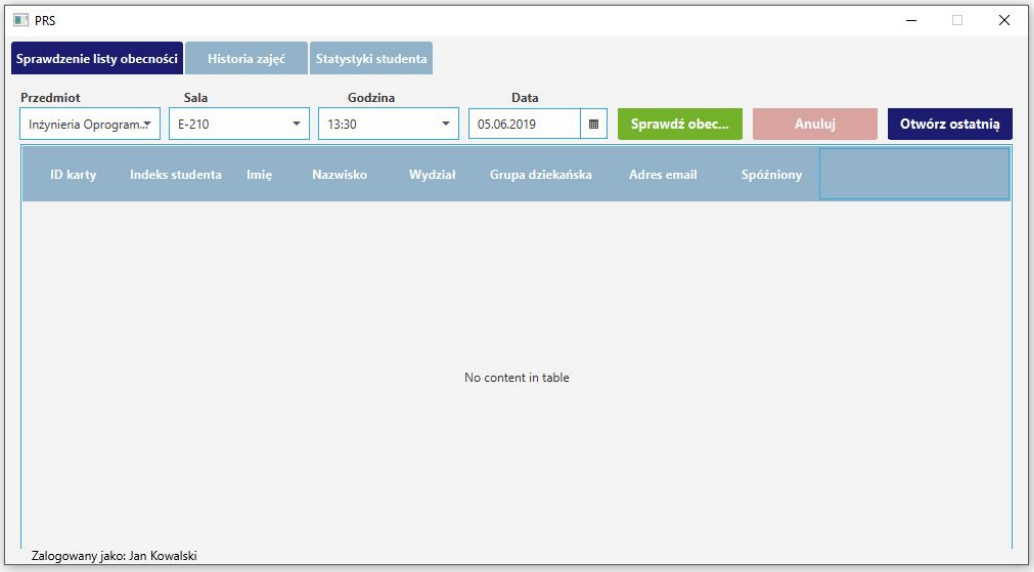
Poniższe zrzuty ekranu powinny (wraz z opisem) powinny przybliżyć możliwości i poprawną obsługę aplikacji.

- Logowanie



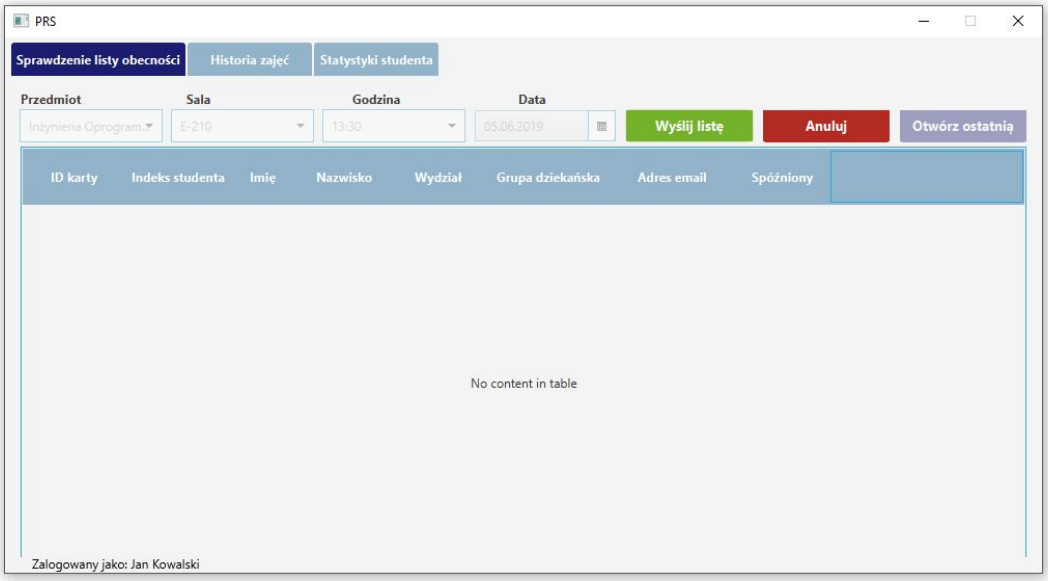
Po wprowadzeniu danych dostępowych określonego prowadzącego, system sprawdzi czy dane są właściwe, a następnie przekieruje użytkownika do widoku głównego aplikacji przedstawionego poniżej.

- Sprawdzanie obecności
    - Tworzenie nowej listy obecności



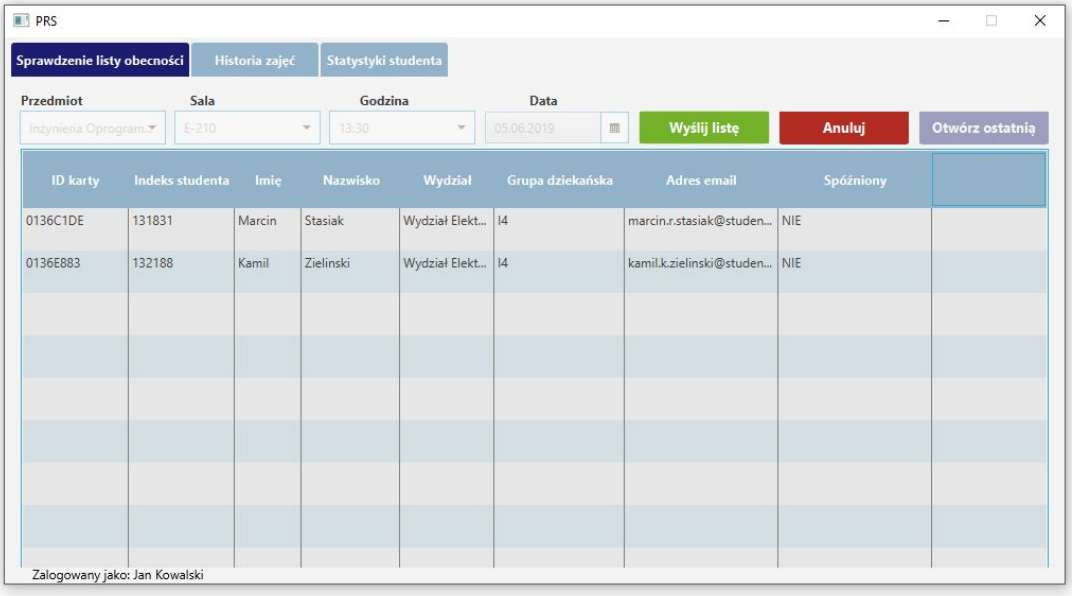
Aby utworzyć nową listę obecności należy wybrać dane, które charakteryzują konkretną jednostkę lekcyjną. Są to między innymi: Przedmiot, Sala, Godzina oraz Data.

    - Anulowanie procesu sprawdzania obecności



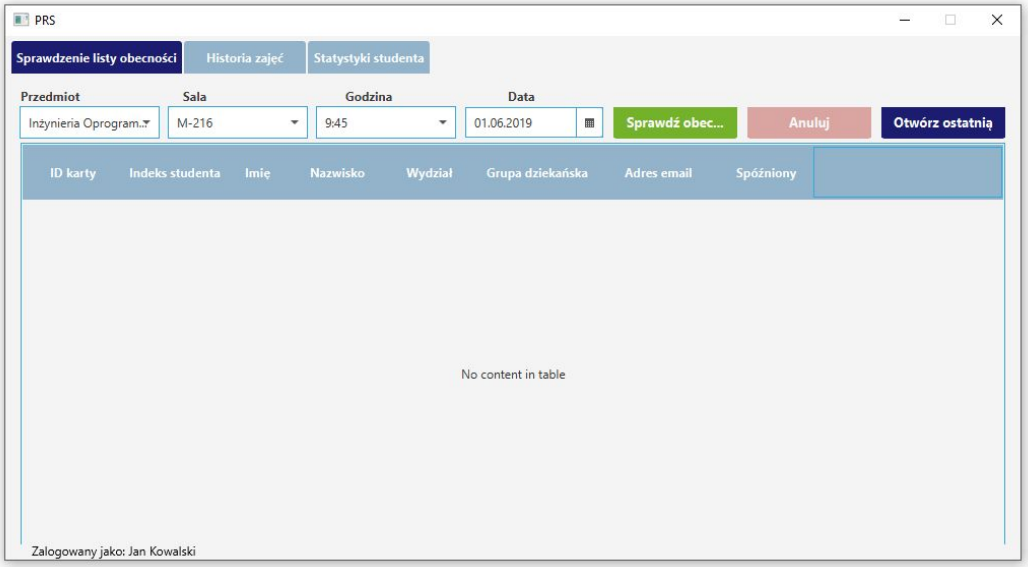
Po wyborze danych i rozpoczęciu sprawdzania listy obecności istnieje możliwość anulowania tej akcji. Służy do tego odpowiedni przycisk, który porzuca wszystkie dotychczasowe akcje, kończy wątek poboczny, który jest odpowiedzialny za nasłuchiwanie.

- Zakończenie procesu sprawdzania obecności i wysłanie listy



Gdy prowadzący utworzy listę obecności dla swojego przedmiotu, a studenci zalogują swoją obecność - użytkownik może wysłać listę do bazy danych. W ten sposób system zaloguje w bazie wszystkich studentów, którzy na danych zajęciach pojawili się. Ponadto widok zostanie przywrócony do podstawowego.

- Ponowne otwieranie zamkniętej listy obecności



Dla osób spóźnionych (po sprawdzeniu obecności na zajęciach i zatwierdzeniu danych przez prowadzącego) istnieje możliwość ponownego otwarcia zapisanej listy obecności. W tym celu należy kliknąć przycisk “otwórz ostatnią”, jeśli otwarta ma zostać ostatnia lista (ta znajdująca się na samej górze spisu list archiwalnych w zakładce “Historia zajęć”) Istnieje także opcja ręcznego wybrania która lista ma zostać otwarta ponownie. W tym celu należy wprowadzić wszystkie dane o wybranej liście. System automatycznie przyporządkuje nowe obecności do wskazanej listy. Po wybraniu odpowiednich ustawień należy rozpocząć sprawdzanie w standardowy sposób.

- Przeglądanie archiwalnych list obecności
    - Przeglądanie wszystkich archiwalnych list obecności

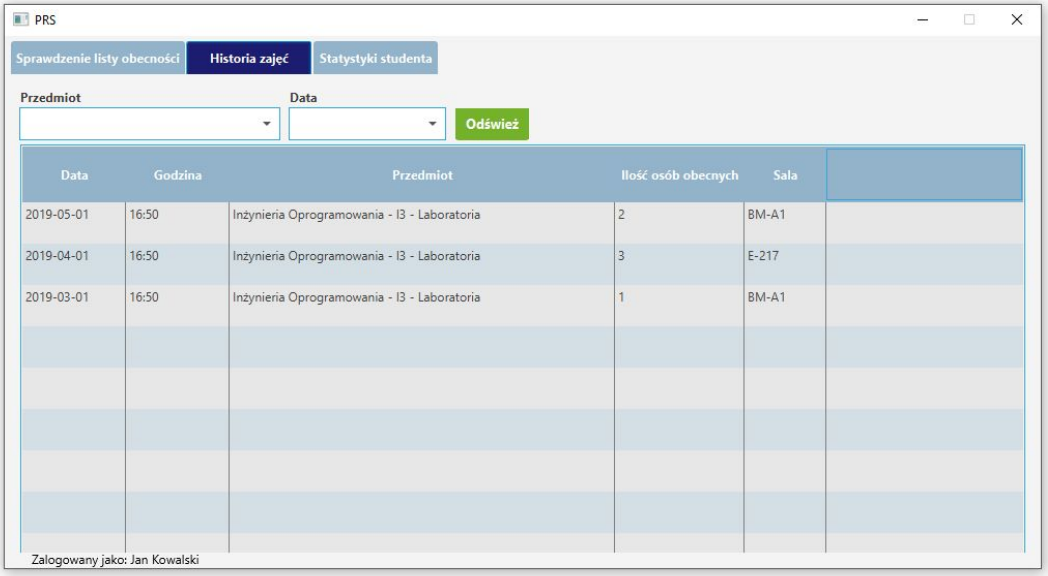
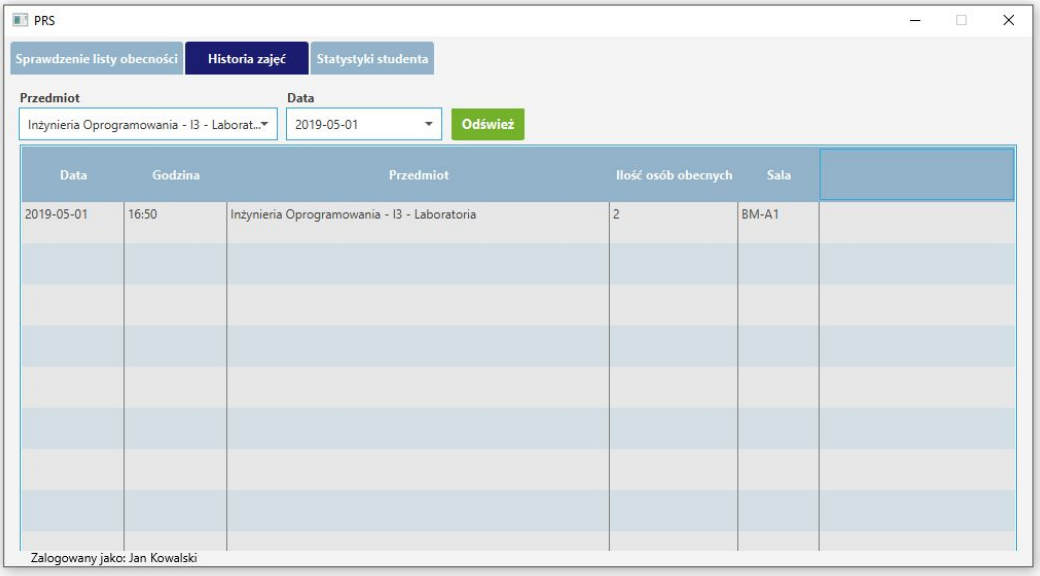


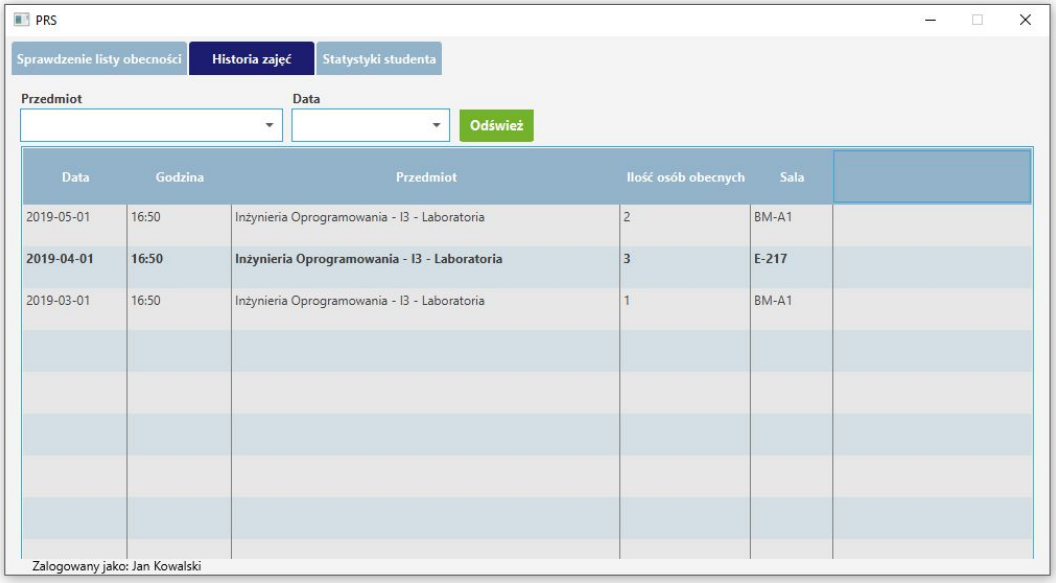
Tabela z archiwalnymi listami obecności ma na celu przybliżyć prowadzącemu informacje na temat dotychczasowych prowadzonych przez niego zajęć.

- Filtrowanie list obecności po nazwie przedmiotu i/lub dacie

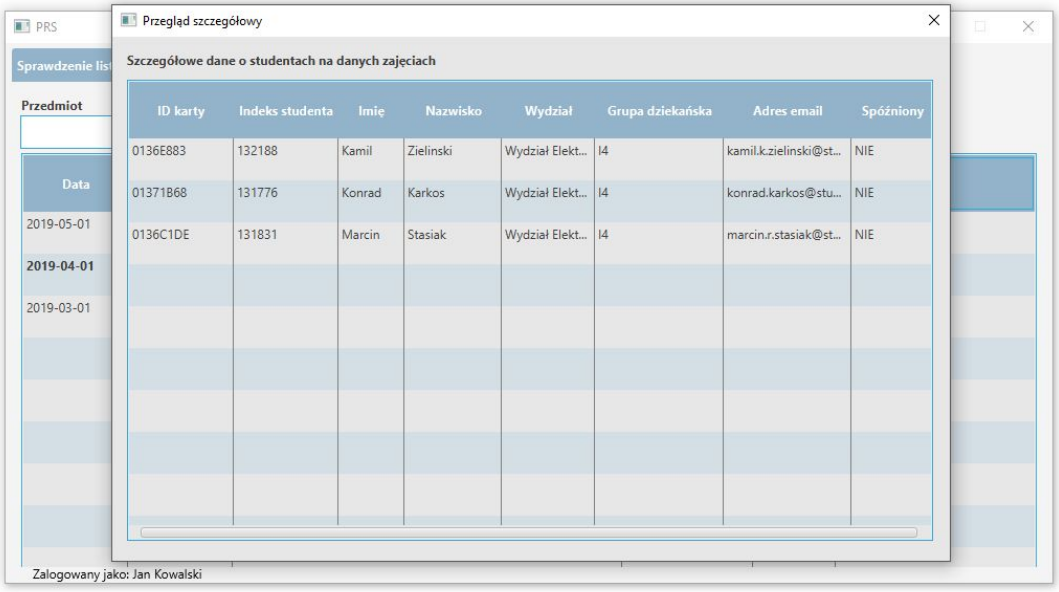


Tak jak w przypadku wyboru danych dotyczących wskazanej przez użytkownika jednostki lekcyjnej, tak i tu istnieje możliwość filtrowania danych. Z bazy danych pobierane są informacje dotyczące lekcji prowadzonych przez aktualnie zalogowaną osobę.

- Przeglądanie szczegółowych informacji dotyczących archiwalnej listy obecności

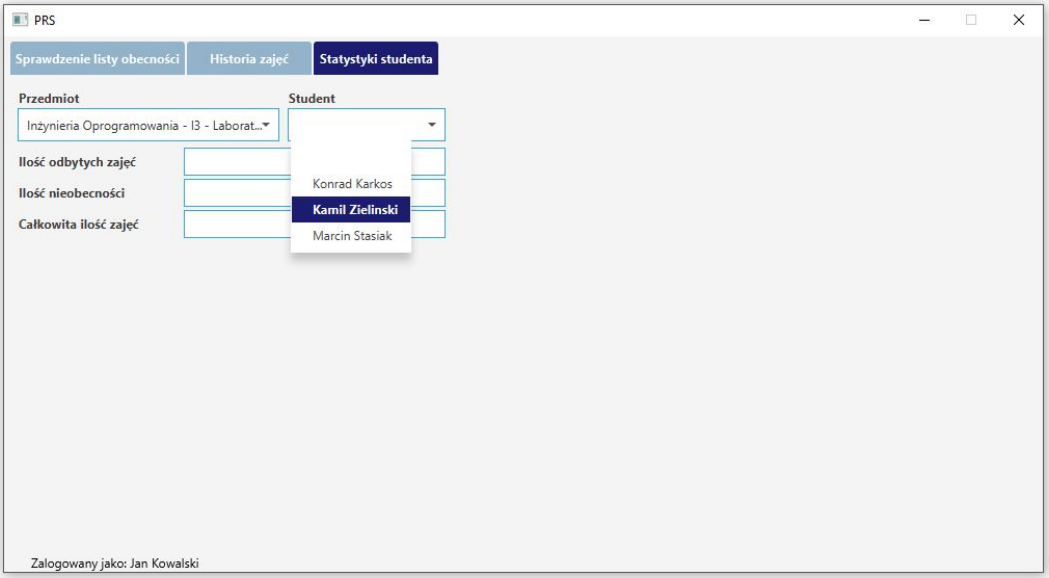


Gdy użytkownik nie wybierze przedmiotu ani daty - system domyślnie pobierze z bazy wszystkie rekordy dotyczące zajęć zalogowanego prowadzącego.

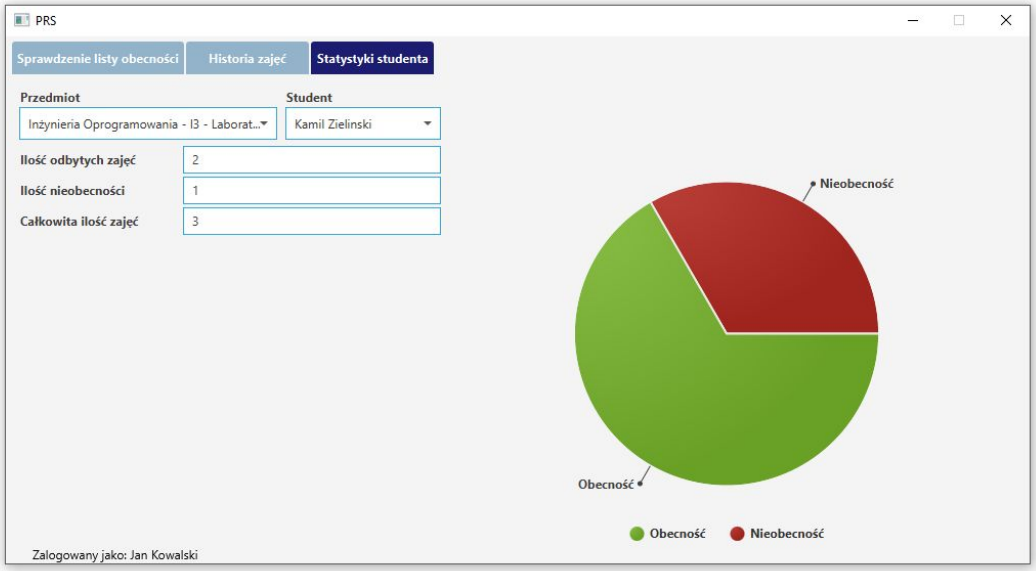


Po procesie filtrowania (po wyborze interesujących danych), system wyświetla listy obecności. Tabela umożliwia podgląd danych szczegółowych wybranej jednostki lekcyjnej. W celu wyświetlenia okna z listą studentów obecnych na zajęciach należy dwukrotnie kliknąć lewym przyciskiem myszy na wybranej pozycji w tabeli zawierającej historyczne listy obecności.

- Przeglądanie statystyk studenta



W trzeciej zakładce zawarte są przedmioty aktualnie zalogowanego prowadzącego. Może on wybrać dany przedmiot - system przefiltruje listę studentów, którzy aktualnie znajdują się w tej grupie.



Po wyborze studenta, system wyświetla informacje odnośnie wybranego przedmiotu, ilości obecności na danych zajęciach, a także jego nieobecnościach. Dane prezentowane są w formie tekstowej. W celu

zwizualizowania stosunku obecności do nieobecności użyty został diagram kołowy znajdującej się w prawej części okna. Jego schemat kolorystyczny jest zawsze identyczny i w jednoznaczny sposób identyfikuje obecności (kolor zielony) oraz nieobecności (czerwony kolor).

11. Możliwa rozbudowa aplikacji.

Zaimplementowana wersja aplikacji udostępnia użytkownikom jedynie podstawowe funkcjonalności, niezbędne do prawidłowego spełniania jej podstawowej funkcji - rejestrowania obecności na zajęciach. Idąc o krok dalej można rozważyć teoretyczne ścieżki rozwoju aplikacji i funkcjonalności dodatkowe usprawniające i poszerzające zakres wykorzystania oprogramowania. Poniżej opisano przykładowe funkcje, które mogłyby zostać zaimplementowane w ramach poszczególnych modułów.

- Sprawdzanie listy obecności:
    - Automatyczne przenoszenie do statystyk studenta

Po dwukrotnym wciśnięciu na pojedynczy wiersz, przechodzimy do widoku statystyk studenta wraz z filtrowaniem dotyczącym sprawdzanych zajęć.

- Czasami dany prowadzący prowadzi zajęcia w dwóch salach jednocześnie. W takim przypadku powinna być możliwość jednoczesnego logowania listy obecności w dwóch grupach. Na chwilę obecną, program nie jest przystosowany do podejmowania takich akcji.

- Statystyki studenta:
    - Automatyczne wysyłanie e-maili do danego studenta

W przypadku gdy w statystykach studenta zauważymy, że jakaś osoba nie uczęszcza na zajęcia, możemy wysłać jej automatyczny e-mail, w zawartości umieścimy ilość obecności oraz nieobecności.

- W związku z tym, że wiemy ile dane zajęcia mają godzin (np. strona https://fee.put.poznan.pl/index.php/studia/plan-studiow-stacjonarne)

        - możemy dodać pole: maksymalna ilość zajęć, system mógłby wtedy sprawdzić czy w danych tygodniach wypadają nam zajęcia (stałe święta) oraz automatycznie pomóc w organizacji kolokwiów (podać ostatnią przewidywaną datę zajęć) bądź poinformować, że np. w przyszłym tygodniu nie ma zajęć.
    - Wraz z wiedzą ile jest maksymalnych zajęć oraz ilości nieobecności studenta
        - możemy poinformować go wysyłając automatyczny e-mail, że od następnych zajęć musi pojawić się na wszystkich, gdyż w przeciwnym wypadku będzie nieklasyfikowany
    - Przechowywanie notatek odnośnie danych studentów, wraz z datą realizacji
        - jeżeli np. mamy pomóc danemu studentowi z materiałem, zapisujemy sobie notatkę wraz z datą. Notatka pojawiłaby się w postaci wyskakującego okna, dzięki któremu moglibyśmy przejść do statystyk studenta.
    - Eksportowanie danych studentów
        - funkcjonalność, która pozwoli nam wyeksportowanie danych odnośnie studenta bądź studentów do pliku typu: csv

12. Opis najważniejszych implementacji wraz z krótkim kodem źródłowym.

- logika wątku pobocznego odpowiedzialnego za nasłuchiwanie - https://pastebin.com/tUV03E3x.

W celu zabezpieczenia przed nieskończoną pętlą - użyta została zmienna atomowa. Jeżeli wykryta zostanie zmiana (prowadzący skończy sprawdzać obecność), w takim przypadku wątek wyjdzie z pętli oraz chwilę później zakończy swoje działanie.

Logika działania poniższej funkcji wygląda następująco:

- Jeżeli nie ma karty, to czekaj

- Jeżeli ktoś włożył kartę, to pobierz informacje oraz uzupełnij tabelę obecności
- Jeżeli jest włożona karta, to czekaj aż zostanie wyjęta

Powyższy sposób oczywiście nie jest najbardziej optymalny, jednakże na potrzebę projektu jest wystarczający.

Wątek poboczny został wykorzystany ze względów praktycznych: w czasie gdy studenci podchodzą oraz rejestrują swoją obecność, prowadzący może przejrzeć statystyki danego studenta.

```
running = new AtomicBoolean(Boolean.TRUE);
try {
   TerminalFactory factory = TerminalFactory.getDefault();
   List<CardTerminal> terminals = factory.terminals().list();
   System.out.println("Terminals: " + terminals);

   CardTerminal reader = terminals.get(0);

   while (running.get()) {
       while (!reader.isCardPresent()) {
           if (!running.get()) {
               return;
           }
       }

       Card card = reader.connect("*");
       card.beginExclusive();
       CardChannel channel = card.getBasicChannel();
       ResponseAPDU resp = channel.transmit(new CommandAPDU(128,
202, 0x9f, 0x7f, 256));
       byte[] data = resp.getData();
       byte[] ICSerialNumber = Arrays.copyOfRange(data, 15, 19);
```

```
        MainScreenController.addRow(hexToString(ICSerialNumber));

        while (reader.isCardPresent()) {
            if (!running.get()) {
                return;
            }
        }
    }
```

- rejestrowanie obecności studenta na podstawie jego karty (String cardId) - https://pastebin.com/8GCtrNJp.

Po zalogowaniu, pobierana jest lista wszystkich studentów dla danego prowadzącego.

Jeżeli karta, która została włożona nie znajduje się na liście studentów - wyświetlamy komunikat konsolowy, “Dodaj nowego studenta”. Jest to widoczne jedynie z punktu widzenia programistów.

W przypadku gdy student znajduje się na liście studentów, wtedy pobieramy jego dane oraz dodajemy go do listy obecności. Jeżeli natomiast został wpisany po raz kolejny, to nie dodajemy go do listy obecności (“return;”)

```
boolean registeredStudent = studentHashMap.containsKey(cardId);
if (!registeredStudent) {
    System.out.println("Dodaj nowego studenta");
    return;
}
for (Map.Entry<String, StudentDto> entry :
studentHashMap.entrySet()) {
    if (entry.getKey().equals(cardId)) {
        StudentDto student = entry.getValue();
```

```
        student.setIsLate(areStudentsLate);
        student.setCardId(entry.getKey());
        for (StudentDto student_in_list : students) {
            if (student_in_list.getId().equals(student.getId())) {
                return;
            }
        }
        students.add(student);
    }
}
```

- statystyki studenta - https://pastebin.com/f6L0rtSW.

```
String name = nameAndSurname.split(" ")[0];
String surname = nameAndSurname.split(" ")[1];
Student student = studentRequest.findByFirstNameAndLastName(name,
surname);

Integer frequencyCounter =0;
Integer absenceCounter =0;
Integer totalPresenceCounter =0;

frequencyCounter=presenceOnLectureRequest.findAllByStudent_IdAndLect
ure_Id(student.getId(), temporaryLectureId).size();

List<PresenceOnLecture> totalPresenceList =
presenceOnLectureRequest.findAllByLecture_Id(temporaryLectureId);
Set<String> uniqueDates = new HashSet<>();
for (PresenceOnLecture p : totalPresenceList) {
    uniqueDates.add(p.getPresenceDate().toString() +
p.getHourTime());
}
totalPresenceCounter = uniqueDates.size();
```

```
if(!(totalPresenceCounter < frequencyCounter)){
   absenceCounter = totalPresenceCounter - frequencyCounter;
}

ObservableList<PieChart.Data> pieChartData =
FXCollections.observableArrayList(new PieChart.Data("Obecność",
frequencyCounter), new PieChart.Data("Nieobecność",
absenceCounter));

diagram.setData(pieChartData);
presentTextField.setText(Integer.toString(frequencyCounter));
absentTextField.setText(Integer.toString(absenceCounter));
totalPresenceTextField.setText(Integer.toString(totalPresenceCounter
));
```