POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRYCZNY
KIERUNEK: INFORMATYKA

PROJEKT REALIZOWANY W RAMACH PRZEDMIOTU:
PODSTAWY TELEINFORMATYKI

SPRAWDZANIE OBECNOŚCI W LABORATORIUM Z WYKORZYSTANIEM LEGITYMACJI STUDENCKIEJ

Prowadzący:
mgr inż. Przemysław Walkowiak

Autorzy:
Michał Gozdek
Dominik Kaczmarek
Hubert Kaszuba
Konrad Michalak

Poznań 2018

SPIS TREŚCI

1. WSTĘP

Legitymacja studencka to nieodłączny przedmiot w portfelu każdego studenta. Karta świadczy o statusie studenta, zapewnia mu zniżki, pozwala na identyfikację elektroniczną, a także pozwala na zapisanie dodatkowych usług jak karta płatnicza, czy system Poznańskiej Elektronicznej Karty Aglomeracyjnej ułatwiający podróżowanie. Wszystkie te informacje są przechowywane w pamięci małego układu scalonego, a ich zapis i odczyt zapewnia niewielki procesor zawarty na każdej karcie elektronicznej. Dzięki niemu nawet proste czytniki, są w stanie komunikować się kartą i dostarczać funkcjonalność zapisaną na nośniku.

W dobie przedstawionych wcześniej faktów, jako zespół zdecydowaliśmy się na zaprojektowanie systemu korzystającego z funkcjonalności elektronicznej legitymacji studenckiej, tworząc program ułatwiający i zapewniający dodatkową wygodę w pracy wykładowców. Sprawdzanie obecności, bo ono jest głównym tematem projektu, jest często procesem pomijanym przez prowadzących zajęcia, lub generującym pewne problemy, jak możliwość zagubienia kartki z obecnością, czy dociekanie na jakich zajęciach student rzeczywiście był, a na jakich nie. System automatyzacji przebiegu weryfikacji osób na sali, który chcemy zaproponować mógłby w znaczny sposób pomóc w rozwiązaniu tych problemów, zrzucając odpowiedzialność z prowadzącego i przekazując ją oprogramowaniu.

Głównymi założeniami projektu było dostarczyć program, który będzie prosty i przejrzysty. System miał rozróżniać wykładowców, dzięki systemowi loginów i haseł, umożliwiać ułożenie własnego harmonogramu zajęć i za kliknięciem jednego przycisku zbierać informację o obecności studenta jedynie za przyłożeniem legitymacji. Miejscem składowania danych miała być baza danych umieszczona w chmurze, aby dane były uniwersalnie dostępne w każdej sali wykładowej z dostępem do internetu. Chcieliśmy również, aby wykładowca miał możliwość eksportu danych do pliku, w celu wydrukowania listy, czy przetworzenia jej w indywidualny sposób przy użyciu arkusza kalkulacyjnego lub edytora tekstu. Jedynym wymaganiem, który stawiamy potencjalnemu użytkownikowi programu jest konieczność posiadania czytnika kart elektronicznych. Jego posiadanie jest wymogiem, aby skorzystać z głównej funkcjonalności programu, jednakże jego brak nie odbiera możliwości realizacji innych zastosowań.

Podstawowym powodem wyboru przez nas takiego tematu na aplikacje, był fakt, że wydał się bardzo bezpieczny dla zespołu. Większość narzędzi, które posłużyły do zbudowania aplikacji, jest dobrze znana i przetestowana przez każdego uczestnika projektu, ponieważ podobne programy były realizowane w ramach studiów. Dodatkowo tego typu aplikacje biznesowe, czyli integrujące pewien interfejs użytkownika, z bazą danych, są bardzo popularnym typem prac zlecanym programistom. Realizacja projektu pozwoliła nam doszlifować umiejętności, jeśli kiedyś przyszło by nam pracować nad tego typu systemami. Ciekawym urozmaiceniem, jakie było kolejnym czynnikiem wpływającym na wybór, była praca z kartami elektronicznymi, a dokładniej sposób w jaki można je odczytywać i wykorzystać do własnych celów. Karty elektroniczne można aktualnie znaleźć w wielu miejscach pracy, laboratoriach, czy urzędach, przez co atrakcyjne wydało się poznanie

mechanizmów za nią stojących. Nie jesteśmy w stanie przewidzieć, czy jeszcze kiedyś będziemy mieli sposobność, aby wykorzystać tę technologię w projekcie, dlatego chcieliśmy skorzystać z danej nam okazji.

2. PODZIAŁ PRAC

Napisanie większego projektu, o wielu modułach składających się na jeden, spójny system, związane jest z podziałem pracy nad poszczególnymi elementami pomiędzy wszystkich członków zespołu. Podczas planowania architektury aplikacji sprawdzania obecności udało się w naturalny sposób rozdzielić zadania między dwa mniejsze zespoły projektowe, które mogły działać w częściowej separacji. Takie rozwiązanie pozwoliło na uniknięcie chaosu, jaki może wkraść się podczas tworzenia rozległego oprogramowania, komunikacja odbywała się parach zamiast na zasadzie “każdy z każdym”. Pary mogły bezpośrednio ze sobą współpracować, w tym zgłaszać i poprawiać znalezione błędy. Problemem takiego podejścia było wymuszenie projektowania częściowo bez znajomości struktury modułu pary drugiej. Oczywiście komunikacja odbywała się pomiędzy parami, jednak w stopniu dużo mniejszym, niż standardowo. Jej głównym celem było poznanie w jakiej formie będą dostarczane dane modułów kolegów, aby móc stworzyć funkcje imitujące funkcjonalność komponentu i jednocześnie nie tworzyć przestojów podczas produkcji oprogramowania.

Grupy, na które podzielono pracę wyglądały następująco:

- **Dominik Kaczmarek**, **Konrad Michalak -** odpowiadali za realizację działania mechanizmu wczytywania legitymacji studenckich, przetwarzania obecności, przygotowanie jej do wysyłki do bazy danych, a także projekt i logikę działania graficznego interfejsu użytkownika. W tym implementacja procesu generowania raportów, eksportu obecności studentów do pliku, oraz projekt interfejsu ustalania harmonogramu zajęć.
- **Michał Gozdek**, **Hubert Kaszuba -** byli odpowiedzialni za projekt, oraz budowę bazy danych wykorzystywanej przez aplikację. Ich zadaniem było osadzenie bazy w chmurze Microsoft Azure i zintegrowanie jej z aplikacją kliencką, czyli dostarczenie funkcji realizujących konkretne zapytania wykorzystywane przez interfejs użytkownika. Dodatkowo zaprogramowali mechanizm logowania wykładowców do aplikacji.

Szczegółowy podział zadań pomiędzy wszystkich członków zespołu, z uwzględnieniem konkretnych funkcjonalności, został przedstawiony w punkcie następnym, czyli 2.1.

2.1 Szczegółowy podział zadań

Podział na pary projektowe przedstawiony w poprzednim punkcie jest oczywiście pewnym poglądowym przedstawieniem zakresu prac. W obrębie danej pary również musiał

nastąpić podobny podział, aby uporządkować proces wytwarzania oprogramowania. Poniżej zostały przedstawione zadania powierzone członkom zespołu.

Dominik Kaczmarek

Problem opracowywany przez członka grupy dotyczył głównie pracy nad zrozumieniem sposobu działania czytnika kart korzystających z interfejsu "Smart Card". Do elementów zadania należała praca wejścia związana z poznaniem technologii kart elektronicznych, sposobu przetrzymywania informacji na chipach, a także możliwościach uzyskiwania z nich danych. W kolejnym kroku członek grupy musiał zaimplementować moduł pozwalający na komunikację z kartą poprzez czytnik kart i rozbiór przekazanych danych, aby uzyskać informacje o imieniu, nazwisku i indeksie studenta. Następnym zadaniem było zintegrowanie modułu z interfejsem użytkownika i przekształcenie uzyskiwanych danych do postaci krotek tzw. obecności, czyli struktury zawierającej informacje o studencie, aktualnie odbywanych zajęciach i czasie odbicia karty, gotowych do przekazania bazie danych.

Kolejne zadania podjęte przez członka zespołu były związane z programowaniem logiki przycisków połączonych z aktywacją procesu odczytywania kart, oraz jego zakończenia, a także wyświetlaniem odbitych kart. Dana osoba odpowiedzialna była również za utworzenie interfejsu edycji pojedyńczych obecności i ich aktualizacja na bieżąco.

Do ostatniej grupy zadań należała implementacja zakładki interfejsu odpowiedzialnej za generowanie raportów/eksport danych o obecnościach do postaci plików tekstowych w formacie csv, bądź pdf. W ramach tego zagadnienia należało zaprojektować interfejs użytkownika, a także zaimplementować logikę, odpowiedzialną za przetwarzanie krotek obecności do postaci tekstowej i zapisanie ich w nowym pliku.

Konrad Michalak

Michał Gozdek

Hubert Kaszuba

3. OFEROWANE FUNKCJONALNOŚCI

W tym rozdziale przedstawione zostały oferowane przez aplikacje funkcjonalności, które udało się zaimplementować. Opisy podzielone są na zakładki realizujące daną czynność. Sama obsługa zakładek została przedstawiona w rozdziale 7. natomiast, już teraz wprowadzona zostaje ich koncepcja, ponieważ pozwoliła ona w naturalny i logiczny sposób podzielić pracę pomiędzy wszystkich członków zespołu. Każda zakładka pozwala na realizację różnych funkcjonalności, a ich lista , oraz opis zostały przedstawione poniżej.

Zakładka tytułowa (główna)

- Możliwość sprawdzenia obecności przy pomocy legitymacji studenckich – aplikacja umożliwia po naciśnięciu guzika włączyć aktywny tryb pobierania informacji z legitymacji studenckich. Przy każdym przyłożeniu unikalnej karty elektronicznej przez studenta zostanie zarejestrowana obecność. Karta pobiera imię, nazwisko, oraz indeks i zapisuje wraz z czasem odbicia.
- Opcja edytowania pojedyńczych obecności – program pozwala po zarejestrowaniu obecności przez studenta, oznaczyć go jako spóźnionego, jeśli nie zdążył na początek zajęć, a także dodać notatkę dotyczącą jego zachowania, czy innych powiązanych wydarzeń.

Zakładka projektowania harmonogramów

Zakładka przeglądania obecności

Zakładka generowania raportów

- Możliwość podglądu wygenerowanego raportu – przed zapisaniem aplikacja pozwala na podgląd danych, które zostaną zapisane w pliku. Wygenerowana tabela pojawia się zaraz obok. Pozwala to na upewnienie się użytkownikowi, czy kontrolery odpowiedzialne za wybór zajęć, a także ich zakres zostały wybrane poprawnie.
- Przedstawienie tabeli obecności w dwóch wersjach – system udostępnia dwie wersje wizualizacji obecności m.in. w postaci klasycznej tabeli składającej z krotek pobranych prosto z bazy danych. Taki sposób może okazać się lepszy jeśli wykładowca woli samemu przetworzyć uzyskane dane np. przy pomocy arkusza kalkulacyjnego. I w drugiej postaci, gdzie obecności są przedstawione jako siatka, której wartości kolumn to dni zaplanowanych zajęć, a wiersze informują o statusie obecności studenta. Wersja bardziej atrakcyjna wizualnie, nadająca się np. do druku.
- Eksportu do pliku w dwóch formatach – użytkownika dostaje możliwość zapisu danych w bardziej interesującym go formacie. Format tekstowy *csv* nadaje się do dalszego indywidualnego przetwarzania danych o obecnościach, natomiast format *pdf* może lepiej się sprawdzić, jeśli użytkownik zdecyduje się wydrukować dokument.

4. WYKORZYSTYWANE TECHNOLOGIE

// wybrane technologie wraz z uzasadnieniem dlaczego,

5. ARCHITEKTURA ROZWIĄZANIA

W tym rozdziale została przedstawiona architektura rozwiązania, w tym opis poszczególnych części oprogramowania, a także udokumentowanie sposobu, w jaki zostały połączone. Struktura programu w założeniu została podzielona na moduły. Każdy z nich pokrywał się z zadaniem przydzielonym danemu członkowi drużyny. Dzięki zastosowaniu takiego podejścia członkowie mogli pracować niezależnie nad swoimi komponentami,

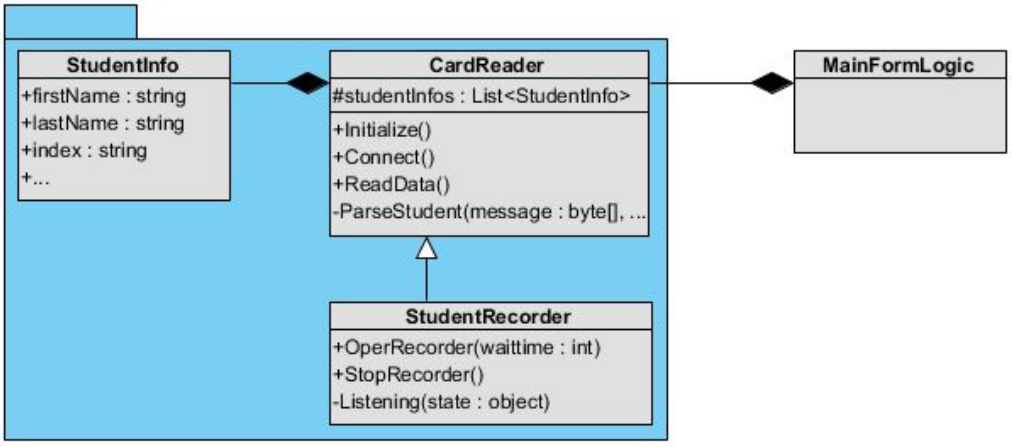
indywidualnie wybierając sposób i czas realizacji. Uproszczony schemat aplikacji można przedstawić na diagramie.

// tutaj diagram aplikacji

// opis diagramu

Moduł czytnika kart

Moduł odpowiedzialny za komunikację opierał się na trzech klasach: CardReader, StudentRecorder i pomocniczej StudentInfo. Relacje je wiążące można przedstawić na schemacie.



Schemat. Diagram klas opisujący moduł odpowiedzialny za obsługę czytnika.

Jak przedstawiono na diagramie klasa StudentRecorder dziedziczy po klasie CardReader, przystosowując jej metody do obsługi w klasie logiki interfejsu. Klasa bazowa jest odpowiedzialna za inicjalizację połączenia z czytnikiem kart (metoda Initialize), a także połączenie bezpośrednio z kartą elektroniczną poprzez metodę Connect. Metody ReadData i ParseStudent służą bezpośrednio do odczytu informacji i interpretacji danych o studencie. W momencie wywołania metody ReadData, wysyłane są poprzez interfejs PC/SC do czytnika, na stałe zapisane w tablicy bajtów komendy APDU, które służa do komunikacji z kartą. W odpowiedzi karta odsyła tablicę bajtów zawierającą potrzebne dane rozdzielone losowym szumem. Aby wydobyć imię, nazwisko i indeks cała tablica zamieniana jest na łańcuch znaków, a następnie rozdzielana poprzez wyrażenie regularne. Metody w klasie do działania wymagają używania ich w odpowiedniej kolejności, nad czym czuwa flaga stanów klasy, reprezentowana przez typ wyliczeniowy enum State.

Założeniem było, aby klasa CardReader była niezależna i udostępniała metody pozwalające na odczytanie pojedyńczych informacji o studencie. Dopiero klasa StudentRecorder rozszerza jej zdolności o automatyzację tego procesu i pozwala na odczyt ciągły przez określony czas. Celem takiego zabiegu było rozdzielenie podstawowych czynności od bardziej złożonych, co w zamyślę miało ułatwić lokalizacje potencjalnych błędów, lecz co ważniejsze pozwoliło na elastyczne dostosowanie metod klasy do napotkanego interfejsu.

Klasa StudentRecorder stanowi most pomiędzy interfejsem, a obsługą czytnika. W razie rozpoczęcia przez użytkownika procesu sprawdzania obecności, obiekt tworzy wątek przy użyciu metody OperRecorder, na którym nasłuchiwany jest moment przyłożenia karty przez studenta. Każde przyłożenie karty jest związane z pobraniem informacji, utworzeniem obiektu StudentInfo i sprawdzeniem, czy istnieje już na lokalnej liście. Jeśli proces zostanie zakończony sukcesem to wywoływane jest wydarzenie ReadedWithSuccess, które może zostać przechwycone w klasie logiki interfejsu użytkownika. Metoda StopRecorder jest odpowiedzialna za zakończenie wątku, w przypadku zakończenia sprawdzania obecności przez użytkownika.

StudentInfo jest klasą pomocniczą, tymczasowo przechowuje informacje o studencie pobrane z legitymacji. Dodatkowo wzbogaca je o informacje przydatne w procesie przekazywania do bazy danych, jak flaga oznaczająca spóźnienie, pole tekstowe na notatkę, a także znacznik czasowy.

Moduł generowania raportów

Moduł generowania raportów zawiera się głównie w klasie TextfileConstructor, gdzie odpowiedzialny jest za przyjęcie tablicy obecności i zamiany jej na ciąg znaków. Mimo to część odpowiedzialna za wygenerowanie tablicy gotowej do przekazania klasie, znajduje się w metodach zawartych w klasie logiki obsługi interfejsu MainForm.

Klasa w zależności od wybranego sposobu zapisu buduje plik inaczej. Chcąc zapisać go w formacie *csv* klasa wykorzystuje obiekt StringBuilder, który można znaleźć w bibliotekach wbudowanych do języka C#. Moduł przekształca odpowiednio nazwy kolumn do postaci jednego nagłówka typu *string* i następnie powtarza to dla każdego wiersza w tabeli. Po przeskoczeniu przez całą tablicę, plik zostaje zapisany w formacie *csv*.

W przypadku zapisu w formacie pdf, klasa korzysta z biblioteki zewnętrznej iText przystosowanej dla języka C#. Biblioteka wykorzystuje własną strukturę do przechowywania tabel o nazwie PdfPTable, do której w pętli przepisywana jest tablica obecności. Po zakończeniu tej operacji, tworzony jest obiekt Document, który odpowiada strukturze pliku PDF. Przy użyciu FileStream obiekt przepisanej tablicy jest podawany dokumentowi i zapisywany w odpowiednim formacie na dysku.

Klasę TextfileContructor wspiera obiekt TextfileContructorParams, który przechowuje informację o wybranych przez użytkownika parametrach generacji pliku. Obiekt aktualizuje swoje pola na podstawie wartości wpisanych, bądź wybranych w interfejsie użytkownika, a następnie jest zapisywany we wnętrzu klasy TextfileConstructor, gdzie może bezpośrednio wpływać na sposób budowania pliku. Klasa zawiera informację o wybranych zajęciach, datach wyznaczających zakres dni, a także dodatkowych elementach umieszczonych w pliku.

// architektura rozwiązania (jak jest zbudowane),

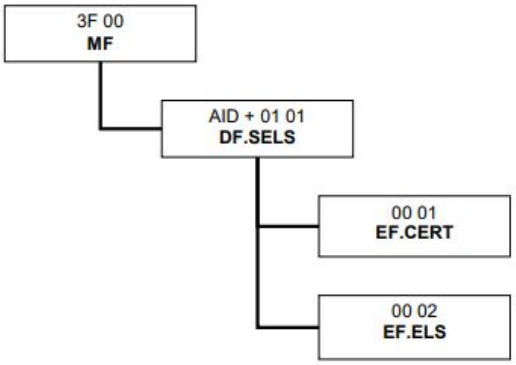
6. NAPOTKANE PROBLEMY

Każdy programista podczas pracy nad projektem, o większej złożoności musi w końcu trafić na różnego rodzaju problemy. Umiejętność dobrego podejścia i radzenia sobie z nimi, świadczy w dużej mierze o zdolnościach osoby kodującej. Tak też w tym rozdziale przedstawione zostały problemy na jakie natknęli się członkowie zespołu, podczas projektowania swoich modułów, a także rozwiązania na jakie udało im się natknąć. Rozdział został podzielony na mniejsze sekcje związane z każdym uczestnikiem projektu.

Dominik Kaczmarek

Jednym z podstawowych problemów, jaki pojawił się podczas implementacji modułu czytnika kart było zrozumienie sposobu porozumiewania się z chipem. Wszystkie karty pracujące w tej technologii posiadają interfejs PC/SC, który służy do komunikacji. Na szczęście istnieje wiele bibliotek dla języka C# realizujące takie zadanie i można je wykorzystać w projekcie. Wykorzystana ostatecznie została biblioteka pcsc-sharp napisana przez użytkownika danm, która jest nakładką na bibliotekę wbudowaną w system windows.

Prawdziwą jednak przeciwnością losu okazało się poznanie komend, na które reaguje karta i sposób ich wykorzystania, aby wyciągnąć potrzebne dane. Język komend, który to realizuje nazywa się APDU i operuje na tablicach bajtów, które muszą zostać wysłane w odpowiedniej kolejności. Dokumentacja języka zawarta w ISO 7816-4 jest napisana dość przejrzyście i pozwoliła na zmuszenie karty do odsyłania pozytywnych odpowiedzi. Mimo to dodatkowym problemem okazała się z pozoru prosta struktura katalogów. Wszystkie katalogi są połączone strukturą drzewiastą, której fragment można przedstawić na poniższym rysunku:



Rys. Rysunek przedstawia fragment struktury katalogów, który można znaleźć w pamięci kart elektronicznych.

Korzeniem drzewa jest plik oznaczony jako MF (ang. *Master File*), jego ważną właściwością jest posiadanie stałego kodu dostępu, do którego można się odwołać. Konkretnymi katalogami w tej strukturze są pliki oznaczone jako DF (ang. *Dedicated File*). Posiadają one własny kod, który jest unikalny dla danej funkcjonalności karty. W plikach oznakowanych jako EF (ang. *Elementary File*) można znaleźć zapisane informacje w postaci tablicy bajtów, czyli miejsce docelowego, z którego trzeba było pobrać informację.

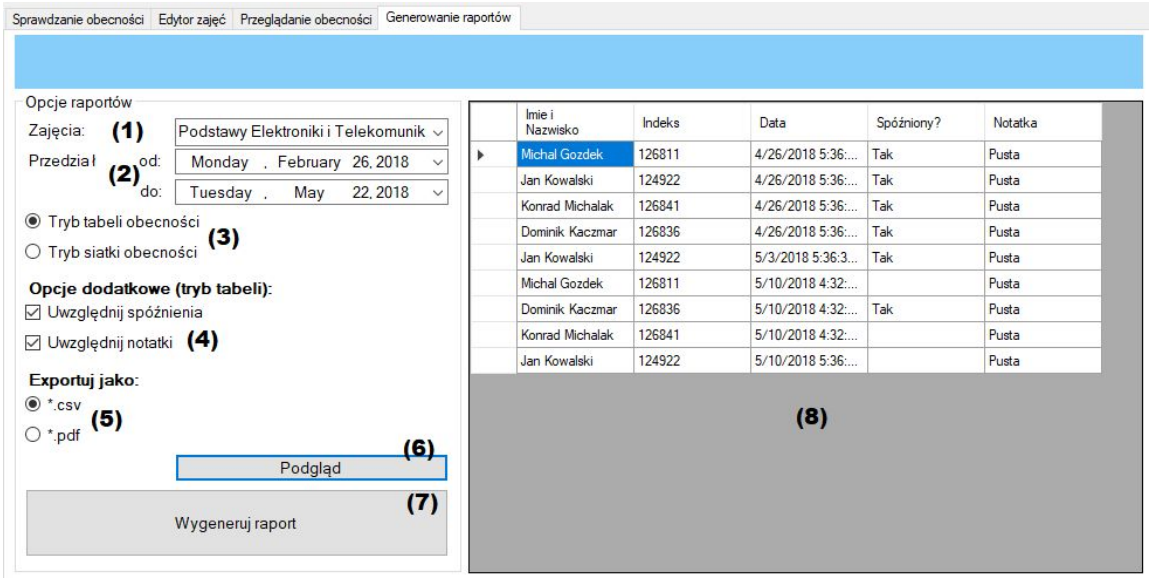
Nigdzie nie istnieje publiczna lista kodów związanych z funkcjami karty, a przeszukiwanie całej jest trudne, ponieważ może trwać nawet parę godzin, co było niedopuszczalne dla systemu, który powinien działać w czasie rzeczywistym. Z tego powodu podjęto kroki, aby poznać konkretny kod miejsca przetrzymywania informacji o studencie. Niestety kody nie były zawarte nigdzie na stronie uczelni, ani w plikach przez nią udostępnionych. Rozwiązanie problemu przyszło dość nieoczekiwanie, udało znaleźć się prezentację z laboratorium z obsługi technologii związanych z legitymacją studencką, w którym były podane dokładne informacje o lokalizacji danych. Udało się skonstruować zapytania i zapisać na stałe w obiekcie CardReader. Takie rozwiązanie zabrało trochę uniwersalności aplikacji, ponieważ sprawiło, że jedynie legitymacje zarejestrowane po roku 2015, czyli te, których informacje zostały zapisane dokładnie w znalezionym katalogu, działają z nią poprawnie, jednak sam odczyt jest dużo szybszy.

// interesujące problemy i rozwiązania ich na jakie się natknęliście

7. INSTRUKCJA UŻYTKOWANIA

// napisać jakiś wstęp co znajduje się w rozdziale

Obsługa zakładki generowania raportów



Rys. Podgląd zakładki generowania raportów, wraz z oznaczeniami.

Na rysunku można zobaczyć jak prezentuje się zakładka generowania raportów po włączeniu programu. Numery naniesione na obraz stanowią punkt odniesienia dla akcji, które może wykonać użytkownika. Opis działania poszczególnych fragmentów został przedstawiony poniżej:

(1) Zakładka rozwijana, pozwala wybrać zajęcia, przyporządkowane do zalogowanego wykładowcy, na podstawie, których będzie generowany raport.

(2) Wybór daty, od której i do której będą brane pod uwagę zajęcia podczas generowania raportu.
(3) Wybór trybu w jakim zostanie zwizualizowana tablica obecności.
(4) Opcje dodatkowe, decydują czy w tabeli zostaną zawarte kolumny informujące o opóźnieniach, bądź notatkach.
(5) Wybór formatu do którego zostanie wyeksportowany plik, csv lub pdf.
(6) Pozwala na wygenerowanie w polu **(8)** podglądu tabeli, przed zapisaniem do pliku.
(7) Przycisk wygenerowania raportu, po naciśnięciu zapisze plik w wybranym przez użytkownika miejscu.

// instrukcja użytkowania aplikacji

8. ŹRÓDŁA

Wykorzystane biblioteki:

- Biblioteka pcsc-sharp napisana przez użytkownika danm: https://github.com/danm-de/pcsc-sharp
- Biblioteka itextsharp udostępniona przez firmę iText: https://github.com/itext/itextsharp

Wykorzystane dokumentacje:

- Wykaz znaczeń kodów odpowiedzi dla języka APDU: https://www.eftlab.co.uk/index.php/site-map/knowledge-base/118-apdu-response-list
- Dokumtacja dla komend APDU zawarta w ISO 7816-4: http://cardwerk.com/smart-card-standard-iso7816-4-section-6-basic-interindustry-commands/
- Dokumentacja biblioteki pcsc-sharp: https://danm.de/docs/pcsc-sharp/PCSC/SCardReader.html

Inne materiały:

- Prezentacja z laboratorium kart elektronicznych, źródło rysunku struktury katalogów: http://www.mcp.poznan.pl/wp-content/uploads/2015/11/20151020_LabPKE_03-ELS.pdf
- Informacje o systemie plików wykorzystywanym przez kartę: http://www.makdaam.eu/2008/04/czytanie-els-przez-interfejs-stykowy/

// wszelkie biblioteki, dokumentacje, artykuły z których korzystaliście podczas pisania