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 pojedynczych 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

Zadaniem członka grupy było zaprojektowanie i zaimplementowanie interfejsu użytkownika. W początkowym okresie prac zaprojektował wstępny szkielet aplikacji z podziałem na zakładki odpowiedzialne za sprawdzanie obecności, edycję zajęć oraz przeglądanie obecności. Wewnątrz zakładki sprawdzania obecności, znalazły się takie pola jak zegar systemowy, informacja o aktualnych i następnych zajęciach użytkownika zalogowanego w systemie, informacja o zalogowanym użytkowniku oraz przyciski odpowiedzialne za rozpoczynanie/kończenie operacji sprawdzania obecności wraz odliczaniem czasu. Resztą funkcjonalności tej zakładki zajął się Dominik Kaczmarek.

Kolejnym zadaniem było stworzenie funkcjonalności edytora zajęć. Wewnątrz tej zakładki członek zespołu był zobowiązany do połączenia interfejsu z bazą danych i zapewnienia możliwości dodawania poszczególnych obiektów oraz wyświetlania listy zajęć.

Aby umożliwić edycję wyżej wymienionych obiektów, należało zaprojektować nowe okno zawierające odpowiednie filtry i przyciski wykonujące odpowiednie operacje edycji elementów w bazie danych.

Ostatnim zadaniem było zaimplementowanie zakładki przeglądania obecności na podstawie filtrów wprowadzonych przez użytkownika aplikacji. Ponownie wymagało to połączenia interfejsu użytkownika z bazą danych i wykonywania odpowiednich metod.

Michał Gozdek

W początkowym okresie prac nad projektem zajmował się wraz z Hubertem Kaszuba znalezieniem odpowiedniego serwisu zewnętrznego, opracowaniem projektu oraz implementacją bazy danych. Pracę nad bazą rozpoczął od zastanowienia się nad tym jakiego typu powinna ona być. (relacyjna czy nierelacyjna) Wybór padł na bazę relacyjną, a dokładniej na bazę SQL. W związku z tym konieczne było zastanowienie się jakie tabele oraz atrybuty powinna ona posiadać, a także w jaki sposób tabele mają być ze sobą powiązane. Następnie wraz z kolegą z grupy zajmującej się częścią bazodanową aplikacji wybrał serwis oferujący usługi chmury internetowej, na którym zaimplementowali wcześniej zaprojektowaną bazę.

W kolejnym etapie realizacji projektu zaprojektował oraz zaimplementował ekran logowania i rejestracji. Program mogą używać jedynie wykładowcy którzy są zarejestrowani.

Następnie zaczął współpracę z grupą zajmującą się interfejsem użytkownika. Dostarczał im potrzebne do działania programu funkcję. Zajął się między innymi implementacją dodawania sal oraz przedmiotów. Wraz z Konradem Michalakiem stworzył również ekran edycji zajęć pojedynczych. Członek ten miał także udział przy tworzeniu zakładki dotyczącej generacji raportów. Zaimplementował w niej jeden z algorytmów wyświetlania osób obecnych na danych zajęciach w określonym przedziale czasu.

Hubert Kaszuba

Na początku prac nad projektem, członek ten współpracował z Michałem Gozdkiem przy stworzeniu bazy danych. W pierwszej kolejności zrobili rozeznanie dotyczące typu bazy danych, która miała być używana w projekcie (relacyjna, czy nierelacyjna). Wybór padł na relacyjną bazę danych SQL. Następnie zaczęli szukać platform, które mogłyby udostępnić usługę składowania danych w chmurze. Ich uwagę przykuła platforma Microsoft Azure, która udostępnia wiele usług, w tym usługę stworzenia bazy danych SQL w chmurze oraz jej obsługę. Następnie członek ten zaprojektował wraz z Michałem Gozdkiem poszczególne tabele w bazie danych oraz połączenia między nimi. Gdy baza danych była już gotowa, członek ten postawił serwer na platformie Microsoft Azure, w którym umieścił stworzoną bazę danych.

W następnej fazie pracy nad projektem zajął się wygenerowaniem modelu bazy danych w projekcie. Następnie jego zadaniem było implementowanie funkcji do obsługi bazy danych. Współpracował z Dominikiem Kaczmarkiem, dostarczając mu algorytm, który pozwalał na dodawanie obecnych studentów do bazy danych. Stworzył algorytm, który, gdy prowadzący dodał nowe zajęcia realizowane w ramach prowadzonego przez Niego

przedmiotu, generował wszystkie zajęcia z tego przedmiotu, wraz z datami do końca semestru. Dzięki temu możliwe było zaimplementowanie harmonogramu dla prowadzącego.

W kolejnym etapie prac, nawiązał współpracę z Konradem Michalakiem, który zajmował się interfejsem użytkownika. Napisał dla niego algorytm, który zwracał aktualne zajęcia prowadzącego, oraz zajęcia następne w danym dniu. Algorytm ten został wykorzystany do wyświetlania tych informacji w interfejsie.

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.

Ekran rejestracji

- Możliwość rejestracji nowych użytkowników - aplikacje może użytkować jedynie osoba, która jest zarejestrowana w bazie danych programu. Ekran pozwala na dodanie nowych użytkowników. Mechanizm rejestracji sprawdza czy w bazie nie ma już użytkownika o podanym loginie. Jeżeli w bazie nie ma nikogo takiego a wprowadzane dane są poprawne to dodaje do bazy nowego użytkownika.

Ekran logowania

- Logowanie się do swojego konta - aplikacja sprawdza czy wprowadzane dane logowania, to jest login oraz hasło, są poprawne. Jeśli tak jest pozwala użytkownikowi na odpalenie części właściwej aplikacji. Mechanizm ten dba o to by z aplikacji korzystali jedynie użytkownicy zarejestrowani.

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 pojedynczych 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.

- Wyświetlanie przydatnych dla użytkownika informacji takich jak obecnie odbywające się zajęcia, zajęcia występujące jako najbliższe zajęcia w danym dniu, informacje o zalogowanym w serwisie użytkowniku.

Zakładka projektowania harmonogramów

- Możliwość tworzenia harmonogramów zajęć. W programie nazwana jest "Edytor zajęć". Jej funkcjonalność jest niezbędna do umożliwienia sprawdzania obecności. Zalogowany użytkownik powinien wprowadzić do bazy danych informacje o harmonogramie prowadzonych zajęć. Na tej podstawie aplikacja wie jakie zajęcia odbywają się w danej chwili oraz do jakiego miejsca w bazie danych zapisać studentów zapisujących się na listę obecności.
- Dodawanie informacji do bazy danych - zakładka podzielona jest na cztery części: dodawanie sal do bazy danych, dodawanie przedmiotu, dodawanie zajęć oraz lista zajęć zawierająca szczegółowe informacje.
- Część odpowiedzialna za dodawanie zajęć jest powiązana z przedmiotami oraz salą, gdyż wewnątrz wysuwanych list znajdują się właśnie te obiekty. W celu utworzenia zajęć w harmonogramie, użytkownik jest zobowiązany wybrać przedmiot jakiego dotyczą zajęcia, salę w której się odbędą, datę pierwszych zajęć oraz częstość występowania (co tydzień albo co dwa tygodnie), by na jej podstawie odpowiedni algorytm wygenerował zajęcia na przód.
- Lista zajęć wyświetla harmonogram prowadzącego. Można ją porównać do planu zajęć, zawierającego informacje o nazwie zajęć, sali w której się odbywają, godzinie, dniu tygodnia oraz dacie.
- Z perspektywy tej zakładki, użytkownik może przejść do szczegółowego edytora zajęć, który uruchomi się w nowym oknie.

Zakładka edytora zajęć

- Możliwość edycji pojedyńczych zajęć -jeżeli jakieś dane zostały wprowadzone niepoprawnie, bądź nastąpiła nagła zmiana w planie zajęć przykładowo, zajęcia się nie odbyły i musiały zostać przeniesione na inny dzień czy godzinę, użytkownik może wykorzystać tę zakładke w celu korekty błędu bądź aktualizacji danych.
- Aby użytkownik mógł łatwo znaleźć zajęcia dla których chce dokonać zmian, wewnątrz zakładki zaimplementowane zostały pola umożliwiające dokładne parametryzowanie takie jak możliwość wyboru zajęć dla których mają zostać wprowadzone zmiany czy przedział czasowy w jakim zajęcia się odbywają.
- Aktualizacji mogą ulec pola takie jak data zajęć oraz godzina. Możliwe jest również całkowite usunięcie informacji o zajęciach z bazy danych.

Zakładka przeglądania obecności

- Możliwość sprawdzenia obecności studentów odbytych zajęć - na podstawie zawartych w zakładce filtrów, użytkownik może sprawdzić jacy studenci byli obecni na konkretnych zajęciach i konkretnym dniu.
- Możliwość wyszukania wszystkich zajęć na jakich był dany student.

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 – użytkownik 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

W tym rozdziale zostały przedstawione technologie wykorzystane do stworzenia projektu wraz z ich opisem oraz uzasadnieniem, dlaczego członkowie zespołu zdecydowali się na ich użycie.

4.1 SQL

SQL jest strukturalnym językiem, który służy do tworzenia oraz obsługi bazy danych. Pozwala na pisanie zapytań, dzięki którym można umieszczać, modyfikować, usuwać oraz pobierać dane z bazy danych. Zespół zdecydował się na wykorzystanie tej technologii, ponieważ pozwala na tworzenie tabel, z odpowiednimi dla każdej, atrybutami, a następnie utworzenie relacji pomiędzy odpowiednimi tabelami. Wynikiem powyższych operacji jest relacyjna baza danych, na której opiera się cały projekt. Utworzenie odpowiednich tabel oraz związków pomiędzy nimi, przez zespół zajmujący się bazą danych, pozwoliło na utworzenie bazy danych, która przechowuje wszystkie, ważne dla projektu, informacje. Pozwala na ich dodawanie, usuwanie, modyfikację oraz pobieranie interesujących nas danych.

4.2 Windows Forms (.NET Framework)

Windows Forms jest biblioteką wizualną dostarczaną przez platformę .NET Framework, która pozwala na zaprojektowanie graficznego interfejsu użytkownika (ang. GUI, Graphical User Interface). Windows Forms zawiera elementy, służące do tworzenia aplikacji okienkowych, które z kolei nazywamy aplikacjami sterowanymi zdarzeniami. Oznacza to, że takowa aplikacja przez większość czasu oczekuje na zdarzenia wywołane przez użytkownika (np. wciśnięcie przycisku, wypełnienie formularza. Zespół zdecydował się na użycie tej technologii, ponieważ pozwala ona na stworzenie prostej i przejrzystej aplikacji okienkowej, złożonej z kolei z wielu zakładek, dzięki czemu zaimplementowane funkcjonalności aplikacji można było intuicyjnie rozlokować.

4.3 LINQ

LINQ (ang. Language Integrated Query) jest częścią technologii Microsoft .Net i jak sama nazwa wskazuje, dostarcza zintegrowane zapytania językowe, czyli umożliwia tworzenie zapytań do obiektów. Składnia języka jest prosta i przypomina SQL. Tworzenie zapytań LINQ pozwala na wydzielenie pewnej liczby obiektów według naszych potrzeb. Zespół sięgnął po tą technologię, ponieważ każdy element w bazie danych jest obiektem, a dzięki LINQ, członkowie zespołu mogli przy pomocy prostych i przejrzystych zapytań wybierać interesujące ich dane, by następnie je wyświetlić, lub na nich operować.

4.4 Entity Framework

Entity Framework jest narzędziem używanym do mapowania obiektowo-relacyjnego (ang. ORM, object-relational mapping). Generuje obiekty biznesowe oraz encje na podstawie wcześniej stworzonej bazy danych. Obiekt biznesowy jest encją w wielowarstwowej aplikacji, która działa w połączeniu z dostępem do bazy danych oraz warstwą logiki biznesowej służącą do przesyłania danych. Z kolei encja odnosi się do czegoś co jest unikatowe i istnieje samodzielnie (np. tabela bazy danych). W projekcie aplikacji do sprawdzania obecności z wykorzystaniem legitymacji studenckiej, Entity Framework został wykorzystany do wygenerowania obiektów biznesowych oraz encji na podstawie tabel i relacji z utworzonej wcześniej bazy danych SQL.

4.5 PC/SC

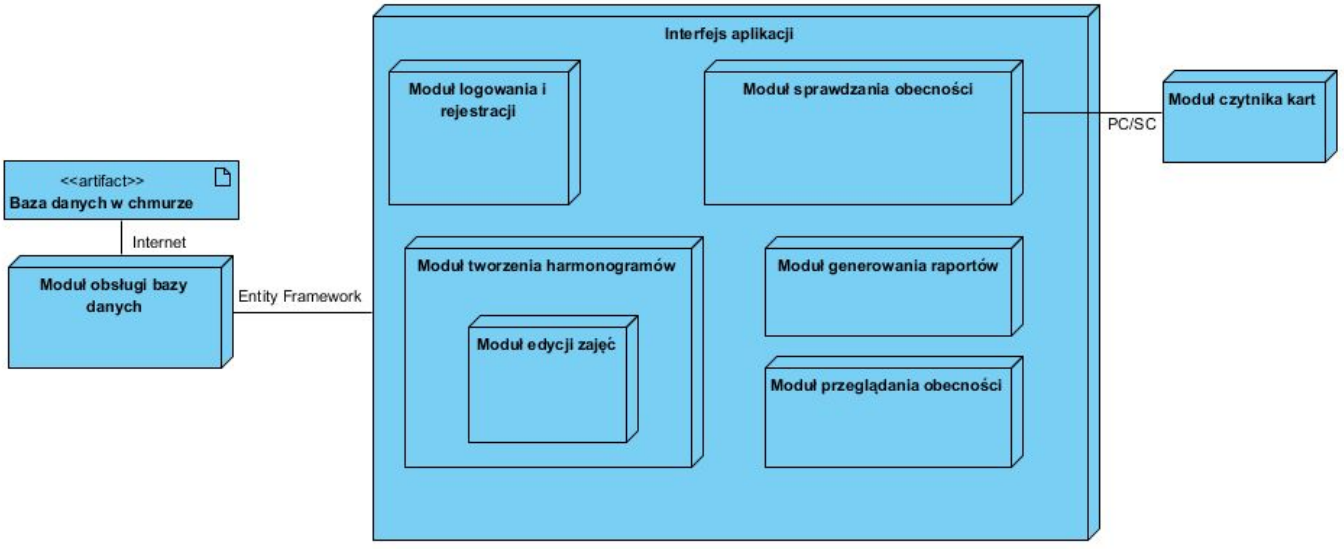
PC/SC jest standardowym interfejsem dostępu do kart elektronicznych, stworzony, aby ustandaryzować i ułatwić użycie ich w komputerach osobistych. Zespół skorzystał z nakładki na bibliotekę PCSC-sharp przeznaczoną dla języka C#. W projekcie została ona użyta do pobierania danych o studentach z legitymacji studenckich.

4.6 iText

iText jest darmową bibliotekę skierowaną do programistów, przy pomocy której możliwe jest generowanie dokumentów w formacie PDF. Biblioteka iText, w pierwotnej postaci, napisana była dla języka Java. Zespół wykorzystał jednak wersję przeportowaną do środowiska platoformy .NET - iTextSharp. Użycie jej w stworzonej aplikacji, pozwoliło na dodanie funkcjonalności generowania list obecności studentów na danych zajęciach do pliku PDF.

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.



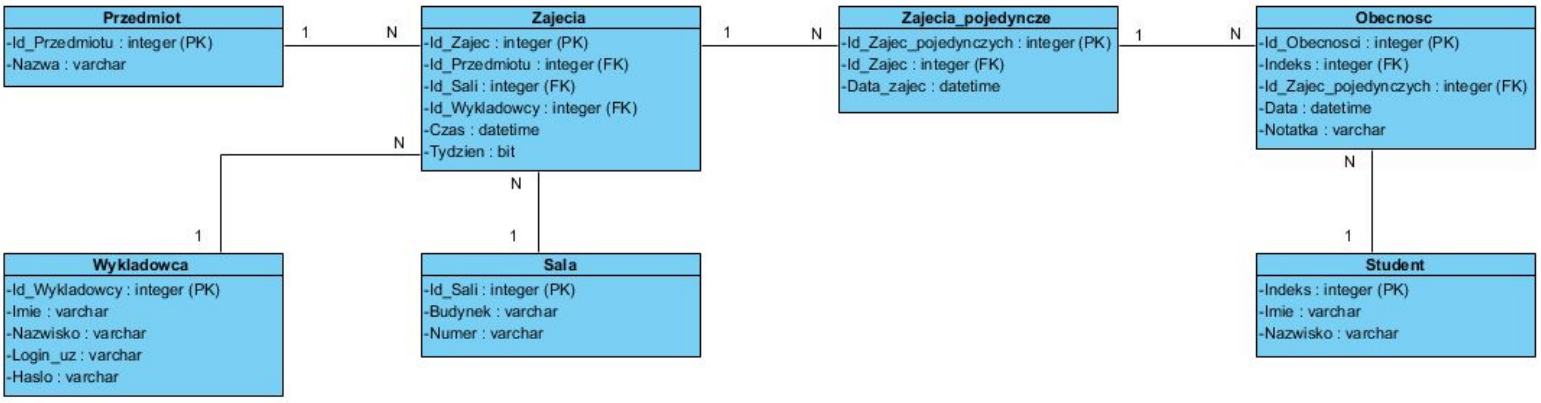
Rys. 1. Uproszczony schemat aplikacji przedstawiony na diagramie

Interfejs aplikacji składa się z wyżej przedstawionych modułów, tj. Moduł logowania i rejestracji, Moduł sprawdzania obecności, Moduł tworzenia harmonogramów i zawarty w nim Moduł edycji zajęć, Moduł generowania raportów i Moduł przeglądania obecności. Ponadto Moduł sprawdzania obecności jest połączony z Modułem czytnika kart przy pomocy interfejsu PC/SC. Cały interfejs aplikacji jest podłączony do Modułu obsługi bazy danych

przy pomocy Entity Framework i służy do komunikacji między bazą danych umieszczoną w chmurze, a modułami znajdującymi się w interfejsie.

5.1 Baza danych

W aplikacji wykorzystywana jest baza danych SQL. Pozwala ona na tworzenie tabel wraz z określonymi atrybutami najróżniejszych typów. SQL pozwala określić między innymi jakiej wielkości mają być określone atrybuty lub czy dany atrybut może być pusty czy nie. Każda z tabel posiada własny unikalny identyfikator. Dzięki atrybutom identyfikującym możliwe jest tworzenie relacji między poszczególnymi tabelami. Pozwala to na tworzenie dużych baz danych, w których tabele są wzajemnie od siebie zależne.



Rys. 2 .Schemat. Diagram bazy danych.

Nasza baza danych posiada 7 tabel (Przedmiot, Zajecia, Zajecia_pojedyncze, Wykladowca, Sala, Obecnosc, Student). Są one przedstawione na powyższym diagramie. Widać na nim również relacje zachodzące między poszczególnymi tabelami. Tak zaprojektowana baza okazała się przejrzysta i pozwoliła na zaimplementowanie wszystkich zaplanowanych funkcjonalności. Dokładny opis bazy znajduje się poniżej.

Rys. 3. Tabel Wykladowca

Powyższy rysunek przedstawia budowę tabeli Wykładowca. Jest ona tak naprawdę obiektem zalogowanego użytkownika. Tworzy podstawę aplikacji. Ma następujące atrybuty:

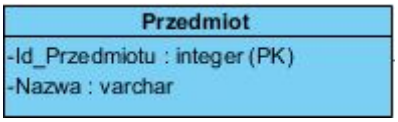
- Id_Wykladowcy : typ integer - jest kluczem głównym tabeli Wykladowca. Każdy wykładowca posiada inną wartość w tym atrybucie.
- Imie : varchar - atrybut, w którym znajduje się imię wykładowcy.
- Nazwisko : varchar - atrybut, w którym znajduję się nazwisko wykładowcy.

- Login_uz : varchar - login za pomocą którego użytkownik loguje się do programu. Wartość unikalna dla każdego wykładowcy/użytkownika.
- Haslo : varchar - hasło konieczne w procesie logowania. W bazie przechowywane jest zaszyfrowane w formie skrótu.

Rys. 4. Tabel Zajecia

Powyższy rysunek przedstawia budowę tabeli Zajęcia. Tworzy podstawę do generowania zajęć pojedynczych. Jest łącznikiem tabel: Sala, Przedmiot oraz Wykładowca. Ma następujące atrybuty:

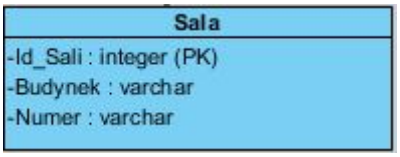
- Id_Zajec : integer - jest kluczem głównym tabeli Zajecia. Każdy obiekt Zajęcia posiada inną wartość w tym atrybucie.
- Id_Przedmiotu : integer - klucz obcy tabeli Przedmiot. Dzięki niemu może odwoływać się do obiektów tabeli Przedmiot.
- Id_Sali : integer - klucz obcy tabeli Sala. Dzięki niemu może odwoływać się do obiektów tabeli Sala.
- Id_Wykladowcy : integer - klucz obcy tabeli Przedmiot. Dzięki niemu można przypisać określonemu wykładowcy, dane zajęcia.
- Czas : datetime - dzień tygodnia oraz godzina odbywania się zajęć.
- Tydzień : bit - określa czy dane zajęcia odbywają się co tydzień czy co dwa tygodnie. Jest istotny przy generowaniu zajęć pojedyńczych.



Rys. 5. Tabel Przedmiot

Powyższy rysunek przedstawia budowę tabeli Przedmiot. Pozwala tworzyć nowe przedmioty do bazy danych. Ma następujące atrybuty:

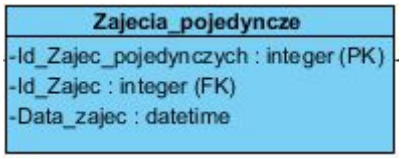
- Id_Przedmiotu : integer - jest kluczem głównym tabeli Przedmiot. Każdy obiekt Przedmiot posiada inną wartość w tym atrybucie.
- Nazwa : varchar - nazwa przedmiotu.



Rys. 6. Tabel Sala

Powyższy rysunek przedstawia budowę tabeli Sala. Pozwala na przechowywanie Sal w których odbywają się zajęcia. Ma następujące atrybuty:

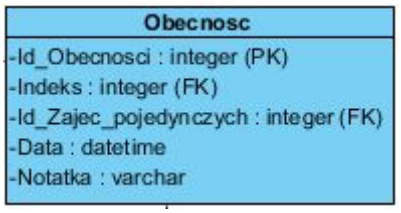
- Id_Sali : integer - jest kluczem głównym tabeli Sala. Każdy obiekt Sala posiada inną wartość w tym atrybucie.
- Budynek : varchar - nazwa budynku, w którym znajduje się sala.
- Numer : varchar - numer sali.



Rys. 7. Tabel Zajecia_pojedyncze

Powyższy rysunek przedstawia budowę tabeli Zajecia_pojedyncze. Pozwala na przechowywanie poszczególnych zajęć odbywających się w trakcie semestru. Do tworzenia obiektów tej tabeli, potrzebujemy informacji z jakiego zajęcia chcemy je wygenerować oraz czy zajęcia te odbywają się co tydzień, czy co dwa tygodnie. Na podstawie daty pierwszych zajęć, zawartej w tabeli Zajecia, oraz informacji z tej tabeli o tym, czy zajęcia są co tydzień, czy co dwa tygodnie, generowane są obiekty Zajecia_pojedyncze, z kluczem obcym Zajec, oraz z datą inkrementowaną odpowiednio o 7 dni, lub 14 dni. Obiekty generowane są automatycznie do połowy miesiąca lutego (semestr zimowy), lub do połowy miesiąca czerwca (semestr letni). Semestr z kolei jest określany przy pomocą początkowej daty z tabeli Zajęc. Ma następujące atrybuty:

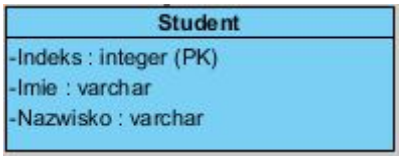
- Id_Zajec_pojedynczych : integer - jest kluczem głównym tabeli Zajecia_pojedyncze. Każdy obiekt Zajecia_pojedyncze posiada inną wartość w tym atrybucie.
- Id_Zajec : integer - klucz obcy tabeli Zajecia. Dzięki niemu może odwoływać się do obiektów tabeli Zajecia.
- Data_zajec : datetime - data oraz godzina odbywania się zajęć.



Rys. 8. Tabel Obecnosc

Powyższy rysunek przedstawia budowę tabeli Obecnosc. Pozwala na przechowywanie obecności studenta na danych zajęciach. Ma następujące atrybuty:

- Id_Obecności : integer - jest kluczem głównym tabeli Obecnosc. Każdy obiekt Obecnosc posiada inną wartość w tym atrybucie.
- Indeks : integer - klucz obcy tabeli Student. Dzięki niemu może odwoływać się do obiektów tabeli Student.
- Id_Zajec_pojdyczych : integer - klucz obcy tabeli Zajecia_pojedyncze. Dzięki niemu może odwoływać się do obiektów tabeli Zajecia_pojedyncze.
- Data : datetime - data oraz godzina zapisania obecności danego studenta.
- Notatka : varchar - notatka dotycząca obecności danego studenta.



Rys. 9. Tabel Student

Powyższy rysunek przedstawia budowę tabeli Student. Pozwala ona na przechowywanie danych o studencie, które odczytywane są z legitymacji studenckiej. Ma następujące atrybuty:

- Indeks : integer - jest kluczem głównym tabeli Student i przedstawia indeks studenta odczytany z legitymacji studenckiej. Każdy obiekt Student posiada inną wartość w tym atrybucie.
- Imie : varchar - imię studenta.
- Nazwisko : varchar - nazwisko studenta.

5.2 Moduł logowania oraz rejestracji

Moduł logowania oraz rejestracji użytkowników składa się z trzech klas: LoginForm, Registration oraz SHA2.

Po odpaleniu programu wyświetlane jest okno logowania, za które odpowiedzialna jest klasa LoginForm. Klasa ta odpowiada za proces logowania użytkowników. Składa się ona z dwóch prywatnych metod typu void. Pierwsza z nich, to jest Zalogujbutton_Click, odpowiedzialna jest za wczytanie wprowadzonych przez użytkownika danych, a następnie

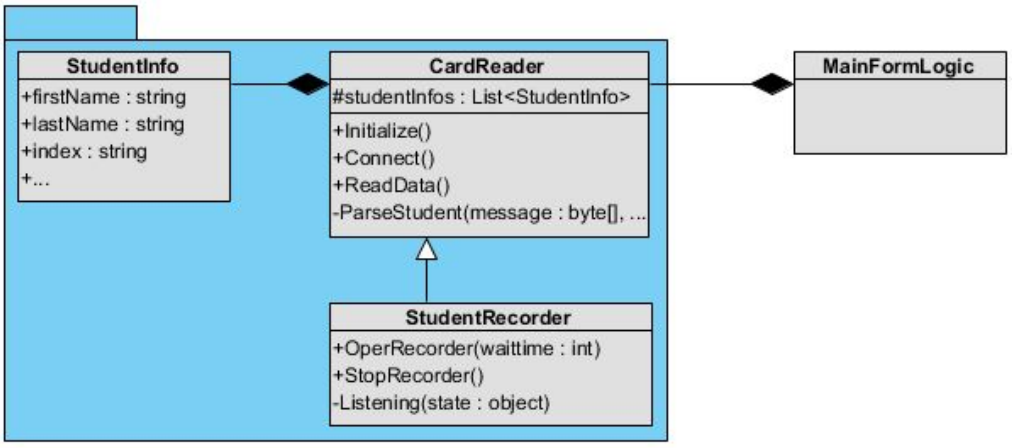
sprawdzenie ich poprawnością w bazie danych. Jeśli wszystko przebiegnie poprawnie okno logowania zostaje zamknięte, a odpalana jest właściwa część programu, wraz z danymi dotyczącymi zalogowanego użytkownika. Druga z metod w klasie LoginForm, to jest Rejestracjabutton_Click, chowa dotychczasowe okno oraz odpala okno rejestracji.

Klasa Registration odpalana jest podczas wywołania okna rejestracji. Jest ona , jak sama nazwa wskazuje, odpowiedzialna za proces rejestracji użytkowników. Podobnie jak klasa LoginForm składa się również z dwóch metod prywatnych typu void. Pierwsza z nich, to jest Zarejestrujbutton_Click, jest odpowiedzialna za sczytanie wprowadzanych przez użytkownika danych, sprawdzenia czy wpisany login nie istnieje jeszcze w bazie danych oraz porównania wpisanych haseł. Jeśli wpisane wartości pozwalają na utworzenie nowego użytkownika, takowy jest tworzony, a do bazy danych dodawane są odpowiednie rekordy.
Druga z metod tej klasy, to jest, Registration_FormClosing zamyka okno rejestracji i pokazuje, wcześniej ukryte okno logowania.

Klasa SHA2.cs jest wykorzystywana przez obie powyższe klasy do szyfrowania haseł wpisywanych podczas procesu logowania oraz rejestracji. W obecnych czasach, gdy na świecie rośnie przestępczość internetowa, a hakerzy z całego globu prześcigają się w tworzeniu wirusów oraz wymyślaniu najróżniejszych sposobów na wykradania poufnych danych, bardzo ważne jest by tworzone programy posiadały jak najlepsze mechanizmy bezpieczeństwa. Stworzony przez powyższą grupę system również musi takie posiadać. Szczególnie że przeznaczony jest do użytku na uczelniach publicznych, a jak wiadomo pomysłowość i spryt studentów nie zna granic. W związku z czym z czasem znajdzie się grupa która postanowi podszyć się pod wykładowcę i pozamieniać poszczególne obecności.
W systemie tym postawiono duży nacisk na bezpieczeństwo przesyłanych haseł. Twórcy postanowili skorzystać z funkcji skrótów SHA-2, która obecnie jest jednym z najbezpieczniejszych i najczęściej wykorzystywanych mechanizmów szyfrowania. Występuje ona w następujących wariantach: SHA-224, SHA-256, SHA-384 i SHA-512. SHA-256 działa na słowach 32bitowych, SHA-512 na słowach 64bitowych, natomiast dwa pozostałe są po prostu obciętymi wersjami dwóch pierwszych. W tym programie wykorzystano wersję SHA-256. Spełnia ona trzy podstawowe kryteria bezpieczeństwa funkcji skrótów, to jest odporność na kolizje, odporność na kolizje konkretnych wiadomości pierwszego i drugiego rzędu oraz jednokierunkowość . Dodatkowo nie ma udokumentowanego przypadku złamania owej funkcji. Hasła wpisywane przez użytkowników podczas procesu logowania są szyfrowane opisaną wcześniej funkcją skrótów, a dopiero potem porównywane są z zawartością bazy danych. Hasła w bazie nie są przechowywane w sposób jawny, a właśnie w formie skrótu.

5.3 Moduł czytnika kart

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



Rys. 10. 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 pojedynczych 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.

5.4 Moduł tworzenia harmonogramów

Moduł tworzenia harmonogramów zawiera się w klasie *MainForm*, gdzie zdefiniowane są wszystkie metody odpowiedzialne za funkcjonalność interfejsu. W celu zintegrowania aplikacji z bazą danych, moduł odwołuje się do klasy DatabaseController, gdzie zdefiniowane są metody zawierające zapytania do bazy danych.
W skład implementacji modułu wchodzą metody: *btn_dodaj_zajecia_Click*, która wywoływana jest w momencie wciśnięcia przez użytkownika przycisku dodawania zajęć i odpowiedzialna jest za pobranie odpowiednich informacji z pól ustawionych przez użytkownika oraz komunikację z bazą danych, w celu zapisania odpowiednich informacji. Podobną rolę pełnią metody *btn_dodaj_przedmiot_Click* oraz *btn_dodaj_sale_Click* z tą różnicą, że dodają do bazy danych obiekty przedmiotu oraz sali.
Kolejną metodą jest *Refresh_dgv_zajecia*, która odświeża listę zajęć zalogowanego użytkownika i wywoływana jest wewnątrz metody *btn_dodaj_zajecia_Click*.
Ostatnią metodą jest *RefreshComboBoxes*, której zadanie polega na wypełnieniu pól wyświetlających salę i przedmioty prowadzącego. Wywoływana jest na samym początku uruchomienia aplikacji, oraz po każdej interakcji z przyciskami dodawania obiektów do bazy danych. Moduł tworzenia harmonogramów jest rozszerzony o dodatkowy moduł, umożliwiający edycję pojedyńczych zajęć.

5.5 Moduł edycji zajęć

Moduł jest rozszerzeniem tworzenia harmonogramów i zawiera metody umożliwiające aktualizację bądź usuwanie konkretnych zajęć. Zdefiniowany jest w klasie *ZajeciaForm* i zawiera metodę *btn_szukaj_obecnosci_Click,* która ma za zadanie skomunikować się z bazą danych i wyświetlić listę zajęć pojedynczych na podstawie parametrów wprowadzonych przez użytkownika. Metoda ta wywołuje funkcję *FillDataTable*, która wypełnia tabelę na podstawie obiektu zwróconego przez bazę danych.
Kolejną metodą wewnątrz opisywanego modułu jest *RefreshComboboxes_PrzegladanieObecnosci* która wypełnia pola wyświetlające zajęcia.
Metody *btn_usun_Click* oraz *btn_aktualizuj_Click* wywoływane są w momencie wciśnięcia przez użytkownika przycisków “Usuń” lub “Aktualizuj”. Odpowiedzialne są za wywołanie odpowiedniej metody z klasy DatabaseController z odpowiednimi parametrami.

5.6 Moduł przeglądania obecności

Zawiera się klasie MainForm i jest odpowiedzialny za zarządzanie zakładką przeglądania obecności. Zakładka zawiera takie pola jak *cb_przedmiot_przegladanie*, *cb_data_przegladanie*, *cb_student_przegladanie* które wypełniane są przez użytkownika i są

informacją dla modułu, na podstawie jakich parametrów ma nastąpić wyszukiwanie obecności studentów na zajęciach.
Moduł zbudowany jest z metod *btn_szukaj_obecnosci_Click,* która na podstawie wyżej wymienionych pól łączy się z modułem bazy danych i wykonuje odpowiednie zapytanie. Wynikiem metody jest wypełnienie tabeli widocznej w interfejsie użytkownika zawierającej informacje o obecności studentów na zajęciach.
Metoda *RefreshComboboxes_PrzegladanieObecnosci* jest odpowiedzialna za wypełnienie pól wyświetlających listę przedmiotów oraz studentów. Wywoływana podczas uruchamiania aplikacji oraz w momencie dodania nowych obiektów do bazy danych.

5.7 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.

6. NAPOTKANE PROBLEMY

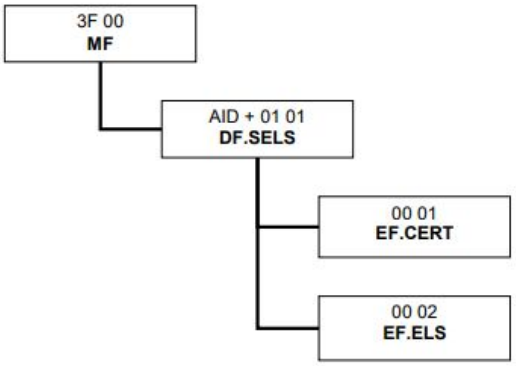
Każdy programista podczas pracy nad projektem, o większej złożoności musi w końcu trafić na różnego rodzaju problemy. Umiejętność dobrego podejścia i radzenia sobie z nimi, świadczy w dużej mierze o zdolnościach osoby kodującej. Tak też w tym rozdziale przedstawione zostały problemy na jakie natknęli 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. 11. 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.

Konrad Michalak

Podstawowym problemem było zaprojektowanie szkieletu aplikacji w sposób który byłby jednocześnie czytelny dla użytkownika i niezbyt obszerny. Aplikacja w której każda operacja skutkowałaby otwarciem nowych okienek wydaje się zbyt chaotyczna i rozpraszająca. Aby rozwiązać ten problem, należało podzielić aplikację na trzy okna: Okno logowania które zostało w pełni zaimplementowane przez Michała Gozdka, okno główne za którego funkcjonalność odpowiedzialny jest cały zespół, oraz jedno mniejsze okno umożliwiające edycję zajęć. Okno główne zostało podzielone na zakładki dzięki czemu użytkownik nie ma wrażenia przesytu informacji w jednym miejscu.

Kolejnym problemem było wyświetlanie informacji o obiektach bazy danych wewnątrz pól oferowanych przez Windows Forms zwanych ComboBoxami, tak by w przyszłości można było odwołać się do tego samego obiektu na podstawie zaznaczonego w ComboBoxie wiersza. Domyślnie są one polami tekstowymi, jednak definiując wewnątrz klasy obiektu który chcemy zamieścić odpowiednie pole typu string zwracające informacje o obiekcie można było rozwiązać ten problem.

Ostatnim problemem który okazał się czasochłonny było działanie na obiektach typu Datetime. Ze względu na wiele znaczników czasowych znajdujących się w interfejsie i ich częste formatowanie, oraz potrzeby informowania bazy danych o czasie należało dobrze zapoznać się ze strukturą typu Datetime oraz oferowanymi metodami pozwalającymi między innymi na zapis daty wielu formatach zarówno tych już zdefiniowanych jak i własnych.

Michał Gozdek

Podczas prac nad projektem pierwszymi problemami na jakie się natknął wraz z kolegą Hubertem Kaszubą był wybór typu bazy danych oraz odpowiedniego serwisu oferującego usługi chmury internetowej. Po przeanalizowaniu założeń projektowych postanowiono, że wykorzystana zostanie baza relacyjna, a dokładniej baza SQL. Wybór ten był spowodowany koniecznością powiązania poszczególnych tabel między sobą. Bez nich nie byłoby możliwości zaimplementowania zaplanowanych funkcjonalności systemu. Po dokonaniu wyboru bazy danych potrzebny był przegląd serwisów chmurowych dających możliwość utworzenia bazy SQL oraz korzystania z niej. Internet oferuje wiele takich serwisów. Wybór padł na platformę Azure. Był on dla programistów wygodny ze względu na to, że platforma ta jest własnością firmy Microsoft, która w dużej mierze tworzy usługi kompatybilne z językiem C#, w którym zaimplementowano aplikację. Dodatkowo jest ona również właścicielem środowiska programistycznego VisualStudio, które było wykorzystywane do prac implementacyjnych.

Podczas prac nad implementacją funkcjonalności aplikacji programista nie napotkał

na większe problemy. Oczywiście pojawiły się różne mniejsze problemy, między innymi, z konwersją typów danych, złożonymi zapytaniami bazodanowymi oraz odwołaniami między klasami jednak nie były to problemy, które w poważny sposób wpłynęły na czas implementacji systemu. Duży wpływ na to, że prace implementacyjne przebiegły bez większych kłopotów, miało wcześniej zdobyte przez programistę doświadczenie pracy nad podobnymi programami oraz znajomość wykorzystanych narzędzi.

Hubert Kaszuba

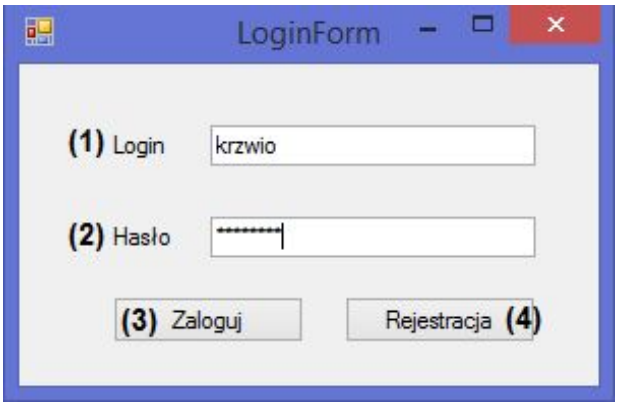
Pierwszym problemem na jaki natknął się wraz z Michałem Gozdkiem podczas prac nad projektem aplikacji, był wybór typu bazy danych. Po rozeznaniu się w problemie, wspólnie zdecydowali się, że projekt aplikacji powinien być oparty na relacyjnej bazie danych. Następnym problemem, jaki napotkali, był wybór platformy, która pozwoliłaby na utrzymywanie takiej bazy w chmurze, aby dane były dostępne dla każdego użytkownika aplikacji z każdego miejsca. Rozwiązaniem problemu okazał się wybór platformy Azure, która pozwoliła stworzyć własną bazę danych SQL, a następnie umieścić ją w chmurze. Oczywiście taka usługa nie była darmowa, ale z pomocą przyszła darmowa subskrypcja dla studentów uczelni wyższych, a z nią przyszły darmowe środki do wykorzystania na tej platformie.

Podczas dalszych prac nad aplikacją, członek zespołu napotkał tylko na jeden poważniejszy problem, a dokładniej na niewystarczającą ilość informacji przechowywanej w bazie danych, przez co trzeba było dokonać jej modyfikacji, jak również dokonać modyfikacji modelu wygenerowanego przy pomocy Entity Framework w projekcie aplikacji. Po rozwiązaniu tego problemu, programista ten nie napotkał na żadne większe problemy, które mogłyby wpłynąć negatywnie na przebieg prac. Pojawiały się pomniejsze problemy podczas implementowania funkcji do obsługi bazy danych, ale szybko były przez niego rozwiązywane.

7. INSTRUKCJA UŻYTKOWANIA

Rozdział ten zawiera instrukcję użytkowania aplikacji. Pokazane w nim zostały poszczególne ekrany wraz z dokładnym opisem co się na nich znajduje, jak z nich korzystać oraz jakie funkcjonalności oferują. Zapoznanie się z tym rozdziałem jest bardzo ważne dla każdego użytkownika aplikacji. Bez tego nie będzie dokładnie wiedział co aplikacja oferuje oraz jak z niej w poprawny sposób korzystać. Nie przeczytanie poniższej części dokumentacji może skutkować napotkaniem, przez osoby użytkujące system, na problemy lub niewykorzystaniem wszystkich oferowanych przez aplikację funkcjonalności.

7.1 Obsługa ekranu logowania



Rys. 12. Podgląd ekranu logowania, wraz z oznaczeniami.

Powyższy rysunek przedstawia ekran logowania. Jest on wyświetlany podczas startu aplikacji. Ma on za zadanie sprawdzenia tożsamości użytkownika. Składa się z następujących elementów:

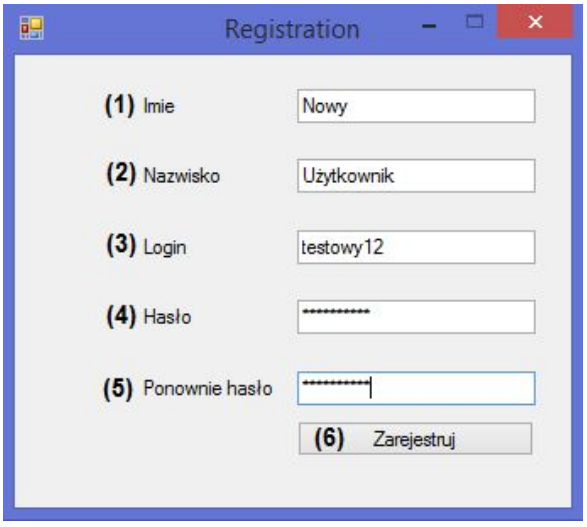
(1) Pole tekstowe loginu. Służy do wpisania przez użytkownika loginu, a co za tym idzie konta na które chce się zalogować.

(2) Pole tekstowe hasła. Służy do wpisania przez użytkownika hasła, które jest przypisane do wyżej podanego loginu.

(3) Przycisk zaloguj. Jego naciśnięcie skutkuje wysłaniem wpisanych danych do bazy oraz sprawdzeniem ich poprawności. Jeśli wszystko się zgadza okno logowania zostaje zamknięte a odpalana jest część właściwa aplikacji dla zalogowanego użytkownika.

(4) Przycisk rejestracji. Jego naciśnięcie skutkuje ukryciem ekranu logowania oraz odpaleniem okna rejestracji.

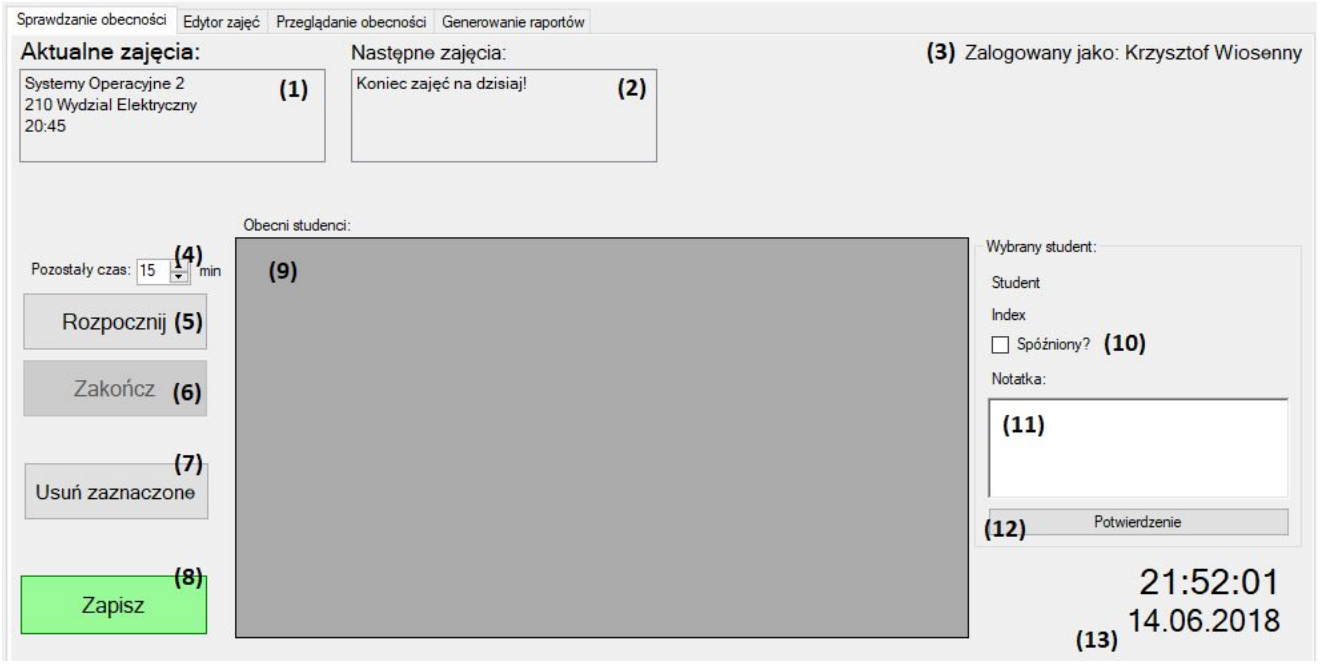
7.2 Obsługa ekranu rejestracji



Rys. 13. Podgląd ekranu logowania, wraz z oznaczeniami

Zamieszczony powyżej rysunek przedstawia ekran rejestracji. Jest on odpalany poprzez naciśnięcie odpowiedniego przycisku na ekranie logowania. Pozwala on na dodanie do serwisu nowych kont użytkowników. Składa się z następujących elementów:
(1) Pole tekstowe imienia. Służy do wpisania przez użytkownika imienia, przypisanego do nowo tworzonego konta.
(2) Pole tekstowe nazwiska. Służy do wpisania przez użytkownika nazwiska, przypisanego do nowo tworzonego konta.
(3) Pole tekstowe loginu. Służy do wpisania przez użytkownika loginu, przypisanego do nowo tworzonego konta. Login jest konieczny do procesu logowania.
(4) Pole tekstowe hasła. Służy do wpisania przez użytkownika hasła, przypisanego do nowo tworzonego konta. Hasło jest konieczne do procesu logowania.
(5) Pole tekstowe powtórnego hasła. Służy do powtórnego wpisania przez użytkownika hasła przypisanego do nowo tworzonego konta. Jest ono porównywane z wpisanym powyżej hasłem. By rejestracja mogła się powieść muszą być one identyczne. Jeśli tak nie jest rejestracja jest niemożliwa. Ma ono za zadania zapobiec błędom wpisywania hasła przez użytkownika. Brak mechanizmu powtórnego wpisywania hasła oraz porównania haseł mógłby doprowadzić do wpisania niechcianego, przypadkowego znaku.
(6) Przycisk zarejestruj. Jego naciśnięcie skutkuje sczytaniem wpisanych danych porównaniem haseł. Jeśli dane są poprawne, sprawdzana jest dostępność wpisanego loginu. W przypadku gdy jest on dostępny tworzone jest nowe konto, ekran rejestracji jest zamykany a ukazywane jest ponownie ekran logowania.

7.3 Obsługa ekranu sprawdzania obecności



Rys. 14. Podgląd ekranu sprawdzania obecności, wraz z oznaczeniami

Na powyższym rysunku przedstawiony jest ekran widoczny zaraz po pomyślnym zalogowaniu się do aplikacji. Ekran jest najważniejszym elementem aplikacji, jakim jest sprawdzanie obecności studentów, z tego też powodu posiada wiele mechanizmów które wymagają dokładniejszego omówienia:

(1) Pole aktualnych zajęć - Aktualne zajęcia są polem, w którym przedstawione są informacje dotyczące obecnie odbywających się zajęć. Są to informacje takie jak nazwa przedmiotu, sala, budynek a także godzina o której zajęcia się rozpoczęły. W przypadku gdy nie odbywają się żadne zajęcia, pole to zostanie wypełnione odpowiednim komunikatem mówiącym, że obecnie nie odbywają się żadne zajęcia.

(2) Pole następnych zajęć - Zastosowanie tego pola jest bardzo podobne jak w przypadku pola z punktu pierwszego, informuje ono użytkownika aplikacji, o następnych zajęciach jakie odbędą się po aktualnych zajęciach, bądź o następnym zajęciach w ogóle w dniu dzisiejszym. Jeżeli w obecnym dniu użytkownik nie ma wpisanych w harmonogramie żadnych zajęć po godzinie późniejszej niż aktualna na zegarze, pole będzie zawierać komunikat z informacją, że nie odbędą się już żadne zajęcia.

(3) Pole użytkownika - Przeznaczeniem tego pola, jest informowanie użytkownika korzystającego z aplikacji o tym, na jakie konto się zalogował. Pole jest zbudowane z imienia i nazwiska zalogowanego użytkownika.

(4) Pole stopera - Zanim prowadzący zajęcia zacznie sprawdzać obecność, powinien ustalić czas w ciągu którego studenci mogą się zapisywać na listę. Domyślną wartością tego pola jest 15 (słownie: piętnaście) minut. Wartość pola można wpisać przy użyciu klawiatury bądź myszki wykorzystując strzałki widoczne po prawej stronie od cyfry reprezentującej czas przeznaczony na sprawdzanie obecności.

(5) Przycisk uruchamia procedurę sprawdzania obecności. Od tego momentu czytnik zaczyna nasłuchiwać czy wczytana została karta oraz uruchamia się stoper opisany w punkcie (4). Po wciśnięciu tego przycisku, zostanie on zablokowany a odblokowany zostanie przycisk opisany w punkcie (6).

(6) Przycisk może być stosowany w momencie wcześniejszego zakończenia procesu sprawdzania obecności. Po jego wciśnięciu przycisk ten zostanie zablokowany, natomiast przycisk z punktu (5) zostanie odblokowany. Lista zapisanych studentów pozostanie bez zmian.

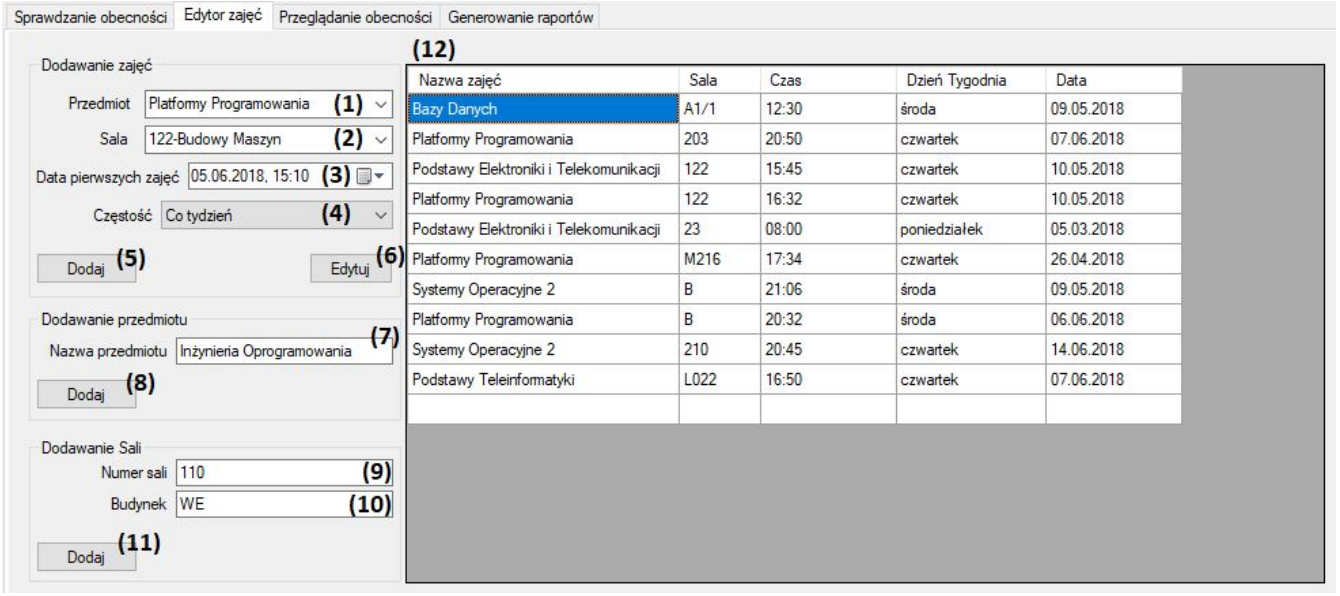
(7) Zadaniem tego przycisku jest usuwanie wierszy z listy studentów (9). Należy zaznaczyć lewym przyciskiem myszy wiersz który chcemy usunąć, a następnie zastosować ten przycisk.

(8) Przycisk Zapisz, umożliwia zapisanie w bazie danych listy studentów którzy znajdują się na liście (9). Przycisk zadziała dopiero, jeżeli zakończony zostanie proces sprawdzania obecności. Możliwe jest wielokrotne zapisywanie listy studentów dla danych zajęć, jednak zalecane jest wykonywanie tego procesu dopiero, gdy lista będzie w pełni gotowa do wysłania.

(9) Lista studentów - lista wyświetlająca studentów zapisujących się jako obecni. Dane pobierane i zapisywane w liście są automatycznie, po włożeniu karty do czytnika. Lista zawiera następujące pola: Imię, Nazwisko, Indeks, Spóźniony, Notatka, Znacznik.

Pole “Spóźniony” jest informacją o tym, czy student spóźnił się na zajęcia. Domyślnie pole to jest niezaznaczone co oznacza, że przyszedł na czas. Notatka pozwala na wprowadzenie dodatkowych informacji o studencie a znacznik, jest dokładną informacją o czasie zarejestrowania studenta w systemie.
(10) Aby skorzystać z tego pola, należy zaznaczyć wiersz ze studentem z listy (9). Pole jest informacją dla prowadzącego, czy student przyszedł spóźniony czy na czas. Domyślnie pole to jest niezaznaczone co oznacza, że przyszedł na czas.
(11) Aby skorzystać z tego pola, należy zaznaczyć wiersz ze studentem z listy (9). Notatka pozwala na wprowadzenie dodatkowych informacji o studencie.
(12) Aby skorzystać z tego przycisku, należy zaznaczyć wiersz ze studentem z listy (9) oraz zmienić wartości pól (10) lub (11). Przycisk Potwierdzenie jest informacją, że użytkownik chce zmienić dane o zaznaczonym studencie. Dane zostaną zapisane na liście (9) a nie od razu w bazie danych.
(13) Informacja o godzinie oraz dacie. Pobierana jest z zegara systemowego.

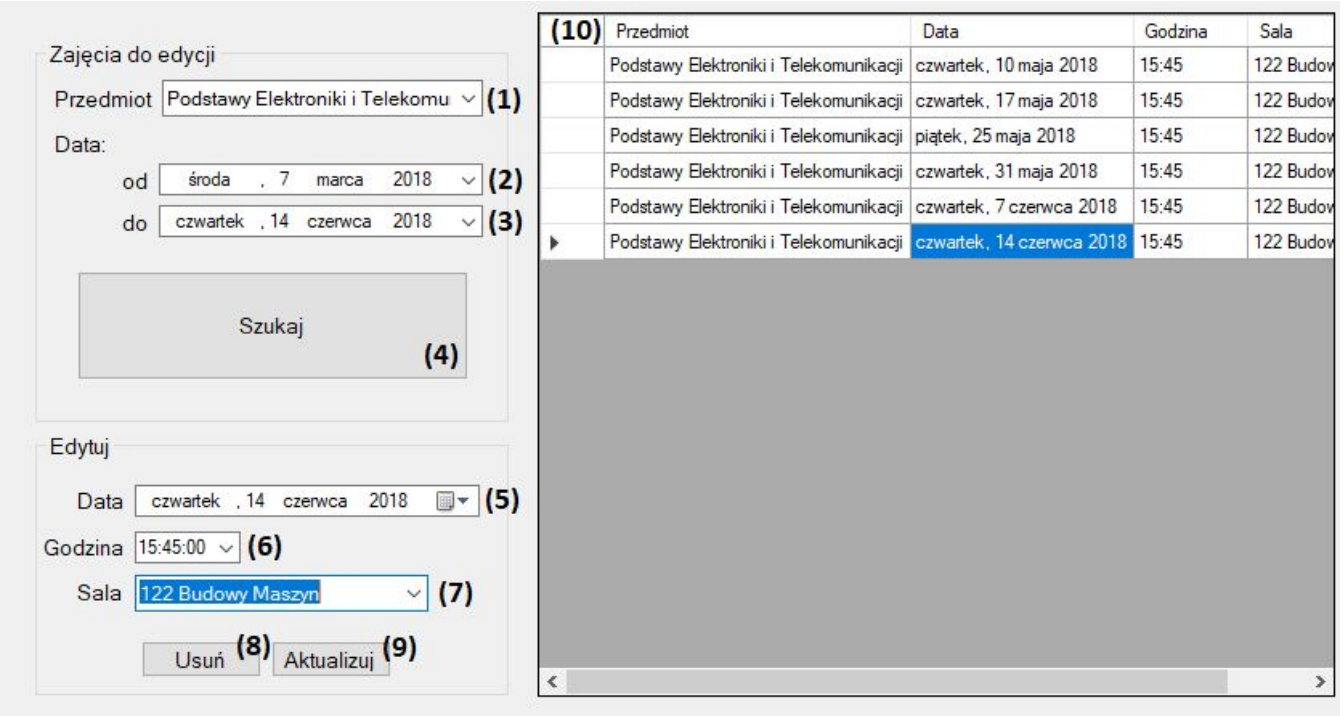
7.4 Obsługa ekranu edycji zajęć



Rys. 15. Podgląd ekranu tworzenia harmonogramów, wraz z oznaczeniami

(1) Pole wyboru przedmiotu - pole jest wysuwaną listą zawierającą przedmioty prowadzone przez użytkownika. Lista zawiera również wbudowaną funkcję autosugestii, wystarczy wpisać znaki zawierające się w danej frazie, a system podpowie resztę nazwy przedmiotu.
(2) Pole wyboru sali - pole jest wysuwaną listą zawierającą informację o sali oraz budynku w którym mają odbyć się dodawane zajęcia. Lista zawiera również wbudowaną funkcję autosugestii, wystarczy wpisać znaki zawierające się w danej frazie, a system podpowie resztę nazwy sali i budynku.
(3) Pole wyboru daty - pole jest informacją o dacie pierwszych zajęć jakie się mają odbyć oraz o godzinie ich rozpoczęcia. Dane można wpisać ręcznie, bądź rozwinąć kalendarz przy użyciu strzałki znajdującej się po prawej stronie pola.

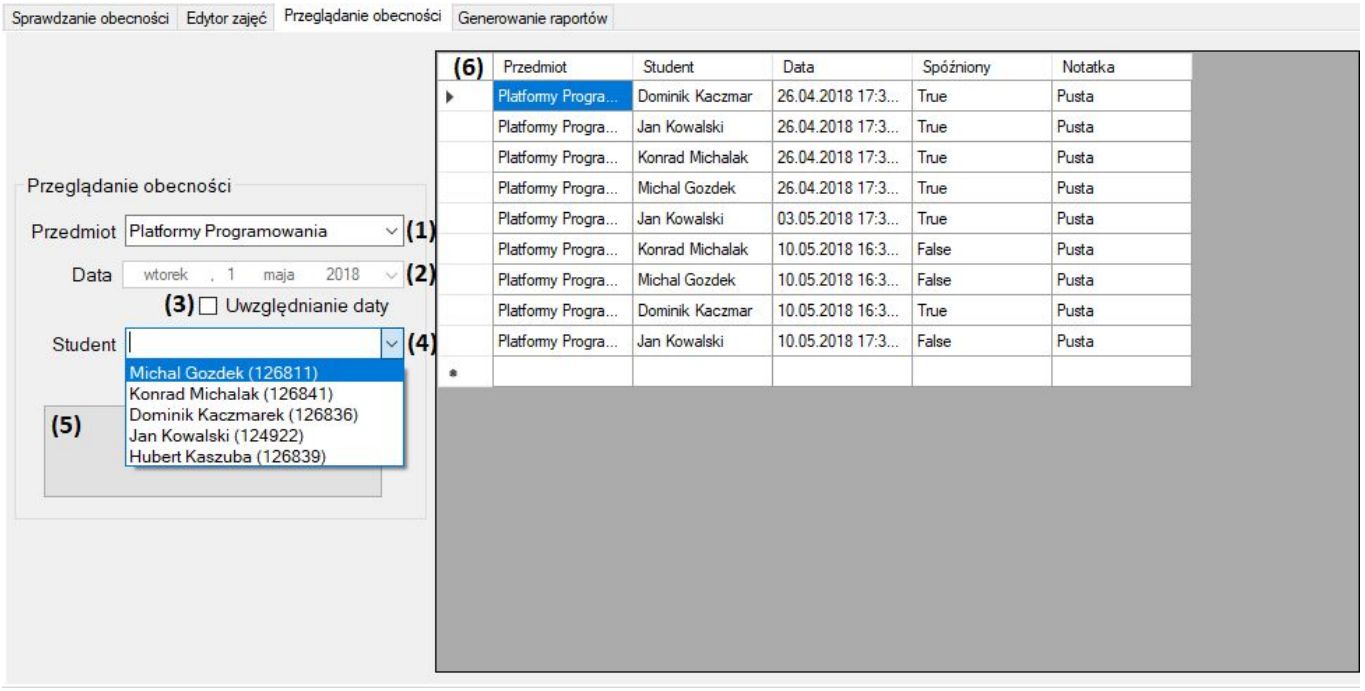
(4) Pole wyboru częstości - częstość jest informacją o częstotliwości występowania zajęć: co tydzień/co dwa tygodnie. Na tej podstawie algorytm wygeneruje zajęcia pojedyncze w przód i zapisze je w bazie danych.
(5) Przycisk dodaj, służy do zapisania informacji z pól wymienionych w poprzednich punktach w bazie danych w postaci jednego obiektu zajęć. Dodanie zajęć spowoduje odświeżenie się listy (12).
(6) Przycisk edycji jest wykorzystywany w przypadku chęci zmiany danych o zajęciach pojedynczych. Spowoduje to otworzenie nowego okna w którym użytkownik będzie mógł podać szczegółowe informację o zajęciach których ma dotyczyć edycja.
(7) Pole nazwy przedmiotu - nazwa przedmiotu jest polem tekstowym wewnątrz którego użytkownik może podać nazwę przedmiotu który chce zapisać w bazie danych.
(8) Przycisk dodaj umożliwia dodanie nowego przedmiotu o nazwie wprowadzonej w polu nazwy przedmiotu (7) do bazy danych.
(9) Pole numeru sali - numer sali jest polem tekstowym wewnątrz którego użytkownik może podać numer sali który chce zapisać w bazie danych.
(10) Pole nazwy budynku - budynek jest polem tekstowym wewnątrz którego użytkownik może podać nazwę budynku która zostanie zapisana w bazie danych.
(11) Przycisk dodaj umożliwia dodanie nowej sali o nazwie wprowadzonej w polu numeru sali (9) znajdującej się w budynku wprowadzonym w polu (10) do bazy danych.
(12) Lista zajęć - lista zajęć prowadzonych przez zalogowanego użytkownika. Zawiera informacje takie jak nazwa zajęć, sala, godzina rozpoczęcia, dzień tygodnia oraz data.



Rys. 16. Podgląd ekranu edytora zajęć, wraz z oznaczeniami

(1) Pole jest wysuwaną listą zawierającą przedmioty prowadzone przed zalogowanego użytkownika. Lista zawiera również wbudowaną funkcję autosugestii, wystarczy wpisać znaki zawierające się w danej frazie, a system podpowie resztę nazwy przedmiotu.
(2) Data od, jest polem informującym od kiedy algorytm ma zacząć wyszukiwanie zajęć podanych w polu (1). Dane można wpisać ręcznie, bądź rozwinąć kalendarz przy użyciu strzałki znajdującej się po prawej stronie pola.
(3) Data do, jest polem informującym do kiedy algorytm ma wyszukiwać zajęcia podanych w polu (1). Dane można wpisać ręcznie, bądź rozwinąć kalendarz przy użyciu strzałki znajdującej się po prawej stronie pola.
(4) Przycisk szukaj rozpoczyna wyszukiwanie zajęć pojedynczych na podstawie parametrów podanych przez użytkownika w polach (1), (2), (3). Wyniki wyszukiwania wyświetlone zostaną na liście (10).
(5) Pole edytuj datę, jest polem którego zawartość zostanie przekazana jako nowa wartość zajęć pojedynczych które mają zostać zaktualizowane. Dane można wpisać ręcznie, bądź rozwinąć kalendarz przy użyciu strzałki znajdującej się po prawej stronie pola.
(6) Pole pozwala na zmianę godziny zajęć pojedynczych. Dane można wpisać ręcznie, bądź rozwinąć kalendarz przy użyciu strzałki znajdującej się po prawej stronie pola.
(7) Pole sala jest wysuwaną listą zawierającą informację o sali oraz budynku w którym mają odbyć się edytowane zajęcia pojedyncze. Lista zawiera również wbudowaną funkcję autosugestii, wystarczy wpisać znaki zawierające się w danej frazie, a system podpowie resztę nazwy sali i budynku.
(8) Przycisk usuń ma za zadanie usunąć zajęcia pojedyncze z bazy danych. Należy pamiętać, o zaznaczeniu wiersza zajęć z listy (10).
(9) Przycisk aktualizuj ma za zadanie zaktualizować zajęcia pojedyncze w bazie danych. Należy pamiętać o zaznaczeniu wiersza zajęć z listy oraz o poprawnym wypełnieniu pól (5), (6) oraz (7).
(10) Lista zajęć pojedynczych prowadzonych przez zalogowanego użytkownika. Zawiera informacje takie jak nazwa przedmiotu, sala, godzina rozpoczęcia oraz data. Lista jest wypełniana po użyciu przycisku szukaj (4) i wyświetla informacje na podstawie parametrów podanych w polach (1), (2) oraz (3).

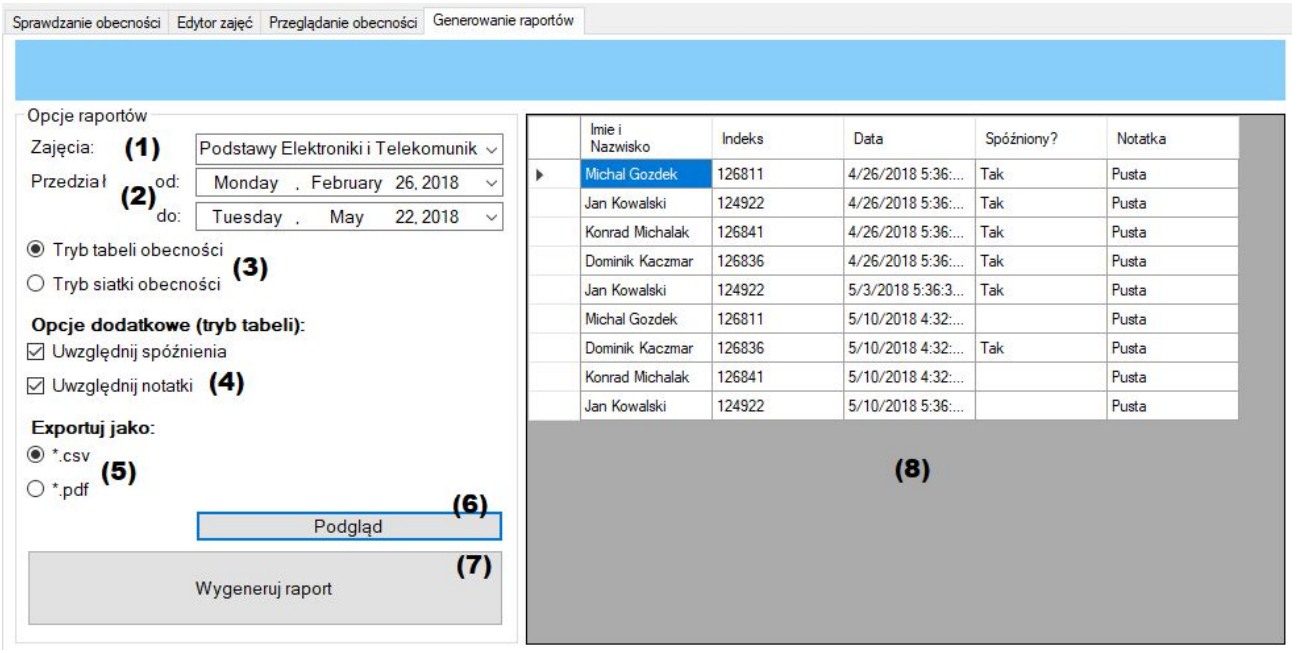
7.5 Obsługa zakładki przeglądania obecności



Rys. 17. Podgląd ekranu przeglądania obecności, wraz z oznaczeniami

(1) Pole jest wysuwaną listą zawierającą przedmioty prowadzone przed zalogowanego użytkownika. Lista zawiera również wbudowaną funkcję autosugestii, wystarczy wpisać znaki zawierające się w danej frazie, a system podpowie resztę nazwy przedmiotu.
(2) Pole jest informacją o dacie szukanych zajęć jakich szukamy. Dane można wpisać ręcznie, bądź rozwinąć kalendarz przy użyciu strzałki znajdującej się po prawej stronie pola.
(3) Znacznik uwzględnienie daty, służy jako informacja czy użytkownik szuka zajęć po dacie czy też nie. Odhaczenie tego pola oznacza odblokowanie dostępu do pola daty (2).
(4) Pole jest wysuwaną listą zawierającą studentów jacy pojawili się na zajęciach obecnie zalogowanego użytkownika. Lista zawiera również wbudowaną funkcję autosugestii, wystarczy wpisać znaki zawierające się w danej frazie, a system podpowie resztę nazwy studenta.
(5) Przycisk szukaj rozpoczyna wyszukiwanie studentów którzy przyszli na zajęcia na podstawie parametrów wybranych przez użytkownika. Aby przycisk wykonał swoją funkcję, należy wypełnić przynajmniej jedno pole z parametrami.
(6) Lista obecnych studentów. Zawiera informacje takie jak nazwa przedmiotu, imię i nazwisko studenta, data sprawdzenia obecności danego studenta, znacznik czy student się spóźnił na dane zajęcia czy nie oraz ewentualna notatka. Lista jest wypełniana po użyciu przycisku szukaj (5) i wyświetla informacje na podstawie parametrów podanych w polach (1), (2), (3) oraz (4).

7.6 Obsługa zakładki generowania raportów



Rys. 18. 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 zajęć prowadzącego - 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) Przycisk podglądu - 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.

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
- Dokumentacja struktury DateTime https://msdn.microsoft.com/pl-pl/library/system.datetime(v=vs.110).aspx

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/