System kontroli obecności na podstawie elektronicznych legitymacji studenckich

(Students attendence verifing system with the use of student card)

Dawid Szczyrk

Praca magisterska

Promotor: dr Jakub Michaliszyn

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

31 sierpnia 2020

Streszczenie

. . .

. . .

Spis treści

Rozdział 1.

Wprowadzenie

Celem niniejszej pracy jest zaprojektowanie użytecznego i niedrogiego w utrzymaniu systemu do kontroli obecności studentów na wykładzie.

Niemalże każde zajęcia na naszym instytucie zaczynają się od zebrania listy obecności. Odbywa się to w siermiężny sposób, ponieważ każdy ze studentów musi wpisać swoje dane na listę obecności. Taka forma jest uciążliwa, zajmuje kilka pierwszych minut zajęć, a studenci się rozpraszają podając sobie listę obecności. Dziś, kiedy do ochrony danych osobowych przykłada się coraz większą wagę, starodawne sposoby sprawdzania obecności są tym bardziej wątpliwe.

W dobie współczesnych zdobyczy technologicznych można uznać za zaskakujące, że wziąć trafamy przy tym czasochłonnym procesie. Wydawać by się mogło, że obecne podejście jest tak zakorzenione w akademickiej kulturze, że nie ma możliwości tego zmienić. W niniejszej pracy chciałbym podjąć tę rękawicę i spróbować zaproponować rozwiązania które mogłyby cały proces znacznie usprawnić.

Poniższa praca przedstawia proces od postawienia wymagań do realizacji finalnej wersji wielokomponentowego projektu. Na efekt ostateczny składają się aplikacje wykorzystujące najnowsze technologie i oparte na popularnych frameworkach, oraz niskopoziomowe elektroniczne urządzenie zaprojektowane i wykonane na potrzyby tego projektu.

Z jednej strony ramach pracy musiałem zaprojektować komunikacje pomiędzy poszczególnymi komponentami, zaproponować i wdrożyć architekturę aplikacji internetowych, tzn. serwery na któych wykonywany jest kod stron internetowych oraz bazy danych. W wersji finalnej użytkownikowi umożliwiono stworzenie własnego konta w internetowym serwisie i zarządzanie danymi o obecnościach w jego kontekście. Wszystko to zostało oparte na nowoczesnych rozwiązaniach chmurowych.

Z drugiej strony moim zadaniem było zaprojektowanie niedrogiego urządzenia elektronicznego, które miało za zadanie udostępniać zrozumiały interfejs i realizować zadania zwiazane ze zbieraniem danych od uczesników wykładu. Zagadnienie to wymagało wiedzy z zakresu elekroniki i zapoznania się ze schamatami połączeń pomiędzy poszczególnymi modułami i zrozumienia zasady ich działania. Dodatkowo

ważnym elementem było połączenie odrębnych komponentów w jedną całość, a więc projekt układu elektronicznego na płytce stykowej i zamknięcie wszystkiego w opakowaniu umożliwiającym bezpieczne i wygodne użytkowanie.

Rozdział 2.

Określenie wymagań

2.1. Analiza obecnego sposobu sprawdzania obecności

W celu doprecyzowania wymagań stawianych przed projektowanym przeze mnie system postanowiłem rozważyć znany mi z autopsji proces weryfikowania obecności uczestników zajęć na zajęciach prowadzonych przez Uniwersytet Wrocławski

Sprawdzanie listy obecności jest w dużej mierze uzależnione od preferencji prowadzącego i zwyczajów panujących na konkretnym wydziale - przykładowo obecność może być potwierdzona słownie - poprzez kolejne wywołanie osób z listy, albo pisemnie - poprzez udostępnienie listy na którą uczestnicy będą zobowiązani się wpisać.

Dla uproszczenia dalszych rozważań postanowiłem przyjąć za ogólnie stosowaną metodę pisemną - która według moich obserwacji jest stosowana częściej - i na tej podstawie wyodrębnić czynności które składają się na wypełnienie listy obecności podczas zajęć.

Zbieranie informacji o obecności studentów na wykładzie jest czynnością na pierwszy rzut oka oczywistą, po bliższym przyjrzeniu się całej procedurze, okazuje się jednak, że daje się ona rozłożyć na jeszcze prostsze elementy.

- Przygotowanie kartki z listą obecności (przypisanie listy obecności do zajęć których będzie ona dotyczyła).
- Udostępnienie kartki uczestnikom oraz wpisywanie się na listę.
- Weryfikacja poprawności zebranej listy poprzez przeliczenie osób obecnych na zajęciach.
- Przeniesienie zebranych danych do uczelnianego systemu.

Powyższy schemat zbierania listy obecności podatny jest na dodatkowe błędy. Nie trudno wyobrazić sobie sytuację w której student intencjonalnie podaje nie swoje

dane w celu zaliczenia obecności innej osobie. Kolejnym krokiem który może doprowadzić do błędów na ostatecznej liście obecności jest błąd ludzki podczas przenoszenia danych z kartki do uczelnianego systemu komputerowego. Podobny błąd trudno jest od razu wychwycić ze względu na to, że studenci zazwyczaj nie przeglądają na bieżąco list obecności dostępnych np. na usosie. Kiedy na koniec semestru obecność jest rozliczana bardzo trudno dojść do faktów.

Pierwszym elementem nadającym się w moim mniemaniu do usprawnienia jest samo tworzenie listy obecności z informację na temat zajęć których taka lista będzie dotyczyła. Fizyczna lista, poza czasem poświęconym na jej stworzenie musi być po odbytych zajęciach przechowywana - co stwarza ryzyko jej zgubienia. Dodatkowo istnieje prawdopodobieństwo, że trafi w niepowołane ręce i ktoś nieżyczliwy wpisze na nią dodatkowe nazwiska. Do tego wszystkiego dochodzi jeszcze aspekt ekologiczny, ponieważ każda kolejna kartka to dodatkowy element wyrwany matce naturze.

Również udział studentów w rejestrowaniu własnej obecności może ulec optymalizacji. Wpisanie własnego imienia i nazwiska, a czasem numeru albumu zabiera czas, naraża na upublicznienie dane osobowe i utrudnia studentowi śledzenie wykładu.

Weryfikacja poprawności listy jest nie tylko pracochłonna, ale również podatna na błędy. Czas poświęcony na policzenie uczestników rośnie wprost proporcjonalnie do ich liczby. Dodatkowo prowadzący zajęcia może się zwyczajnie pomylić albo ktoś dopisać do listy już po tym sprawdzeniu.

Z całą pewnością usprawnienia wymaga proces przenoszenia danych z fizycznego nośnika do uczelnianego systemu komputerowego. Problemem jest nie tylko to, że podczas wpisywania obecności do komputerowego systemu można poopełnić błąd, ale również to, że jest to proces czasochłonny.

2.2. Zdefiniowanie wymagań

Na podstawie poszczególnych kroków w procesie zbierania obecności, wyodrębniłem cztery właściwości przez pryzmat których mogłem ocenić potencjalne rozwiązania. Dodatkowo w dalszych rozważaniach do grona kryteriów oceny poszczególnych możliwości dołącze jeszcze dwa istotne elementy - przewidywany koszt, oraz stopień skomplikowania rozwiązania.

2.2.1. Identyfikacja zajęć na których obecność była weryfikowana

Jestem w stanie zidentyfikować dwa możliwe podejścia do tego problemu. Rozwiązanie tradycyjne realizuje to zadanie w naturalnej kolejności tzn. w momencie zbierania obecności wiemy dla jakich zajęć jest ona zbierana. W moim przekonaniu warto rozważyć opcję odwrotną, tzn. najpierw zbierać dane (pamiętając jedynie kiedy zostały zebrane), a w dalszej kolejności decydować jakich zajęć one dotyczą.

W ten sposób unikamy konieczności wprowadzenia do nośnika informacji na temat tego jakich zajęć dotyczą zebrane dane.

2.2.2. Zbieranie danych od studentów

W tej kwestii brałem pod uwagę trzy możliwości. Tradycyjne rozwiązanie realizowane poprzez wpisywanie się na listę może zostać zastąpione poprzez użycie bardziej nowoczesnego sposobu identyfikacji - np. wbudowaną funkcję legitymacji studenckiej tj. identyfikację zbliżeniową. Innym sposobem byłoby zastąpienie fizycznej kartki wypełnianej przez osoby uczesniczące w zajęciach cyfrowym formularzem udostępnianym przez internetową aplikację. Ostatnią rozważaną przeze mnie możliwością jest skorzystanie z istniejącego albo stworzenie własnego systemu do rozpoznawania twarzy.

2.2.3. Wprowadzenie danych do systemu

To zadanie może być realizowane na dwa sposoby. Pierwszym z nich jest rejestrowanie obecności w systemie od razu przy jej odnotowaniu. W tym wypadku wymagane jest bezpośrednie połączenie z systemem. Drugim - zebranie wszystkich danych na wstępie i późniejsze, zbiorcze wprowadzanie ich do aplikacji. W tym wypadku niezbędny będzie nośnik na którym dane zostaną przetransportowane, wskazane jest również zaszyfrowanie tych danych na czas transportu.

2.2.4. Weryfikacja poprawności danych

W przypadku tradycyjnego zbierania obecności weryfikacja jest niedoskonała, ale konieczna ze względu na to że nośnik danych - kartka z listą obecności - udostępniana jest szerokiej publiczności. Podobny problem będzie występował jeśli kartkę papieru zastąpimy cyfrowym tabletem. Możemy również odseparować medium i każdą zebraną obecność weryfikować osobno.

W kontekście powyższych rozważań zdecydowałem się na następujące założenia

- Implementowany przeze mnie system powinien być w stanie zbierać obecności bez wcześniejszego określenia w kontekście jakich zajęć jest to robione.
- Dane od studentów powinny być zbierane bez ingerencji wykładowcy i nie zaburzając porządku zajęć.
- Wprowadzanie danych powinno odbywać się w taki sposób, żeby uniknąć pomyłki.
- Student nie powinien mieć możliwości ingerencji w listę obecności.

2.3. Inne brane pod uwagę realizacje realizacje

2.3.1. Responsywna aplikacja internetowa umożliwiająca wygodne rejestrowanie obecności studentów na zajęciach w samej aplikacji przez internet albo telefon

.

W mojej ocenie to najprostsze rozwiązanie składające się z jednolitego systemu, w którym jednocześnie można zbierać obecności uczestników wykładu, oraz je przechowywać i przeglądać. Jej niewątpliwym plusem jest całkowite wyeliminowanie problemu związanego z wprowadzaniem danych do systemu katalogującego obecności, ponieważ dane wprowadzane są wprost do systemu. Identyfikacja konkretnych zajęć w ramach których rejestrowana jest obecność, odbywałaby się na początku zajęć poprzez wejście w kontekst opowiedniego wykładu w aplikacji. Obecność uczestników na zajęciach byłaby sprawdzana przez prowadzącego i od razu wprowadzana do systemu.

Nie zdecydowałem się na to rozwiązanie, ponieważ wydaje mi się ono nieszczególnie odbiegać od aktualnej procedury, poza zminą nośnika z kartki papieru na ekran telefonu.

2.3.2. Urządzenie rejestrujące obecności za pośrednictwem sieci bezprzewodowej

To rozwiązanie początkowo wydało mi się bardzo atrakcyjne i przez długi czas miałem zamiar je zrealizować. Planowałem rozszerzyć mikrokontroler o moduł do bezprzewodowej łączności z internetem i w stworzonej aplikacji udostępnić interfejs do odnotowywania obecności.

Po dłuższych rozważaniach okazało się jednak, że takie rozwiązanie przerasta moje możliwości. Ten sposób wysyłania danych do aplikacji katalogującej obecności wymagałby umożliwienia powiązania obecności z zajęciami na których zostały zebrane

Wymagałoby to dołączenia dodatkowego interfejsu który pozwalałby wprowadzać takie dane, co jeszcze bardziej komplikowałoby projektowane przeze mnie urządzenie. Dodatkowo ogtraniczyłoby to zbiór układów scalonych z których mógłbym skorzystać, ponieważ musiałbym wybrać takie, które posiadają wystarczającą ilość pamięci, pinów i innych potrzebnych peryferiów.

Ostatecznie moje plany dotyczące takiego projektu pokrzyżował moduł ESP, który miał realizować komunikację bezprzewodową, ale okazał się bardzo kapryśny.

Rozdział 3.

Określenie struktury projektu

Finalnie zdecydowałem, żeby oddzielić sposób zbierania danych od ich przechowywania i przetwarzania.

Komponent umożliwiajacy studentowi odnotowanie swojej obecności na wykładzie - weryfikator - postanowiłem oprzeć o platformę arduino - znaną mi z zajęć systemów wbudowanych.

Rolę oprogramowania odpowiedzialnego za katalogowanie zebranych przez weryfikator danych będzie w moim systemie ogólnodostępna aplikacja internetowa, umożliwiająca założenie konta i zarządzenie zebranymi obecnościami w konekście całego przedmiotu i w kontekście pojedynczych zajęć. Dodatkowo ma ona umożliwiaać generowanie raporrtów ze zagregowanymi danymi, tak aby dało się łatwo prześledzić obecności w konktekście całego semetru.

Kolejnym elementem mojego systemu, niejako wymuszonym przez brak dostępu do rzeczywistych danych uniwersyteckich, jest aplikacja pozwalająca na powiązanie danych pobranych z legitymacji studenckiej z danymi pozwalającymi na jego identyfikację. Jest to aplikacja rozłączna względem systemu zarządzającego obecnościami osób uczesniczących w zajęciach, wystawiająca RESTowe api.

Rozdział 4.

Realizacja

Opis technologi wykorzytywanych przez ELS

4.1. Weryfikator - czytnik legitymacji studenckich

Zgodnie z wymaganiami projetkowane przeze mnie urządzenie powinno udostępniać co najmniej następujące możliwości.

- Możliwość podłączenie zewnętrznego nośnika danych.
- Stworzenie pliku na którym zapamiętani zostaną uczestnicy zajęć
- Zebranie informacji z elektronicznej legitymacji studenckiej, potwierdzające obecność uczesników
- Wyświetlanie informacji o stanie urządzenia, w szczególności sukcesy odczytania danych z legitymacji
- Zapamiętanie czasu kiedy zebrane zostały obecności

Dodatkowo, chciałbym żeby cała elektronika schowana była wewnątrz plastkiwego opakowania, a zasilanie dostarczane zapomocą kabla podłączenego do sieci elektrycznej. Opakowanie powinno zostać wyposarzone w otwory na kabel zasilający, zewnętrzny nośnik danych, wyświetlacz. Czytnik kart RFId powinien być tak zlokalizowany aby umożliwić łatwe zebranie danych z ELS.

4.1.1. Zastosowana platforma i peryferia

Arduino to cała platforma programistyczna umożliwiająca łatwe projektowanie i testowanie układów składających się z wielu elektronicznych modułów. Podstawą każdego projektu Arduino jest układ elektroniczny oparty na mikroprocesorze. Pod

egidą tego systemu udostępniony jest cały wachlarz wesrji takich układów, różniących się rodzajem kontolera i udostępnianymi peryferiami (przykładowo: slot na kartę SD, gniazdo RJ-45, wbudowany wyświetlacz, czy moduł wifi). Do wyboru użtkownika pozostaje cała gama wersji płytek drukowanych spod znaku Arduino. Dodatkowo platforma udostępniona jest na licencji open hardware, co sprawiło, że z projektem związana jest olbrzymia międzynarodowa społeczność entuzjastów elektroniki. Cżęścią platformy jest również środowisko programistyczne umożliwiające wgrywanie własnych programów na podłączoną do komputera płytkę. Ciekawą możliwościa dostępną dla użytkowników platformy jest możliwość skonstruowania własnej płytki z procesorem na bazie powszechnie dostępnych schamatów gotowych układów. Może to być rozwiązanie bardzo atrakcyjne w przypadku kiedy projektantowi zależy na ograniczeniu kosztów układu. W moim rozwiązaniu postanowiłem nie iść tak daleko i skorzystać z jednej z gotowych płytek. Podstawą mojego weryfikatora postanowiłem uczynić płytkę Arduino Uno opartą na mikroprocesorze AVR ATmega328. Kontroler w tym układzie jest bardzo prosty, posiada bardzo mocno ograniczone zasoby, co wymusza dodatkowy wysiłek włożony w zarządzanie pamięcią i czasem procesora. Do moejej dyspozycji miałem następujące zasoby

- 32 kB Flash - przestrszeń na kod programu
- 2 kb SRAM zajmowane przez pamięć operacyjną.
- 2 porty SPI, port I2C
- Jednostka obliczeniowa o częstotliwości zegara 20MHz

Dodatkowo, w celu zapenienia wszystkich wymaganych funkcjonalności przez projektowany przeze mnie weryfikator, skorzystałem z następujących dodatkowych modułów:

- Czytnik RFID - do odczytywania danych z legitymacji studenckiej wyposażonej w interfejs zbliżeniowy. Czytnik podłączony jest przez port SPI
- USB Host Shield - dodające port USB, umożliwiający podłączenie pendriva, korzystający z drugiego portu SPI
- Wyświetlacz LCD wraz z potencjometrem do ustawiania kontrastu, przekazujący użtkownikowi informacje na temat stanu urządzenia
- RTC - zegar czasu rzeczywistego z baterią do zapamiętywania czasu zbierania danych, korzystający z interfejstu I2C
- Buzzer piezo wydający dźwiękowe sygnały w razie sukcecsu

4.1.2. Konstrukcja

Do skonstuowana ostatecznegoukładu z zebranych elementów użyłem dodatkowo jednej dwustronnej płytki stykowej.

- Jednej dwustronnej płytki stykowej
- Dwóch rezystorów 220 Ohm
- kilkudziesięciu kabli różnej długości
- Jednego przycisku do resetowania zegara

Cały układ został osadzony w plastikowym opakowaniu z wywierconymi otworami

4.1.3. Działanie

Osią działania zaprojektowanego przeze mnie weryfikatora jest interakcja z dwoma zewnętrznymi urządzeniami. Jednym z nich jest nośnik danych na którym zapisane zostaną obecności uczesników zajęć jest w moim projektcie pendrive sformatowany w systemie FAT. W zależności od tego czy w danym momencie penrive jest podłączony, urządzenie jest w stanie oczekiwania na drugie z zewnętrznych narzędzie - elektroniczną legitymację studencką. Dodatkowo weryfikator jest zdolny do wyświetlanie komunikatów o swoim stanie na ekranie LCD i wydawania sygnałów dźwiękowych przy pomocy brzęczka. Cżęścią urządzenia jest również zegar RTC, który pozwala śledzić aktualną godzinę. Weryfikator monitoruje swój własny stan i jest zdolny do wykrywania błędów takich jak:

- Błąd inizjalizacji podłączonego pendriva
- Błąd zapisu odczytanych danych na nośniku
- Błąd zegara RTC - spowodowanego najczęściej rozładowaną baterią

Kod mojego weryfikatora projektowałem z myślą o tym, żeby łatwo dało się przedstawić przebieg programu na diagramie. Strukura kodu jest przemyślana w taki sposób, żeby podczas jednego przebiegu pętli głównej wykonywał się jedynie kd powiazany z aktualnym stanem urządzenia. Taki układ nie tylko pozwala zwięźle opowiadać o działaniu urządzenia, ale również ulatwiał testowanie poprawności podczas pisania kodu i montowania urządzenia. Weryfikator w danej chwili może znajdować się w jednym z 12 stanów. Na przedstawionym diagramie stany oczekujące na zdarzenie przdstawiono kolorem kremowym, a stany przejściowe kolorem niebieskim:

- STATE_USB_WAIT - czekanie na penrive - ekran wyświetla informację - "Insert pendrive". Próba inicjalizacji nośnika zajmuje około 5 sekund dlatego wyświetlany czas jest aktualizowany z opóźnieniem

- STATE_USB_INIT - inicjalizacja pendriva - po odnalezieniu pendriva wyświetlana jest informacja "USB Inserted". W przypadku błędu inicjalizacji - przejście do stanu STATE_INIT_ERROR, w razie sukcesu na podstawie aktualnego czasu stworzona i zapiętana zostaje nazwa YY-MM-DD-hh_mm_ss dla pliku który będzie przechowywał zebrane uidy. STATE_CARD_WAIT w przeciwnym

- STATE_CARD_WAIT - oczekiwanie na kartę RFID - ekran: Ścan Card". W razie wykrycia odłączenia penriva przejście do stanu STATE_USB_LOST. W przypadku wykrycia karty, ale błędu w odczycie (np. ze względu na zbyt szybkie odsunięcie karty) wyświetlenie informacji: Read error.

- STATE_CARD_FOUND - zapisywanie Uidu do pliku - otwarcie pliku o zapamiętanej nazwie zapisanie do niego odczytanego Uidu. W razie błędu przy zapisie przejście do stanu STATE_FILE_ERROR, w przypadku sukcesu sygnał dźwiękowy, wyświetlenie informacji Żead Success"i powrót do stanu STATE_CARD_WAIT

- STATE_INIT_ERROR - błąd inicjalizacji pendriva - ekran: "Init Error"

- STATE_USB_REMOVE - czekanie na odłączenie pendriva. Wyświetlacz: Żeinsert USB". W razie wykrycia odłączenia penriva przejście do stanu STATE_USB_LOST.

- STATE_USB_LOST - odłączono pendrive . Ekran: "USB removed", sygnał dźwiękowy oraz przejście do stanu STATE_USB_WAIT

- STATE_FILE_ERROR - błąd zapisu do pliku. Ekran: Ópen file error", przejście do stanu STATE_USB_REMOVE

- STATE_CLOCK_ERROR - błąd zegara - w razie wykrycia problemów z zegarem w którymś ze stanów [STATE_USB_WAIT, STATE_CARD_WAIT, STATE_CARD_FOUND, STATE_USB_INIT, STATE_USB_REMOVE] sterowanie przechodzi do tego stanu. Wyświetlona zostaje informacja: Ćlock Error", następnie sterowanie przechodzi do STATE_CLOCK_ERROR_USB_WAIT

- STATE_CLOCK_ERROR_USB_WAIT - oczekiwanie na pendrive z czasem - wyświetlacz "Insert pendrive with time". Sterowanie pozostaje w tym stanie do czasu kiedy na zewnętrznym nośniku wykryty zostanie plik o nazwie tdav, który w pierwszym wierszu będzie zawierał datę w formacie "YYYY/MM/DD hh:mm:ss". W przypadku wykrycia poprawnego pliku program zapamiętuje datę z pliku tdav i przechodzi w stan STATE_CLOCK_ERROR_CLICK
  - W razie błędu przy inicjalizacji pendriva program przechodzi do stanu STATE_INIT_ERROR
  - W razie wykrycia pendrive i nie wykrycia pliku o nazwie tdav wyświetlona zostaje informacja "Upload time tdav"
  - W razie złego formatu pierwszej linijki w pliku tdav wyświetlona zostaje informacja: "Bad date format"

- STATE_CLOCK_ERROR_CLICK - Czekanie na reset. Wyświetlacz: Ćlick to reset". Sterowanie pozostaje w tym stanie do czasu kiedy zostanie przyciśnięty guzik resetu. Po naciśnięciu guzika na zegarze zostaje ustawiona zapamiętana data, wyświetlony zostaje komunikat "Time set", a sterowanie przechodzi w stan STATE_USB_INIT. Jeśli podczas ustawiania czasu na zagarze wystąpi błąd program przechodzi w stan STATE_CLOCK_ERROR_RESET_FAIL

- STATE_CLOCK_ERROR_RESET_FAIL - błąd resetowania czasu. Wyświetlacz: Ćlock Reset Fail". Ze względu na brak możliwośąciu ustawienia poprawnego czasu jest to stan ostatecznej porażki.

4.1.4. Procedura resetowania zegara RTC

4.1.5. Napotkane problemy

Dwa moduły korzystające z interfejsu SPI

Początko zamiast Arduino Uno planowałem użyć innej wersji tej popularnej płytki - Arduino Leonardo.

Tak jak w wersji którą ostatecznie zrealizowałem, planowałem do układu Leonardo podłączyć USB Host Shield oraz modułu RFID - oba używają intrfejsu SPI. Każdy z tych komponentów osobno współpracował poprawnie z Arduino Leonardo. Problemy pojawiły się kiedy próbwałem podłączyć wszystko w jedną całość i zapisywać odczytane dane z karty magnetycznej na podłączonym pendrivie.

Podczas montowania ukladu problemem nie do przeskoczenia okazało się to, że w przypadku tej wersji płytki piny intefejsu SPI znajdują się w magistrali ICSP, która jest zajęta przez moduł USB w taki sposób, że nie ma możliwości podłączyć tam pinów inych pinów.

Próbowałem jeszcze dolutować kable do magistrali CSP na płytce USB Host Shield, jednak ze względu na to że, są to jedynie milimetrowe wypuski, przepaliłem któreś z połączeń wewnątrz tego układu i ostatecznie musiałem je wyrzucić.
W ostateczności przeszedłem na wersję UNO które interfejs SPI ma doprowadzony również na piny 10-13. Dodatkową modyfikajcą którą musiałem wprowadzić do układu było podłączenie pinu SS (slave select) modułu RFID do innego złącza niż 10, ponieważ był on już wykorzystywany przez USB Host Shield.

Wypełniona pamięć operacyjna

W miarę dołączania kolejnych komponentów do układu, tzn. zegara RTC, brzęczka oraz wyświetlacza LCD, zacząłem natrafiać na problemy z pamięcią.

Pierwsza skończyła się pamięć operacyjna której mój kontroler posiadał jedynie 2KB. Objawiało się to nagłym zawieszaniem się wykonywania programu, co w

pierwszym momencie było dla mnie zaskakujące i niezrozumiałe. Udało mi się rozwiązać ten problem dzięki większej dyscyplinie w korzystaniu z pamięci. Ograniczyłem liczbę zmiennych globalnych, a największy efekt przyniosło przeniesienie wykorzystywanych w programie napisów z pamięci operacyjnej do pamięci flash, do czego służy dyrektywa PROGMEM. Pamięć flash jest przeznaczona do przechwywania kodu programu i nie ma do niej swobodnego dostępu jak w przypadku pamięci operacyjnej. Dodanie kolejnej biblioteki która ułatwiłaby przechowywanie danych w tej przestrzeni nie było możliwym ze względu na graniczone zasoby.

Ostatecznie zdecydowałem się na korzystanie z pamięci flash wczytując dane najt po bajcie. Chcąc więc wyświetlić komunikat na kranie LCD, tak czy inaczej musiałem najpierw wczytać bajtowo łańcuch znaków do pamięci operacyjnej, jednak nie było już potrzeby przechowywać wszystkich łańcuchów znaków jednocześnie w SRAM.

Wypełniona pamięć flash

Przeniesienie łańcuchów znaków do pamięci flash, chociaż poprawiło wydajność pamięci operacyjnej, to spowodowało wyczerpanie przestrzeni przeznaczonej na kod programu.

Pamięć flash jest bardziej pojemna niż SRAM ponieważ w tym wypadku jest to aż 32KB do dyspozycji programisty, jednak duża liczba bibliotek do obsługi peryferiów doprowadziła do wyczerpanie się również tej przestrzeni.

Ułatwieniem w przypadku optymalizowania wykorzystania pamięci flash jest to, że przestrzeń zajmowana przez kod programu jest stała po kompilacji. Nie ma więc obawy że w trakcie przebiegu programu nastąpi nagły wzrost i przekroczenie pojemności. Wytraczy dostosować się do limitu 32 KB.

Udało mi się tego dokonać poprzez ograniczenie liczby funkcji które deklarowałem w moim programie, oraz schowanie komunikatów wysyłanych na serial monitor wewnątrz dyrektyw preprocesora #if DEBUG serial.println(ŚETUP"); #endif. Dzięki temu mogłem łatwo wyłączyć kod programu odpowiedzialny za debugowanie.

W pierszej wersji planowanego przeze mnie weryfikatora, miał on się łączyć z siecią za pomocą modułu WIFI. Bardzo dużo kłopotów sprawił mi jednak układ ESP8266, odpowiedzialny za tę łączność. Po wgraniu sterownika do tego układu przestawał on być wykrywany przez komputer i Arduino
. Pomimo wykorzystania kilku takich układów za każdym razem ojawiał się ten sam problem Nie udało mi się dociec jaka była tego przyczyna. Ostatecznie odszedłem od wykorzytania modułu ESP8266 ze względu na zmianę koncepcji.

4.1.6. Inne koncepcje

Zastanawiając się nad ostateczną formą jaką powinien przyjąć projektowany przeze mnie weryfikator brałem jeszcze pod uwagę układ Rosberry Pi. Ostatecz-

nie zrezygnowałem z tego pomysłu, ze względu na to że takie urządzenie mogłoby wymagać utrzymywania systemu operacyjnego aktualnej wersji. Za wykorzystaniem Arduino przemawiała również cena takiego układu i fakt, że jest on mi bardzo dobrze znany z zajęć Systemów Wbudowanych.

Projektując układ usiałem również podjąć decyzję z jakiego nośnika skorzystać do przenoszenia danych z weryfikatora do aplikacji zarządzającejobecnosciami. Brałem pod uwagę dwie możliwości. Poza opcją na którą się ostatecznie zdecydowałem, brałem jeszcze pod uwagę korzystanie z kart SD

Ze względu na większą dostępnąść i łatwiejsze podłączenie klasycznego pendriva do komputera (nie każdy komputer posiada slot na kartę SD), zdecydowałem się nie tę właśnie możliwość. Później okazało się, że ta decyzja miała swoje konsekwncje.

Po pierwsze USB Host Shield zajął magistralę ICSP, będącą jedynym wyjściem pinów interfejsu SPI w układzie Arduino Leonrado. Po perypetiach wcześniej już przeze mnie opisanych, zdecydowałem się na zmianę wersji płytki na Uno.

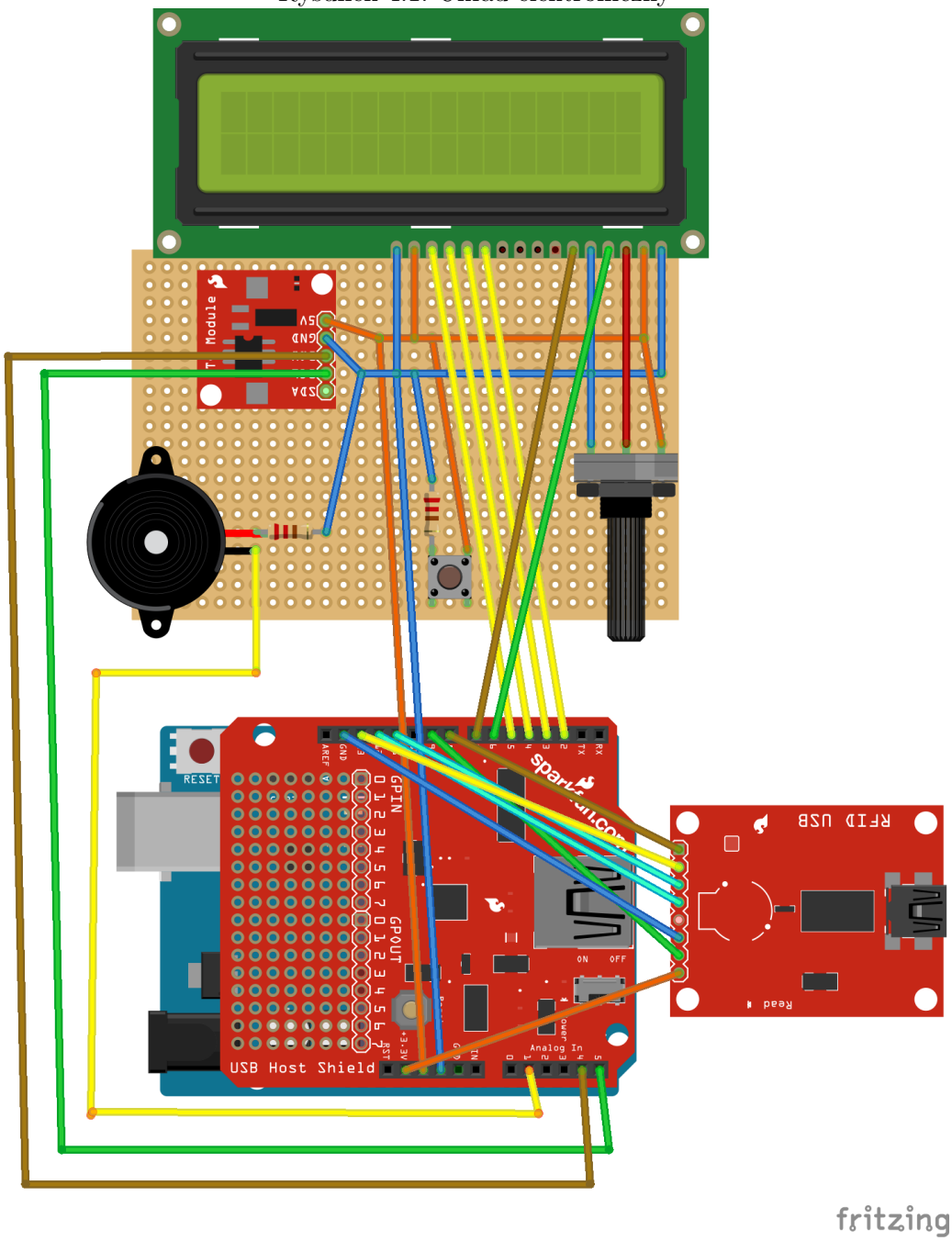
Kolejnym utrudnieniem które zostało postawione na mojej drodze ze względu na korzystanie z pendriva, były problemy z dostępnością biblioteki obsługującej pamięć w tej formie. Podstawowa biblioteka obslugująca układ USB Host Shield nie przwiduje wykorzystania złącza USB do podłączenia pamięci masowej.

Ostatecznie udało mi się znaleźć nieszczególnie popularną bibliotekę do obsługi tego narzędzia - UsbFat, która jednak okazała się być słabo opisana.

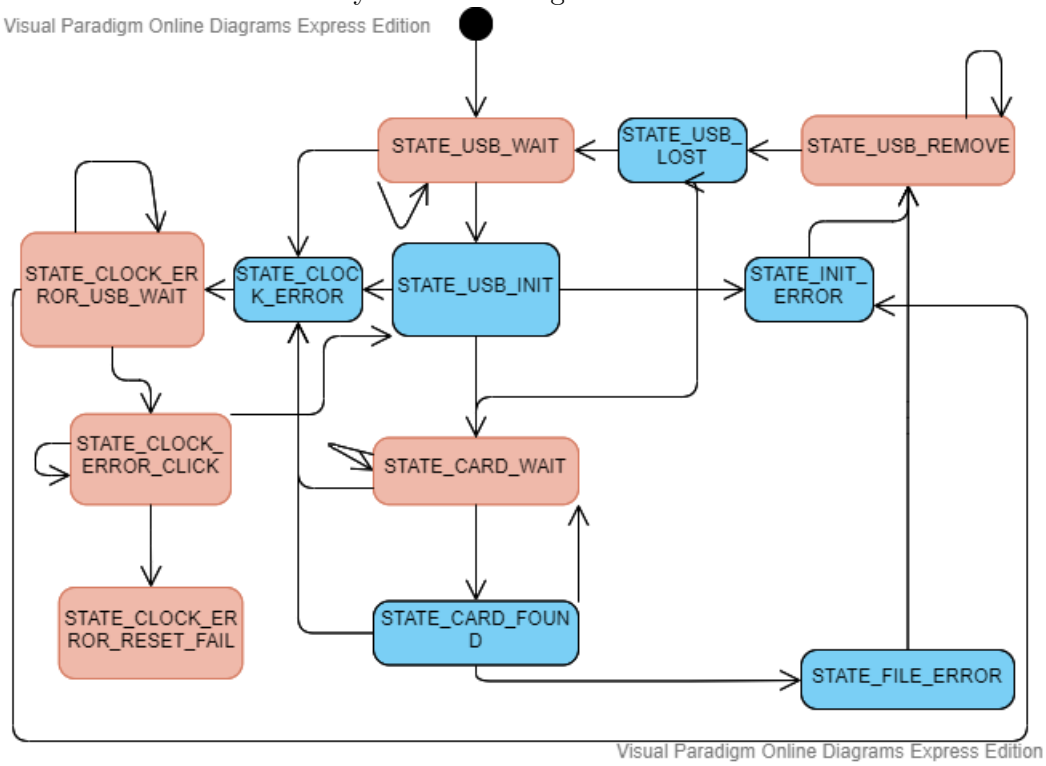
4.2. Menadżer obecności - umożliwiający katalogowanie daych

4.3. Baza studentów - umożliwiająca identyfikakację danych zebranych z ELS

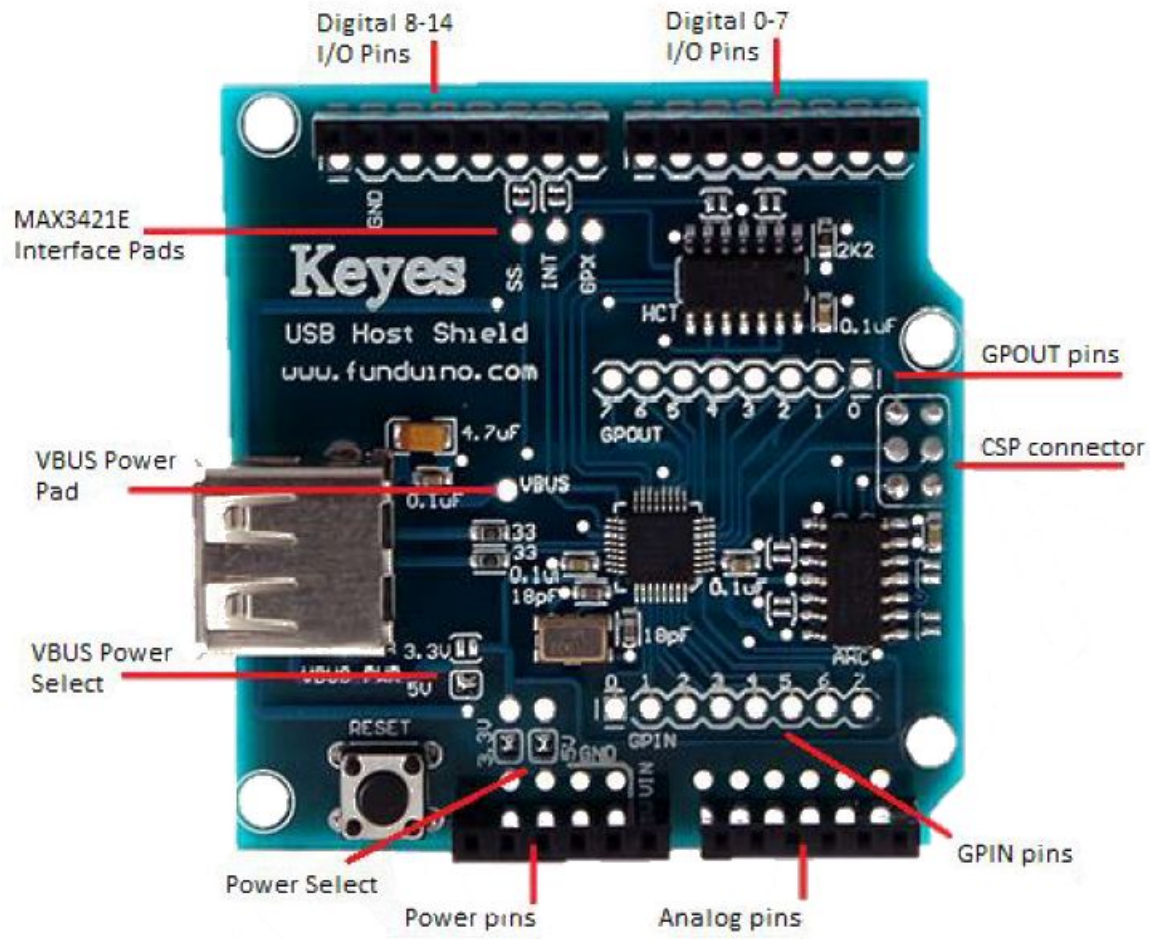
Rysunek 4.1: Układ elektroniczny



Rysunek 4.2: Diagram stanów



Rysunek 4.3: Uklad USB Host shield ze wskzaniem na zakończenia magistrali CSP



Rysunek 4.4: Stałe programu przechowywane w pamięci flash

```
//Deklaracja łańcucha znaków w pamięci operacyjne
const char LCD_INSERT[] PROGMEM = "Insert pendrive";

//Deklaracja łańcucha znakóW w pamięci flash
const char LCD_INSERT[] = "Insert pendrive";

//Ładowanie stałej z pamięci flash do pamięci operacyjnej
// const_ptr - wskazuje na początek łańcucha znakó w pamięci flash
for (byte aux_byte = 0; aux_byte < strlen_P(const_ptr); aux_byte++) {
    lcd.print((char)pgm_read_byte_near(const_ptr + aux_byte));
}
```

Rozdział 5.

Możliwości rozwoju

- dostęp do usos i propagowanie zebranych danych na stronach uczelnianych
- Udostępnienie systemu dla studentów i umożliwienie monitorowania swoich obecności (logowanie tylko przez uczelnianego maila, albo konta utworzone wyłącznie przez administratora)
- powiadomienia mailowe do studenta o zbliżającym się wyczerpaniu limitu nieobecności

Rozdział 6.

Wnioski

...