
DOKUMENTACJA PROJEKTU

Z SYSTEMÓW WBUDOWANYCH

AUTORZY: KASPER SAPAŁA, JAKUB PŁOTNIKOWSKI

Spis treści

1 Opis realizowanego zadania	3
1.1 Realizowany temat	3
1.2 Wymagania technologiczne	3
1.3 Oceniane aspekty	3
1.4 Oczekiwana funkcjonalność	3
2 Instrukcja dla użytkownika	4
2.1 Wymagania do korzystania z czytnika	4
2.1.1 Posiadane przedmioty	4
2.1.2 Zainstalowane programy	4
2.2 Konfiguracja płytki oraz czytnika	5
2.3 Uruchomienie projektu	6
2.4 Korzystanie z czytnika	7
2.4.1 Pierwszy ekran	7
2.4.2 Drugi ekran	8
2.4.2.1 UID	9
2.4.2.2 Avatar	9
2.4.2.3 Informacja o klikniętym avatarze	10
2.4.2.4 Avatars do wyboru	11
2.4.2.5 Informacja o numerze strony z avatarami	13
2.4.2.6 Potwierdzenie wyboru avatara	13
2.4.2.7 Zapis danych na kartę	14
2.4.2.8 Pojawienie się zaktualizowanego avatara	15
3 Instrukcja dla planujących rozwijać projekt	16
3.1 Opis funkcji dostępnych w pliku rfid.h	16
3.1.1 rfid_configure	16
3.1.2 rfid_init	16
3.1.3 rfid_reset	17
3.1.4 rfid_antenna_on	17
3.1.5 rfid_self_test	17
3.1.6 rfid_set_bit_mask	17
3.1.7 rfid_clear_bit_mask	17
3.1.8 rfid_read_register	18
3.1.9 rfid_write_register	18
3.1.10 rfid_cs_write	18
3.1.11 rfid_set_gain	18
3.1.12 rfid_get_gain	18

3.1.13 rfid_read_version	19
3.1.14 rfid_reqa_or_wupa	19
3.1.15 rfid_reqa	19
3.1.16 rfid_to_card	19
3.1.17 rfid_transcive_data	20
3.1.18 rfid_calc_crc	20
3.1.19 rfid_select_tag	20
3.1.20 rfid_authenticate	21
3.1.21 rfid_card_read	21
3.1.22 rfid_card_write	22
3.1.23 rfid_card_transceive	22
3.1.24 rfid_is_new_card	22
3.1.25 rfid_halt	22
3.1.26 rfid_stop_crypto	22
4 Problemy napotkane podczas tworzenia projektu	23
4.1 Problemy z konfiguracją projektu w CubeMX	23
4.2 Problemy z działaniem portu UART na module RFID RC522	23
4.3 Problemy z aktualizacją komponentów na ekranie płytki	24
4.3.1 Problem z responsywnością ekranu	24
4.3.2 Problem z aktualizowaniem komponentów odpowiedzialnych za zmieniający się tekst na ekranie	24
5 Bibliografia	25

Opis realizowanego zadania

1.1 Realizowany temat

Realizowaliśmy ósmy temat z zadań o charakterze inżynierskim - Obsługę modułu RFID RC522

1.2 Wymagania technologiczne

- platforma sprzętowa: STM32 dostępny w laboratorium (STM32F4, STM32F7)
- język programowania: C,
- darmowe (najlepiej na wolnej licencji) narzędzia do rozwoju oprogramowania.

1.3 Oceniane aspekty

- przekonująca demonstracja działania,
- zawarte w sprawozdaniu kreatywne pomysły zastosowań,
- stopień uzyskania zadanej funkcjonalności,
- styl implementacji (przejrzystość kodu źródłowego).

1.4 Oczekiwana funkcjonalność

- urządzenie demonstrujące działanie modułu RFID RC522 z mikrokontrolerem STM32 w przykładowej, użytecznej aplikacji, np. kontroli dostępu.

Instrukcja dla użytkownika

2.1 Wymagania do korzystania z czytnika

2.1.1 Posiadane przedmioty

- czytnik RFID RC522
- odpowiednia, kompatybilna karta - przykładem takiej karty jest: MIFARE Classic 1k
- płytka STM32F469 MCU
- kabel Mini USB
- co najmniej 6 męsko-męskich przewodów połączeniowych
- płytka stykowa

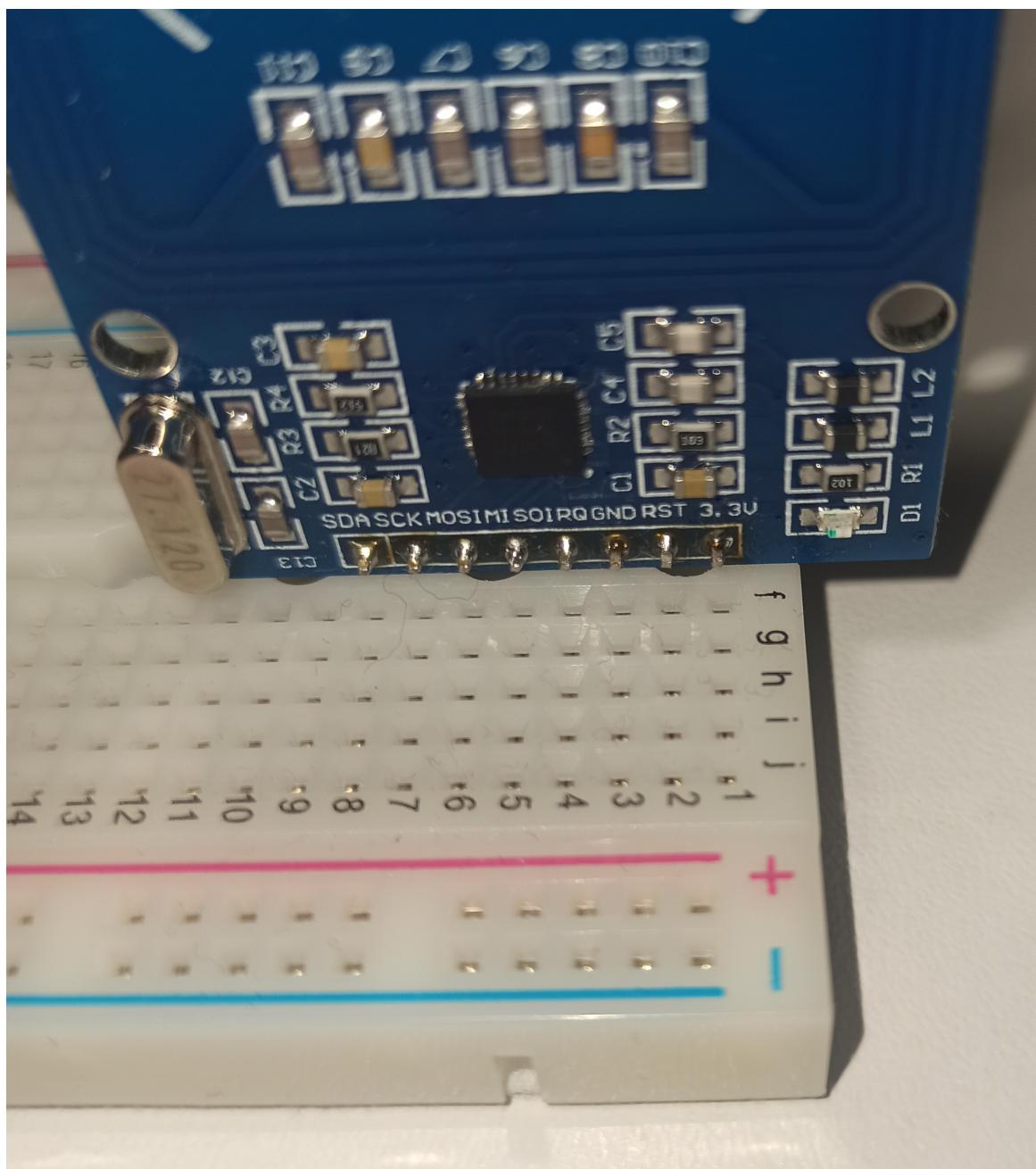
2.1.2 Zainstalowane programy

- kompilator gcc
- make
- STM32CubeIDE - środowisko programistyczne, które przysłuży nam się przy uruchamianiu projektu i przy ewentualnych zmianach
- git

2.2 Konfiguracja płytki oraz czytnika

Pierwszym krokiem jaki powinniśmy wykonać jest przylutowanie pinów z zestawu czytnika RFID do jego głównej części.

Gdy już to zrobimy, pozostaje tylko wpiąć je do płytki stykowej tak, aby napis na czytniku: "3.3V" znajdował się w pierwszym wierszu płytka stykowej, a napis "SDA" znajdował się w wierszu ósmym - właściwe wpięcie czytnika w płytę stykową przedstawione jest na załączonym zdjęciu:



Kolejnym krokiem będzie odpowiednie wpięcie męsko-męskich przewodów połączeniowych w płytę stykową oraz w płytę STM.

Odpowiednie połączenie przedstawione jest w poniższej tabelce:

Numer wiersza z płytka stykowej	Pin z płytka STM
1	3v3
3	GND
5	A1
6	A2
7	D13
8	D10

2.3 Uruchomienie projektu

Kolejnymi krokami jakie trzeba podjąć jest uruchomienie projektu - jego komplikacja i wgranie na płytę STM.

Aby tego dokonać, należy uruchomić terminal w wybranym folderze, a następnie wykonać komendę:

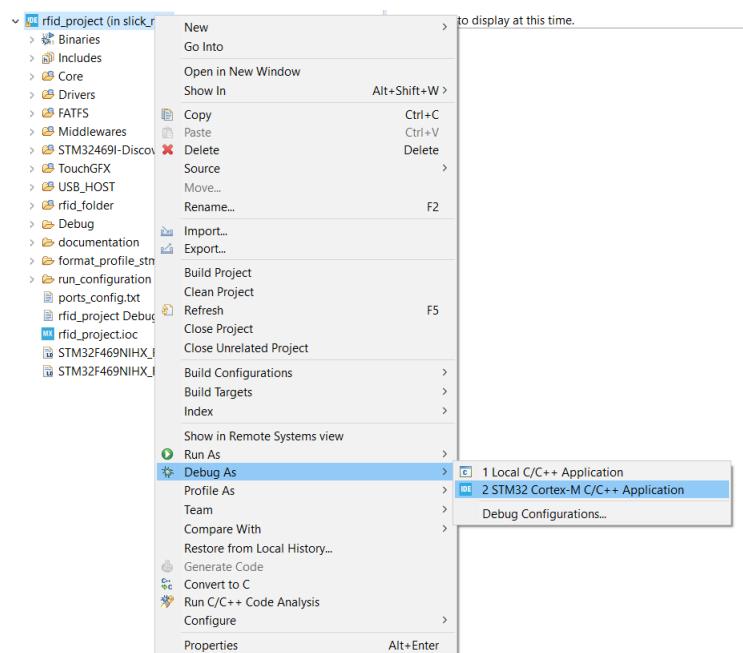
```
git clone https://github.com/jakubowiczish/slick_rfid.git
```

W owym katalogu klikamy 2 razy lewym przyciskiem myszy na plik o nazwie: ".project".

Nasze uprzednio zainstalowane środowisko powinno się uruchomić.

Następnie powinniśmy podpiąć płytę STM do komputera przy użyciu kabla Mini USB.

W uruchomionym środowisku należy kierować się krokami przedstawionymi na obrazku poniżej:



Jeśli wszystko przebiegnie pomyślnie, na płytce pojawią się pierwsze instrukcje dla użytkownika.

2.4 Korzystanie z czytnika

Po wgraniu programu na płytę można rozpocząć korzystanie z interfejsu.

2.4.1 Pierwszy ekran

Pierwszym ekranem po uruchomieniu płytki jest ekran, który zachęca nas do przyłożenia naszej karty do czytnika.

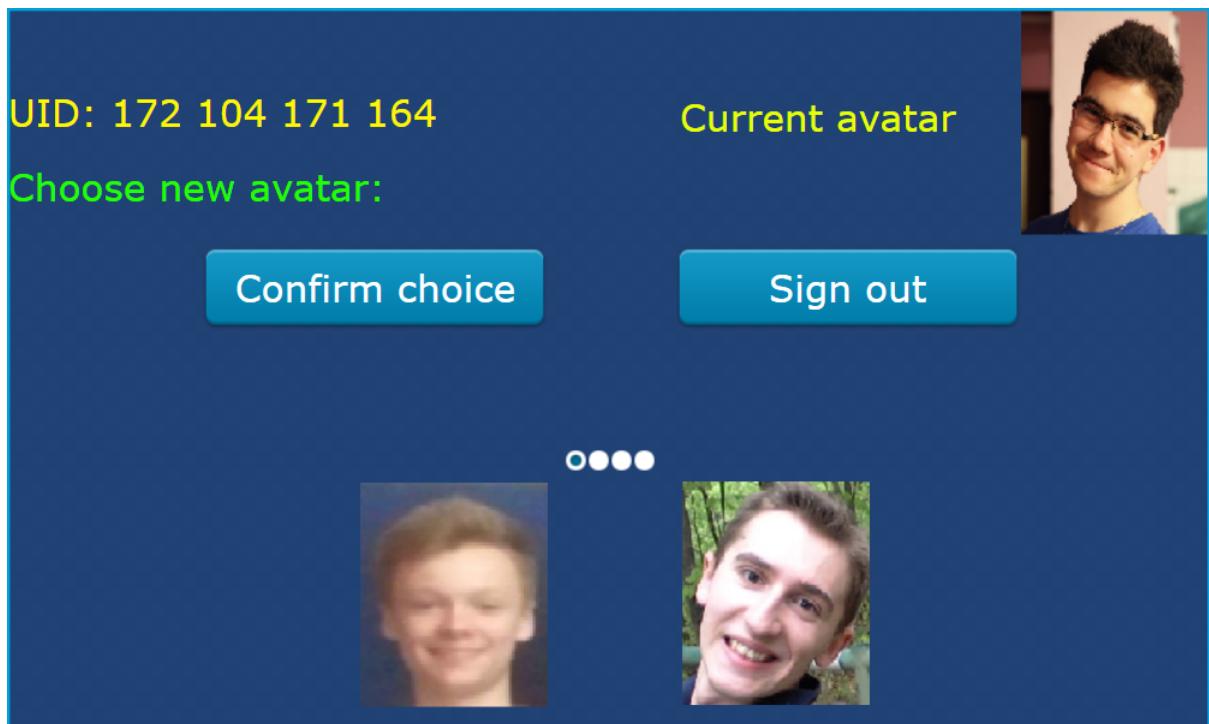
Na płytce prezentuje się on następująco:



Sugeruje on nam, że powinniśmy przyłożyć kartę do czytnika.

2.4.2 Drugi ekran

Po przyłożeniu naszej karty do czytnika (gdy nie wystąpi żaden błąd) zostajemy przeniesieni do kolejnego ekranu, na którym zostaną wyświetcone dalsze informacje:

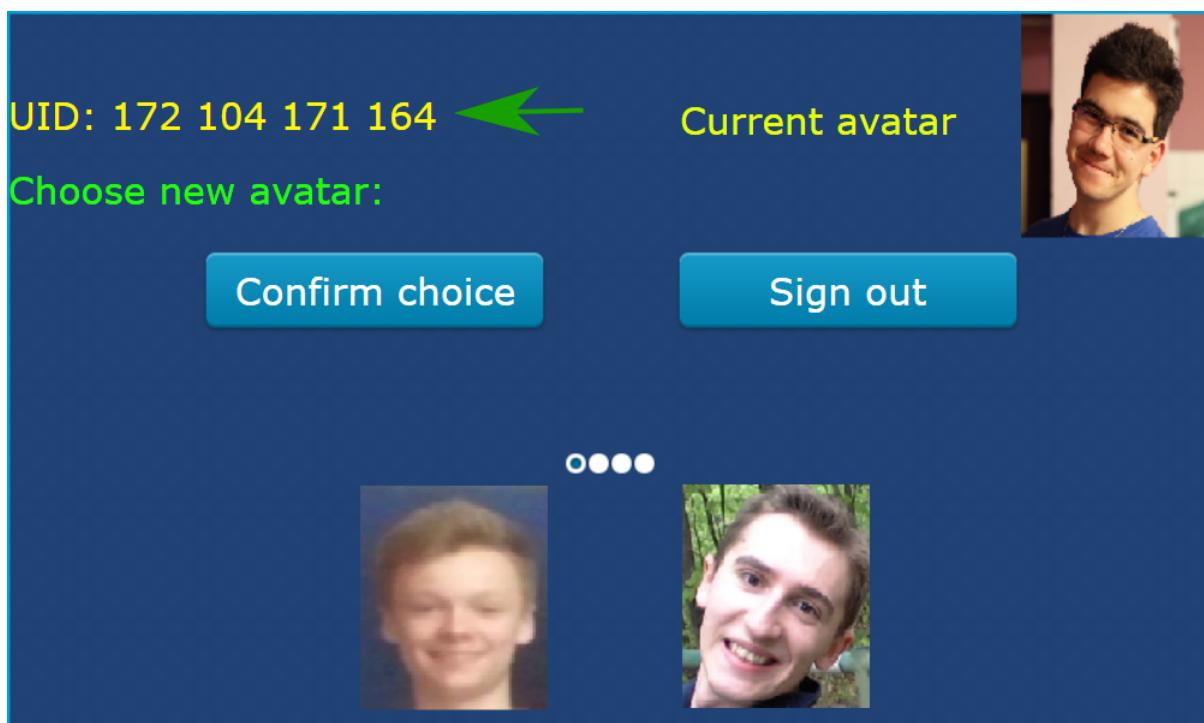


Gdy natomiast wystąpi jakaś komplikacja związana z odczytem danych z karty, pozostaniemy na pierwszym ekranie.

2.4.2.1 UID

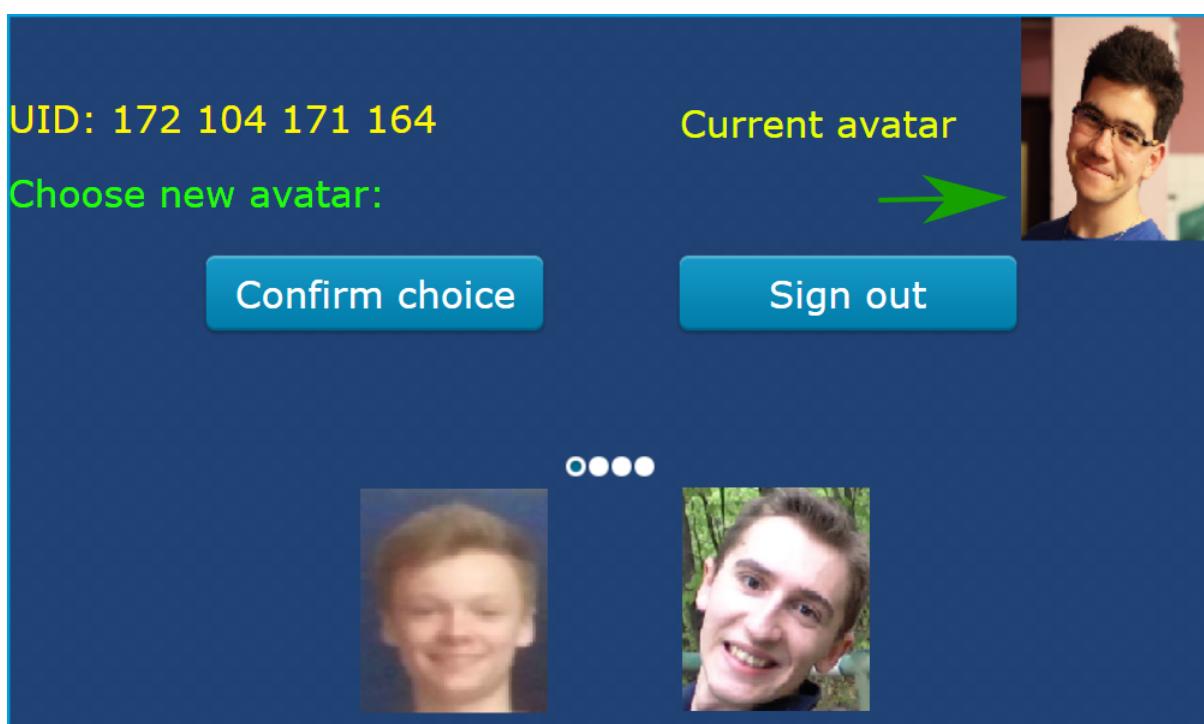
Na ekranie płytki możemy ujrzeć różne, zczytane z płytki wartości:

W lewym górnym rogu zostaje wyświetlone UID karty



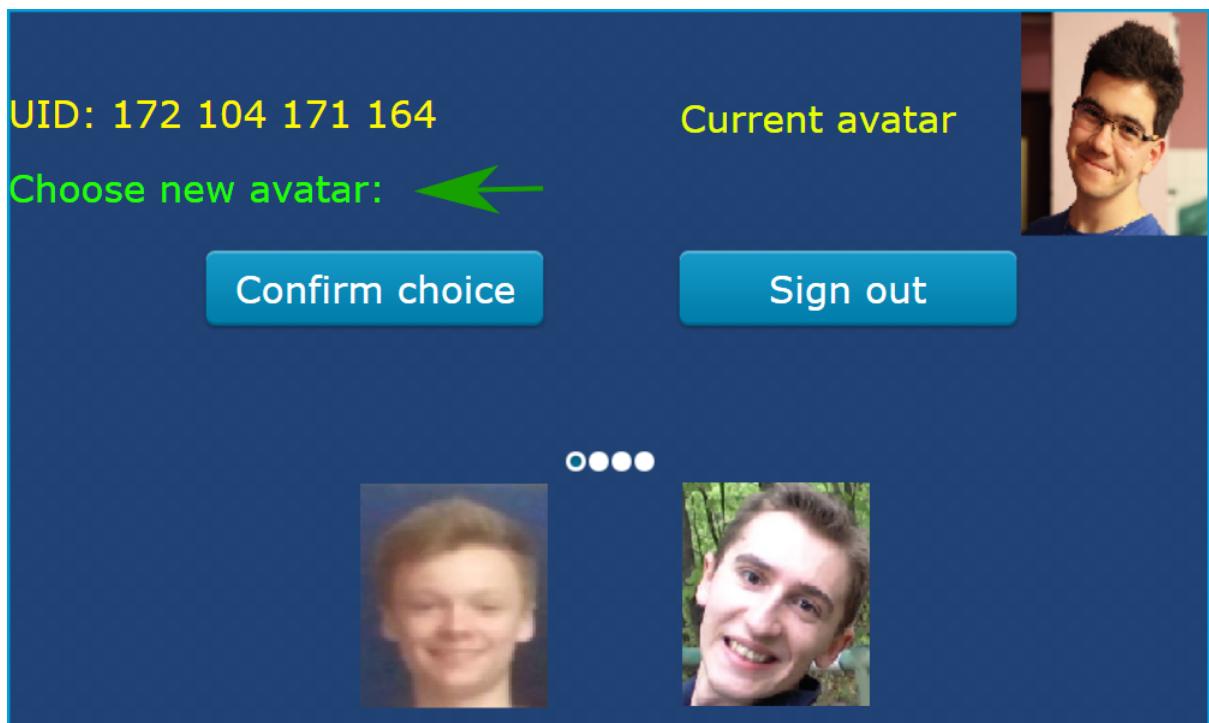
2.4.2.2 Avatar

W prawym górnym rogu zostaje wyświetlony nasz aktualny avatar

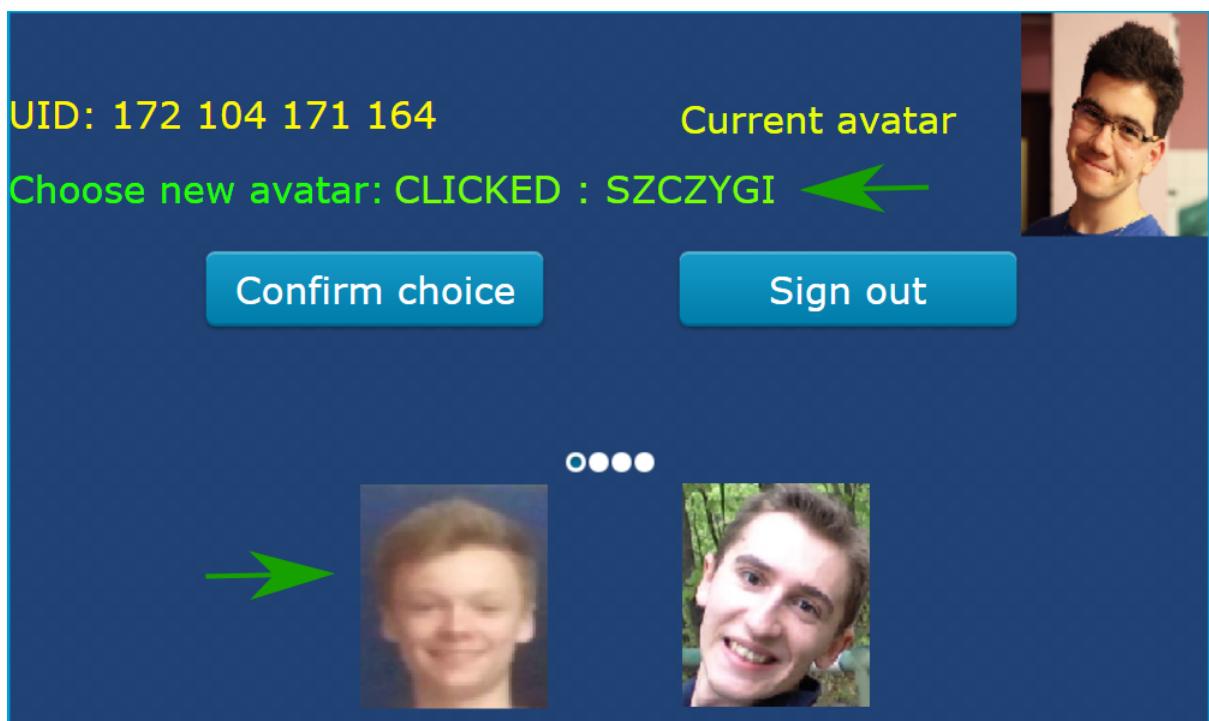


2.4.2.3 Informacja o klikniętym avatarze

Obecny widok zawiera również komunikat o tym, aby wybrać nowy avatar, który będzie aktualizowany zaraz po kliknięciu avatara z panelu poniżej:



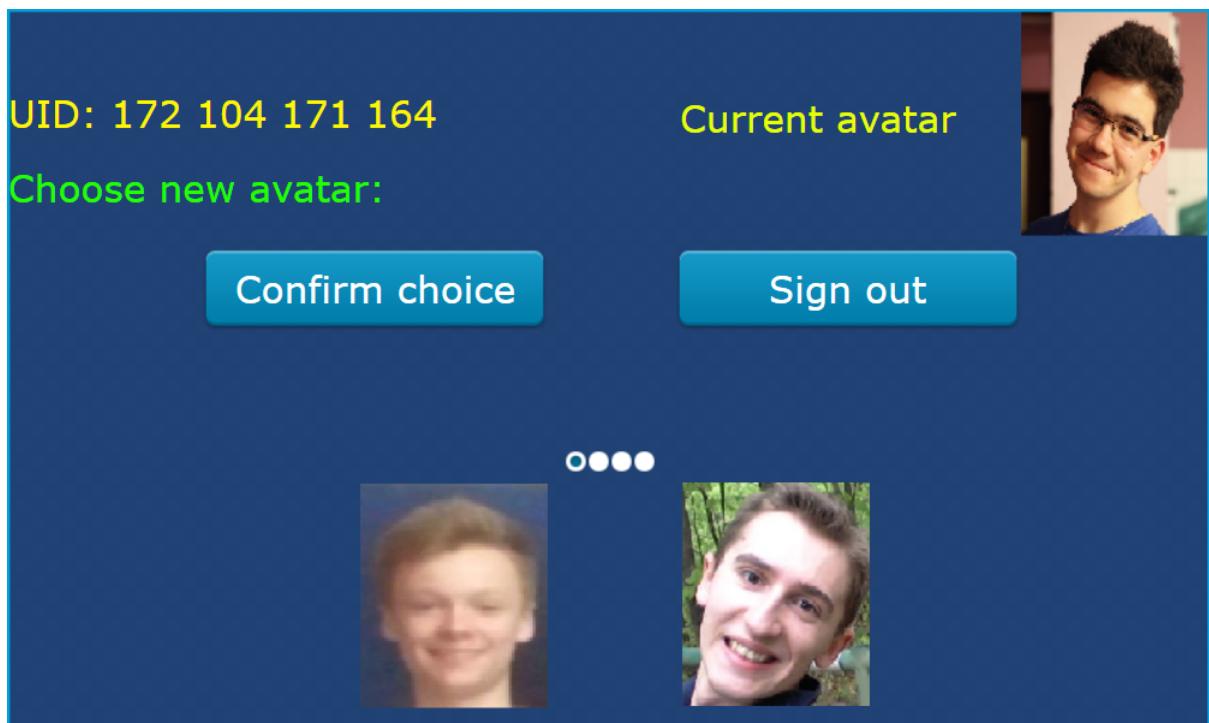
Gdy klikniemy na jakikolwiek avatar, zostaniemy o tym poinformowani:



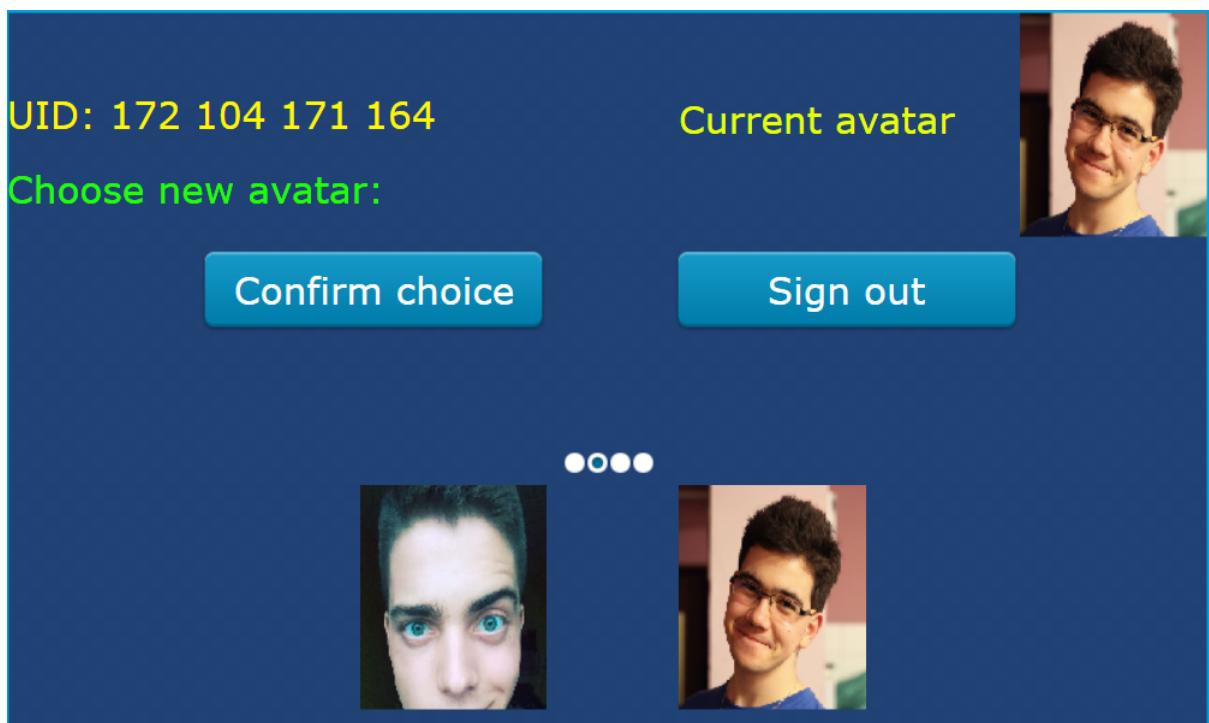
2.4.2.4 Avatary do wyboru

Do wyboru jest 8 avatarów, każdy jest unikalny, posiada inny identyfikator i inną nazwę. Aby wybrać odpowiedni avatar, możemy się poruszać się w dolnym pasku na 4 różnych stronach:

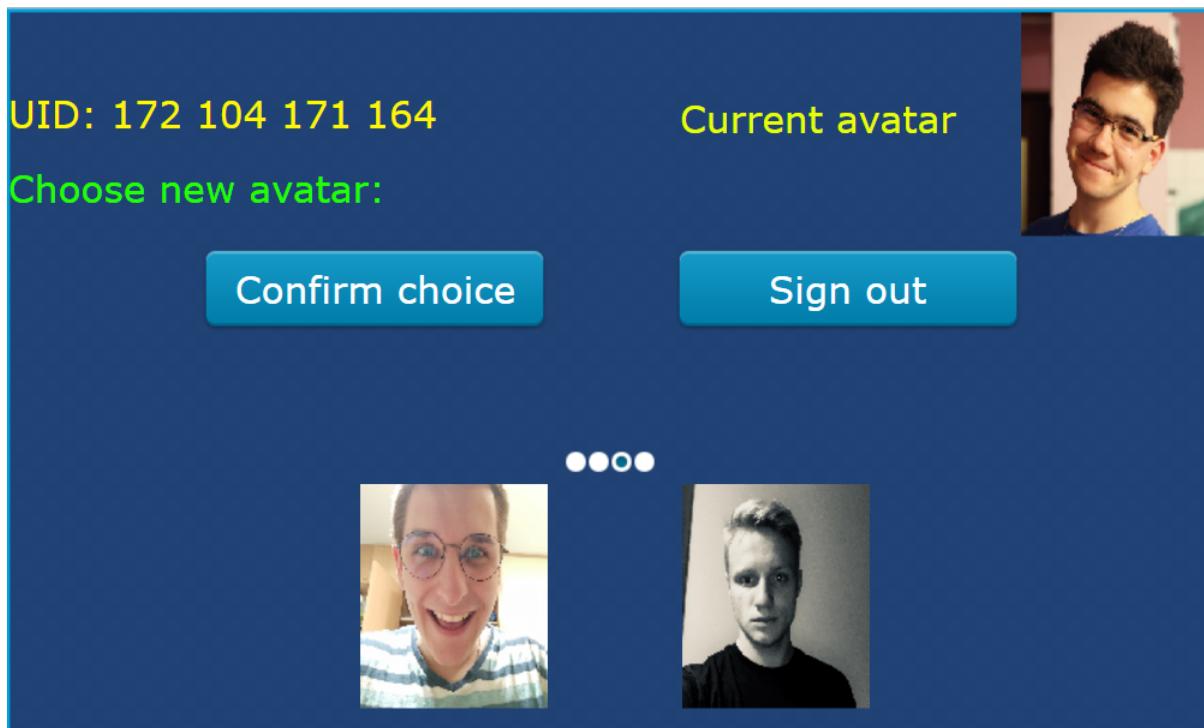
Strona 1



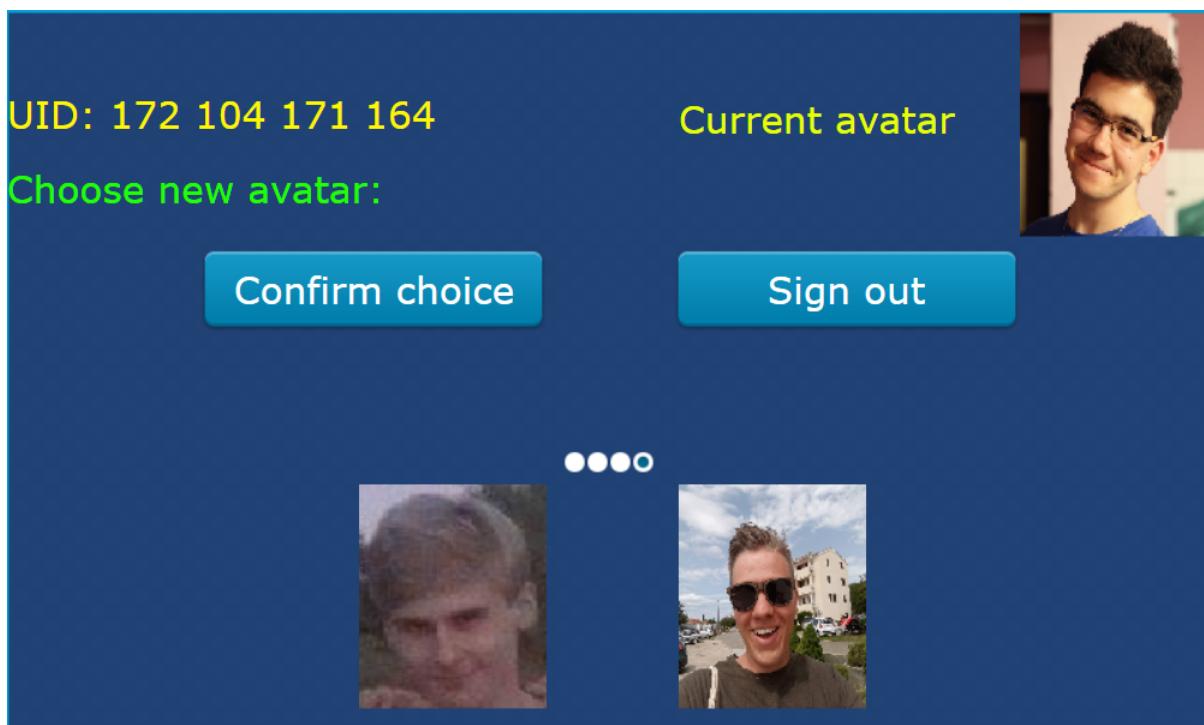
Strona 2



Strona 3

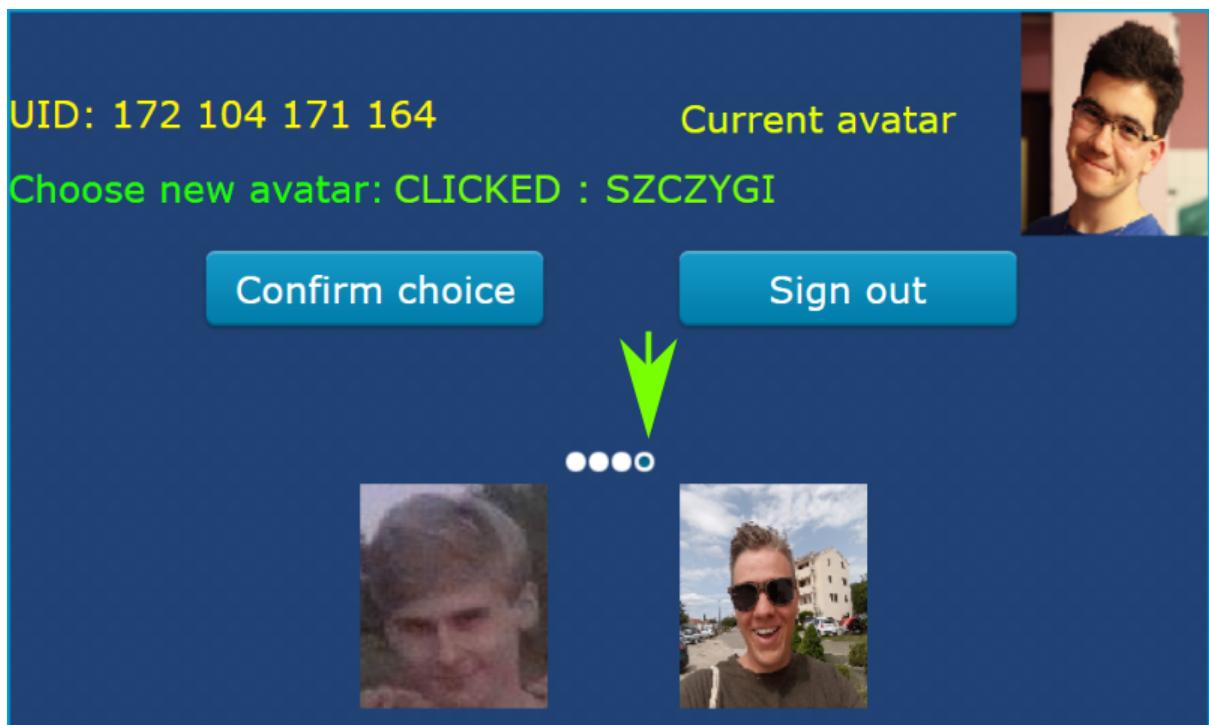


Strona 4



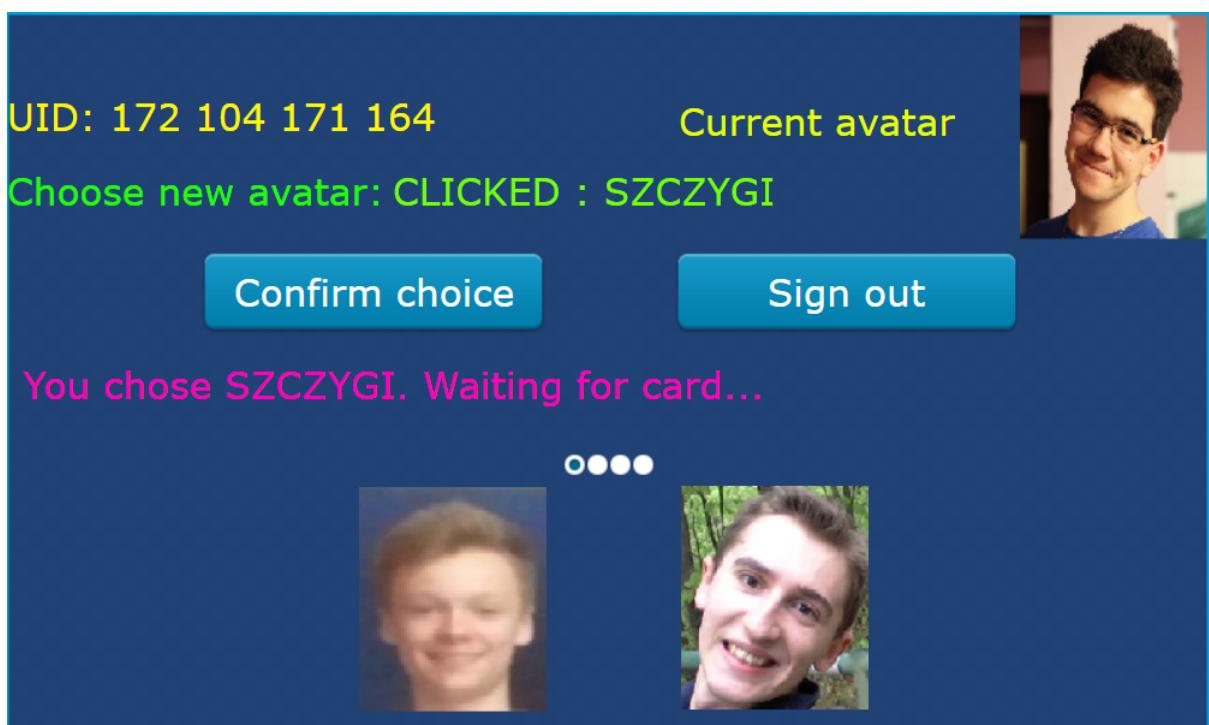
2.4.2.5 Informacja o numerze strony z avatarami

Możemy zaobserwować, że widoczny jest wskaźnik oznaczający stronę na której się aktualnie znajdujemy:



2.4.2.6 Potwierdzenie wyboru avatara

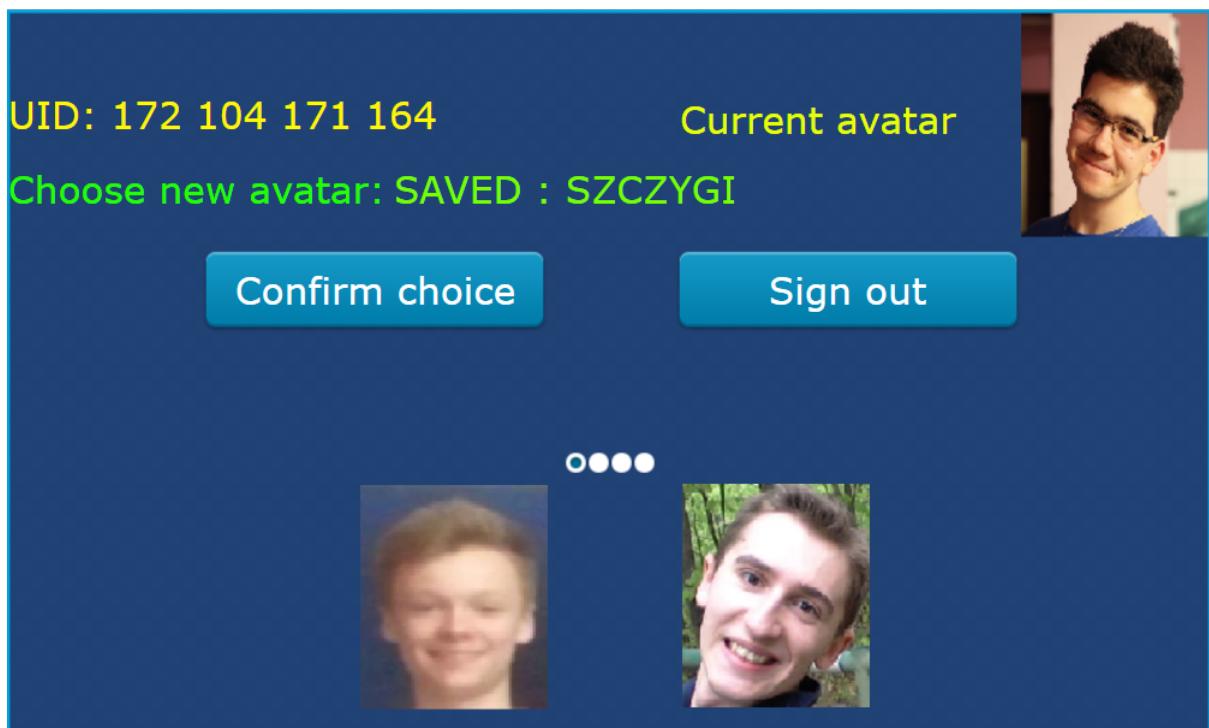
Aby dokonać wyboru avatara, musimy najpierw go kliknąć i gdy już uznamy, że jest to nasz ostateczny wybór, klikamy przycisk z napisem "Confirm choice":



2.4.2.7 Zapis danych na kartę

Następnie czytnik czeka, aż przyłożymy kartę aby dokonać na nią zapisu naszego wyboru.

Gdy już to zrobimy, zostaniemy poinformowani o aktualizacji statusu:



2.4.2.8 Pojawienie się zaktualizowanego avatara

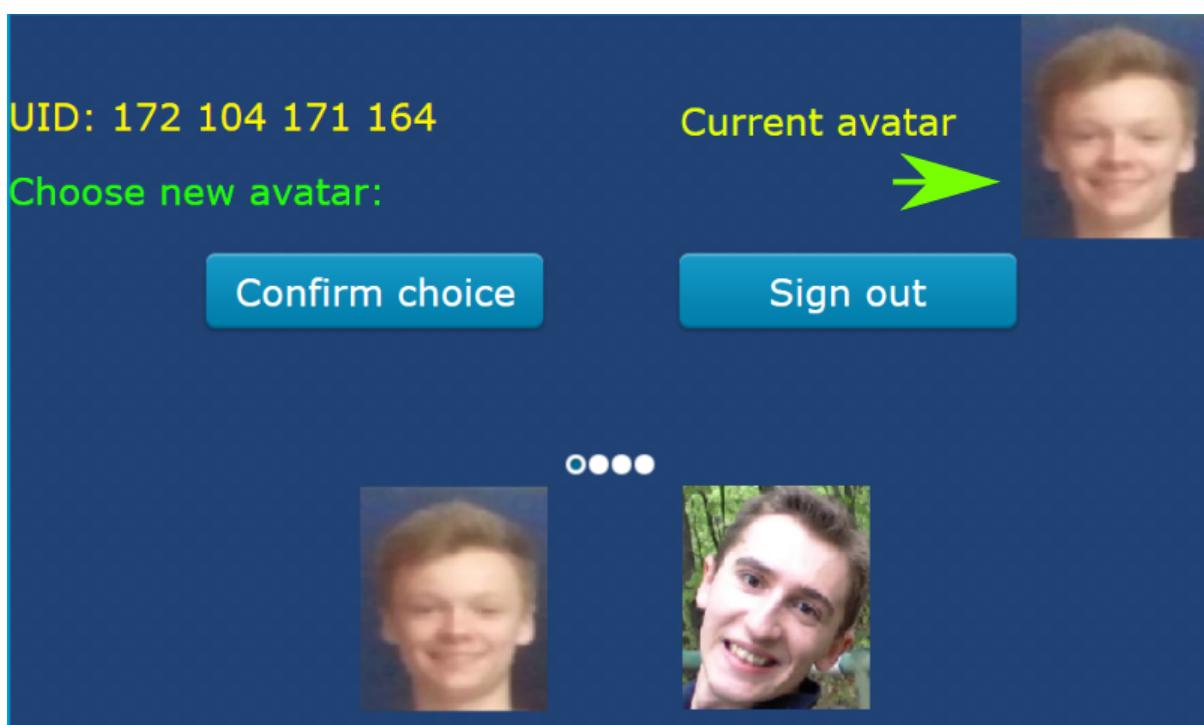
Po zapisie danych na kartę dalej będzie jednak będzie widoczny stary avatar.

Aby ujrzeć nasz nowy avatar, możemy kliknąć przycisk z napisem "Sign out". Możemy również zresetować płytke.

Zostaniemy z powrotem przeniesieni na pierwszy ekran i ponownie poproszeni o przyłożenie karty:



Następnie zostaniemy przeniesieni na kolejny ekran, ale naszym oczom ukaże się już zaktualizowany avatar:



Instrukcja dla planujących rozwijać projekt

Projekt został podzielony na dwa moduły:

- rfid - część odpowiedzialna za komunikację z czytnikiem kard rfid,
- TouchGFX - moduł związany z interfejsem użytkownika

3.1 Opis funkcji dostępnych w pliku rfid.h

3.1.1 rfid_configure

Funkcja wywoływana na początku działania programu, służy do inicjalizacji odpowiednich pinów i portów GPIO, niezbędnych do działania czytnika kart.

```
1 void rfid_configure(  
2     SPI_HandleTypeDef *s,  
3     GPIO_TypeDef *cs_po,  
4     uint16_t cs_pi,  
5     GPIO_TypeDef *reset_po,  
6     uint16_t reset_pi  
7 );
```

3.1.2 rfid_init

Funkcja, którą należy wykonać po inicjalizacji pinów. Służy do wpisania do rejestrów czytnika kart, odpowiednich wartości inicjalizujących. Oto Kilka ważniejszych wpisów w rejestyry:

- przeprowadzenie resetu wszystkich rejestrów,
- ustawienie mocy anteny: 48dB
- włączenie anteny

```
1 void rfid_init();
```

3.1.3 rfid_reset

Funkcja wywoływana przez rfid_configure(), dokonuje resetu czytnika.

```
1 void rfid_reset();
```

3.1.4 rfid_antenna_on

Funkcja wywoływana przez rfid_configure(), włącza antenę.

```
1 void rfid_antenna_on();
```

3.1.5 rfid_self_test

Funkcja, która przeprowadza test działania czytnika, poprzez wpisanie w odpowiednie rejesty wartości, oraz wpisanie 25 zer w kolejkę fifo. Po przeprowadzeniu testu, w kolejce fifo dostajemy informację o wersji czytnika rfid oraz kolejne liczby zgodne z dokumentacją. Tak wygląda zawartość kolejki fifo, gdy test się powiedzie:

```
00h, EBh, 66h, BAh, 57h, BFh, 23h, 95h,  
D0h, E3h, 0Dh, 3Dh, 27h, 89h, 5Ch, DEh,  
9Dh, 3Bh, A7h, 00h, 21h, 5Bh, 89h, 82h,  
51h, 3Ah, EBh, 02h, 0Ch, A5h, 00h, 49h,  
7Ch, 84h, 4Dh, B3h, CCh, D2h, 1Bh, 81h,  
5Dh, 48h, 76h, D5h, 71h, 061h, 21h, A9h,  
86h, 96h, 83h, 38h, CFh, 9Dh, 5Bh, 6Dh,  
DCh, 15h, BAh, 3Eh, 7Dh, 95h, 03Bh, 2Fh
```

```
1 void rfid_self_test();
```

3.1.6 rfid_set_bit_mask

Funkcja pomocnicza, ustawiająca maskę na obecną zawartość rejestru danego.

```
1 void rfid_set_bit_mask(uint8_t reg, uint8_t mask);
```

3.1.7 rfid_clear_bit_mask

Funkcja pomocnicza, usuwająca wcześniej ustawioną maskę na wartość rejestru (poprzez maskowanie odwrotności bitowej podanej maski).

```
1 void rfid_clear_bit_mask(uint8_t reg, uint8_t mask);
```

3.1.8 rfid_read_register

Funkcja zwracająca zawartość rejestru o danym adresie. Wewnątrz następuje modyfikacja adresu rejestru na:

```
addr = (addr << 1) | 0x80;
```

Dopisanie 0x80 do końca adresu mówi rfid, że chcemy czytać dany rejestr (nie pisać).

```
1 uint8_t rfid_read_register(uint8_t addr);
```

3.1.9 rfid_write_register

Funkcja wpisująca dane do rejestrów o podanym adresie. Wewnątrz następuje modyfikacja adresu rejestru na:

```
addr = (addr << 1) & 0x7E;
```

Dopisanie 0x7E do końca adresu mówi rfid, że chcemy wpisać do danego rejestrów (nie z niego czytać).

```
1 void rfid_write_register(uint8_t addr, uint8_t val);
```

3.1.10 rfid_cs_write

Funkcja zmienia stan pinu CS (Chip Select potrzebny do działania SPI) na wysoki lub niski. Jest ona wywoływaną przez funkcje rfid_read_register oraz rfid_write_register.

Ustawienie na stan niski:

```
rfid_cs_write(GPIO_PIN_RESET);
```

Ustawienie na stan wysoki:

```
rfid_cs_write(GPIO_PIN_SET);
```

```
1 void rfid_cs_write(uint8_t val);
```

3.1.11 rfid_set_gain

Ustawia zysk energetyczny anteny (antenna gain). Im większą wartość przekaże się do funkcji, z tym większej odległości będzie można czytać karty. Wartość maksymalna to 0xff. Zysk energetyczny jest ustawiany na maksymalny poprzez funkcję rfid_init.

```
1 void rfid_set_gain(uint8_t mask);
```

3.1.12 rfid_get_gain

Zwraca obecnie ustawiony zysk energetyczny anteny (antenna gain).

```
1 uint8_t rfid_get_gain();
```

3.1.13 rfid_read_version

Zwraca wersję odbiornika. Możliwe są dwie wartości:

- 0x91: wersja 1.0
- 0x92: wersja 2.0

```
1 uint8_t rfid_read_version();
```

3.1.14 rfid_reqa_or_wupa

Wysyła do karty komendę REQA lub WUPA. Zgodnie z dokumentacją, aby zacząć komunikację z kartą, trzeba wysłać do niej jedną z tych dwóch komend.

```
1 rfid_status_t rfid_reqa_or_wupa(  
2     uint8_t command,  
3     uint8_t *response,  
4     uint8_t *response_size  
5 );
```

3.1.15 rfid_reqa

Wysyła do karty komendę REQA. Pod spodem wywołuje funkcję rfid_reqa_or_wupa. W projekcie jest używana tylko komenda REQA, rozpoczynająca komunikację z kartą, więc stworzona została ta funkcja pomocnicza.

```
1 rfid_status_t rfid_reqa(uint8_t *response, uint8_t *response_size);
```

3.1.16 rfid_to_card

Funkcja zapewniająca komunikację z kartą. Jest wykorzystywana przez wszystkie funkcje chcące komunikować się z kartą. Jej argumenty:

- command - komenda dla czytnika rfid. Zazwyczaj CMD_TRANSCEIVE, wtedy czytnik wysła dane do karty i czeka na odpowiedź,
- waitIRq - bajt używany do sprawdzenia czy poprawnie wykonano komendę. Np dla CMD_TRANSCEIVE, jest to 0x30,
- *send_data - dane do wysłania do karty,
- send_len - rozmiar tablicy send_data,
- *back_data - tablica wypełniana bajtami zwróconymi jako odpowiedź z karty,
- *back_len - rozmiar tablicy back_data, ustawiany przy odbiorze.
- *valid_bits - nie koniecznie zwrócony zostanie nam pełny bajt, jest to ilość bitów ostatniego bajtu jaką należy barć pod uwagę,

- rx_align - ustawia od jakiego bitu wpisywane będą wartości do tablicy *back_data; zazwyczaj 0,
- check_CRC - jeśli true, sprawdza czy suma kontrolna złożona z dwóch ostatnich bajtów *back_data się zgadza.

```

1 rfid_status_t rfid_to_card(
2     uint8_t command,
3     uint8_t waitIRq,
4     uint8_t *send_data,
5     uint8_t send_len,
6     uint8_t *back_data,
7     uint8_t *back_len,
8     uint8_t *valid_bits,
9     uint8_t rx_align,
10    bool check_CRC
11 );

```

3.1.17 rfid_transcive_data

Funkcja pomocnicza wywołująca rfid_to_card z command ustawionym na CMD_TRANSCEIVE oraz z waitIRq ustawionym na 0x30. Pozostałe argumenty takie same jak w przypadku funkcji rfid_to_card.

```

1 rfid_status_t rfid_transcive_data(
2     uint8_t *send_data,
3     uint8_t send_len,
4     uint8_t *back_data,
5     uint8_t *back_len,
6     uint8_t *valid_bits,
7     uint8_t rx_align,
8     bool check_CRC
9 );

```

3.1.18 rfid_calc_crc

Oblicza sumę kontrolną CRC podanej tablicy. Korzysta z gotowych mechanizmów dostarczonych przez czytnik rfid, wpisuje do odpowiednich rejestrów i do kolejki fifo dane. Czytnik rfid sam oblicza sumę kontrolną i sprawdza czy się zgadza.

```
1 rfid_status_t rfid_calc_crc(uint8_t *tab, uint8_t len, uint8_t *out);
```

3.1.19 rfid_select_tag

Dokonuje wyboru karty, jako tej, z której będzie następowała później komunikacja. Funkcja ta czyta kolejne bity zwracane przez kartę, w tym jej numer UID (identyfikator), SAK

(Select Acknowledge). Dodatkowo sprawdza czy zgadza się suma kontrolna CRC. Jeśli wszystko się powiedzie:

- zwracany jest status MI_OK,
- w podanej jako parametr tablicy uid_tab zapisywany jest UID karty,
- w size znajduje się wielkość UID karty (czyli 4, bo tylko takie karty obsługujemy),
- w sak znajduje się kod zależny od typu karty. Dla testowych kart było to 0x08.

```
1 rfid_status_t rfid_select_tag(uint8_t *uid_tab, uint8_t *size, uint8_t *sak);
```

3.1.20 rfid_authenticate

Wysyła do karty komendę autentykacji. Dzięki niej można pisać / czytać zawartość karty. Jej argumenty to:

- command - MIF_AUTHENTA jeśli chcemy czytać zwartość karty lub MIF_AUTHENTB jeśli chcemy wpisywać do karty,
- blockAddr - w kartach mifare classic 1k mamy 16 bloków, więc jest to wartość z przedziału 0 - 15. W bloku 0, w sektorze 0 mamy wpisane UID karty + dodatkowe wartości od producenta, ta część jest nie modyfikowalna - tylko do odczytu.
- key - każdy blok jest zabezpieczony kodem, osobny kod w przypadku chęci czytania i osobny do zapisu (w zależności czy w argumencie command podamy MIF_AUTHENTA czy MIF_AUTHENTB).
- uid - musimy podać tablicę długości 4 bajty, w której znajduje się UID karty. Należy go odczytać wywołując wcześniej funkcję rfid_select_tag.

Funkcję tę należy wykonać dopiero po wywołaniu funkcji rfid_select_tag.

```
1 rfid_status_t rfid_authenticate(  
2     uint8_t command,  
3     uint8_t blockAddr,  
4     const uint8_t *key,  
5     const uint8_t *uid  
6 );
```

3.1.21 rfid_card_read

Czyta dane z wcześniej wybranego bloku przez rfid_authenticate (0-15) w wybranym sektorze (0-4). Zapisuje do *buffer.

```
1 rfid_status_t rfid_card_read(  
2     uint8_t sector_address,  
3     uint8_t *buffer,  
4     uint8_t *buffer_size  
5 );
```

3.1.22 rfid_card_write

Wpisuje dane z buffer'a do wcześniej wybranego bloku przez rfid_authenticate (0-15) w wybranym sektorze (0-4).

```
1 rfid_status_t rfid_card_write(
2     uint8_t sector_addr,
3     const uint8_t *buffer,
4     uint8_t buffer_size
5 );
```

3.1.23 rfid_card_transceive

Funkcja pomocnicza, służąca do wydawania komend karcie. Wywołuje pod spodem rfid_transceive_data, lecz karta wymaga aby na końcu przesyłanej tablicy znajdowała się suma kontrolna CRC, więc dopisuje do danych, które chcemy przesłać sumę kontrolną.

```
1 rfid_status_t rfid_card_transceive(
2     uint8_t *send_data,
3     uint8_t send_len,
4     bool accept_timeout
5 );
```

3.1.24 rfid_is_new_card

Sprawdza czy jest jakaś karta w zasięgu nadajnika. Wywołuje pod spodem funkcję auth_a. Jeśli karta jest obecna zwraca true, jeśli nie jest, zwraca false.

```
1 bool rfid_is_new_card();
```

3.1.25 rfid_halt

Wysyła do karty wiadomość, o której przerwania komunikacji. Karta może teraz znowu być rozpoznawana przez odbiornik.

```
1 rfid_status_t rfid_halt();
```

3.1.26 rfid_stop_crypto

Po wywołaniu funkcji rfid_halt, należy wywołać tę funkcję. Czyści ona rejestr i przygotowuje odbiornik do możliwości czytania nowych kart.

```
1 void rfid_stop_crypto();
```

4

SEKCJA

Problemy napotkane podczas tworzenia projektu

4.1 Problemy z konfiguracją projektu w CubeMX

Największym problemem była konfiguracja wyświetlacza LCD mikro-kontrolera STM32f4 w CubeMX. Na wyświetlaczu pokazywał się obraz tylko w 1/4 całej powierzchni ekranu. Po przeczytaniu wielu podobnych wątków na forach internetowych związanych z tą właśnie płytą, okazało się, że problem leży w samym programie CubeMX, który źle konfiguruje wyświetlacz konkretnego mikro-kontrolera.

Należało więc ręcznie zmienić konfigurację wyświetlacza w plikach źródłowych po generacji kodu.

4.2 Problemy z działaniem portu UART na module RFID RC522

Duża część czasu została przeznaczona na próbę nawiązania komunikacji z modułem RFID przy pomocy portu UART, który był wykorzystywany na zajęciach z Techniki Mikroprocesorowej. W dokumentacji modułu, napisane jest, że potrafi on korzystać z UART, lecz próby nawiązania z nim połączenia były bezskuteczne. Ponownie skorzystano z pomocy forów internetowych, gdzie dowiedziano się, że aby port UART działał, należało ręcznie przeciąć połączenie między dwoma stykami na płytce. Zastosowano więc inny port dołączenia z płytą - SPI. Należało również doszkolić się z jego działania i nauczyć się go konfigurować w programie CubeMX.

4.3 Problemy z aktualizacją komponentów na ekranie płytki

4.3.1 Problem z responsywnością ekranu

Aby zapewnić aktualizację danych na ekranie w odpowiednim tempie trzeba było zapewnić odpowiedni flow programu tj. odpowiednie przekazywanie danych z modelu do prezentera i z prezentera do widoku - przy zczytywaniu danych z karty - oraz w drugą stronę - przy zapisywaniu danych na kartę.

Gdy na początku planowane było pominięcie udziału modelu w zapisie danych na kartę, okazało się to problematyczne ponieważ występowały poważne problemy z responsywnością widoku - widok zawieszał się lub przestawał wykonywać nakazane instrukcje.

4.3.2 Problem z aktualizowaniem komponentów odpowiedzialnych za zmieniający się tekst na ekranie

W przypadku komponentów odpowiedzialnych za pokazywanie zmieniającego się tekstu na ekranie, problemy wystąpiły w przypadku wywołań funkcji odpowiedzialnych za aktualizowanie bufora przechowującego aktualny stan napisu.

Sporo czasu zajęło zrozumienie na czym polegał błąd, ponieważ funkcja aktualizująca bufor odpowiedzialny za przechowywany tekst działała poprawnie tylko dla integerów.

Kluczem do rozwiązania problemu była odpowiednia konwersja napisów typu char* na napisy typu UnicodeChar* - specjalnego typu używanego przez funkcję odpowiedzialną za aktualizację bufora (mimo, że żadne ostrzeżenie ani błędy komplikacji nie występowały).

Bibliografia

- <https://touchgfx.zendesk.com/hc/en-us/articles/207015345>
- <https://touchgfx.zendesk.com/hc/en-us/articles/360018667192-Step-1-Setting-up-the-two-Screens>
- <https://touchgfx.zendesk.com/hc/en-us/articles/205443742-Step-2-Adding-Buttons>
- <https://touchgfx.zendesk.com/hc/en-us/articles/205587571-Step-3-Adding-Text>
- <https://touchgfx.zendesk.com/hc/en-us/articles/205443982-Step-4-Adding-code>
- <https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf>
- https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf
- <https://forum.arduino.cc/index.php?topic=442750.0>
- <https://community.st.com/s/question/0D50X0000Ay9hWFSQY/touchgfx-cubemx-cubeide-integration-problems-please-help>