AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa inżynierska

Aplikacja webowa do nauki języka migowego
Web application for sign language learning

Autorzy: *Piotr Ciążyński*
Kierunek studiów: *Inżynieria Biomedyczna*
Opiekun pracy: *dr inż. Jakub Gałka*

Kraków, 2016

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Z podziękowaniami dla

Pana dr inż. Jakuba Gałki za inspirację oraz umożliwienie zrealizowania tej pracy, a także za wiele cennych uwag i pomoc podczas jej realizacji

Marka Ciążyńskiego oraz Pawła Kubiaka za wszelką pomoc w trakcie tworzenia tej pracy

Przemysława Węgrzynowicza za współpracę podczas tworzenia kompletnej wersji aplikacji

oraz dla wszystkich osób, które wzięły udział w testach użyteczności zaprojektowanej aplikacji.

Spis treści

1. Wstęp

W obecnych czasach dostęp do szerokopasmowego Internetu stał się całkowicie powszechny. Bardzo rozwinęły się także technologie webowe, zarówno te po stronie serwera jak i po stronie przeglądarki. Umożliwiło to stworzenie interaktywnych, działających w czasie rzeczywistym stron internetowych. Często wiele z nich do złudzenia przypomina tradycyjne aplikacje komputerowe. Z tego powodu tego typu strony internetowe uzyskały miano *aplikacji webowych*.

Jednocześnie dzięki dużym mocom obliczeniowym współczesnych komputerów bardzo rozwinęły się algorytmy uczenia maszynowego (ang. *machine learning*), dzięki którym możliwe stało się automatyczne wykonywanie zadań, jeszcze niedawno zarezerwowanych wyłącznie dla człowieka. Do zadań takich zaliczyć można m.in. rozpoznawanie tekstu, przetwarzanie mowy czy rozumienie obrazu. Dzięki temu co raz bardziej realne staje się także automatyczne przetwarzanie, tłumaczenie i uczenie języków obcych. Autor pracy ma możliwość uczestniczenia w projekcie WiTKoM, którego zadaniem jest stworzenie komputerowego tłumacza języka migowego [1]. Tak zrodził się pomysł stworzenia aplikacji umożliwiającej naukę takiego języka. Wybrano środowisko webowe, ponieważ z jednej strony możliwe jest wykorzystanie dużych mocy obliczeniowych, które są dostępne na serwerze, a równocześnie aplikacja możliwa jest do uruchomienia na każdym urządzeniu z dostępem do Internetu i kamerki, niezależnie od systemu operacyjnego.

Celem opisywanego projektu inżynierskiego jest, jak już wspomniano, stworzenie interaktywnej aplikacji internetowej, która umożliwiłaby naukę gestów Polskiego Języka Migowego. Aplikacja taka mogłaby pomóc wielu osobom, które potrzebują porozumiewać się z osobami głuchymi. O ile wiadomo, mało jest dostępnych kursów dotyczących konkretnie Polskiego Języka Migowego. Istniejące rozwiązania mają najczęściej formę słownika lub lekcji pod postacią filmów wideo. Z tego względu, projekt stworzenia interaktywnej aplikacji do nauki języka migowego może wypełnić pewną niszę na rynku.

Podczas projektowania aplikacji założono, że użytkownik będzie miał możliwość sprawdzenia, jak dobrze powtarza gest wyświetlony na filmie instruktażowym. W tym celu koniecznie

jest wykorzystanie kamery internetowej aby nagrać użytkownika próbującego wykonać gest. Nagranie takie ma posłużyć do automatycznego sprawdzenia poprawności gestu i zwrócenia tej informacji użytkownikowi.

Ponieważ złożoność takiej aplikacji jest dosyć duża, rozwijana jest ona w ramach dwóch prac inżynierskich. Celem tej pracy inżynierskiej jest stworzenie interfejsu pod postacią interaktywnej strony internetowej oraz programu działającego na serwerze, którego zadaniem będzie pośredniczenie pomiędzy przeglądarką, a zewnętrznym serwerem rozpoznającym gesty. Zewnętrzny serwer rozpoznający gesty jest częścią innego projektu inżynierskiego [2].

Ważnym założeniem aplikacji rozwijanej w ramach tej pracy inżynierskiej jest jej działanie w nowoczesnych przeglądarkach internetowych bez konieczności użycia wtyczek, oraz wykorzystanie otwartego oprogramowania (ang. *open-source*) przy jej tworzeniu.

2. Polski Język Migowy

Wbrew obiegowej opinii, na świecie istnieje wiele różnych języków migowych, których zasięg występowania często różni się od zasięgu występowania języków mówionych. Przykładowo, w Stanach Zjednoczonych oraz w Wielkiej Brytanii używa się różnych języków migowych, mimo, że używa się tego samego języka fonicznego.

Nawet w Polsce istnieją różne sposoby komunikacji migowej. Należą do nich System Językowo-Migowy (SJM) oraz Polski Język Migowy (PJM). Pierwszy z nich jest sztucznym systemem, który polega na przedstawianiu polszczyzny mówionej za pomocą gestów. Jego gramatyka wzorowana jest ściśle na języku polskim – każdemu słowu i końcówce fleksyjnej ze zdania polskiego przypisywany jest znak migany. W przeciwieństwie do SJM, Polski Język Migowy jest językiem naturalnym, który podobnie jak wiele języków fonicznych, wykształcił się spontanicznie z potrzeb komunikacyjnych danej społeczności. PJM posiada własne słownictwo i gramatykę, odmienną od fonicznego języka polskiego. Z tego względu jest on dla słyszących Polaków językiem obcym [3]. Warto podkreślić, że Głusi[1] wolą komunikować się właśnie za pomocą Polskiego Języka Migowego. Z kolei SJM jest odbierany przez te osoby jako sztywny, mniej wygodny w użyciu i sprawia wrażenie języka sztucznego.

To prawda, że z osobami niesłyszącymi można porozumieć się przy pomocy standardowego języka polskiego z użyciem pisma. Trzeba jednak pamiętać, że tak jak wspomniano, wygodniejszym i bardziej naturalnym dla osób Głuchych, sposobem komunikacji będzie właśnie PJM. Dodatkową motywacją do nauki tego języka może być informacja, że jest to język o charakterze przestrzenno–wizualnym. Posiada on cechy niespotykane w językach fonicznych, takie jak nielinearność wypowiedzi [4]. Duża odmienność języków migowych może stanowić pewną trudność podczas ich nauki, ale z drugiej strony nauczenie się języka o tak odmiennej gramatyce jest bardzo ciekawe oraz rozwijające.

Należy zaznaczyć, że aplikacja rozwijana w ramach tego projektu, na chwilę obecną nie umożliwia nauki gramatyki. Do wersji testowej aplikacji wybrano jedenaście gestów.

[1] Słowo „Głuchy” pisane dużą literą odnosi się do członkostwa w społeczności „Głuchych”. Osoby te nie chcą być postrzegane jako niepełnosprawne, tylko jako członkowie mniejszości językowo–kulturowej.

Wybrano cztery zwroty grzecznościowe:

1. Cześć
2. Dobranoc
3. Dziękuję
4. Przykro mi

oraz siedem zwrotów związanych z medycyną:

1. Ciśnienie krwi
2. Karetka
3. Lekarstwo
4. Oddychać
5. Operacja
6. Ostry dyżur

Wybór ten podyktowany jest różnorodnością ich wykonywania, co ułatwia przetwarzanie przez serwer rozpoznający gesty [2]. Natomiast słowa medyczne wybrane zostały z przeświadczeniem, że ich znajomość może być kluczowa w sytuacjach nagłego zagrożenia życia lub zdrowia. W przyszłości planowane jest zwiększenie zasobu słownictwa dostępnego w aplikacji.

3. Wykorzystane biblioteki i technologie

W tym rozdziale przedstawione zostaną biblioteki oraz platformy programistyczne (ang. *framework*) wykorzystane do zbudowania opisywanej aplikacji webowej.

3.1. Node.js i Express.js

Node.js jest środowiskiem, które pozwala uruchomić kod napisany w języku JavaScript poza przeglądarką. Dzięki temu możliwe jest napisanie serwerowej części aplikacji internetowej w języku JavaScript. Node.js zbudowany jest w oparciu o silnik JavaScript V8, wykorzystywany także w przeglądarce Google Chrome. Jedną z najważniejszych cech tego środowiska jest wykorzystanie nieblokującego, asynchronicznego systemu wejścia/wyjścia [5]. Rozwiązanie takie daje możliwość daleko idącej skalowalności aplikacji. Przykładowo, odczytywanie dużego pliku z dysku nie zablokuje działania aplikacji serwerowej, tzn. będzie ona nadal reagowała na bieżąco na inne zdarzenia, takie jak zapytania od przeglądarki. Asynchroniczność wymaga jednak zastosowania innego stylu programowania zwanego *continuation-passing style*, w którym każda funkcja przyjmuje dodatkowy argument, nazywany wywołaniem zwrotnym (ang. *callback*). Innymi słowy, gdy funkcja nadrzędna zakończy swoje działanie, to sterowanie przekazywane jest do funkcji *callback*, a nie do następnej funkcji znajdującej się w kodzie [6].

Platforma Node.js jest zaawansowanym środowiskiem programistycznym, a nie tylko serwerem HTTP. Z tego powodu, funkcjonalność taka jak reagowanie na zapytania przeglądarki musi zostać zaprogramowana jawnie. Express.js jest jednym z frameworków napisanych dla tego środowiska, który ułatwia odpowiednie zaprogramowanie serwera HTTP, reagowanie na zapytania od przeglądarki oraz pisanie bardziej złożonych aplikacji w środowisku Node.js [7].

3.2. Socket.IO

Socket.IO jest biblioteką napisaną w języku JavaScript, która umożliwia dwukierunkową komunikację w czasie rzeczywistym pomiędzy przeglądarką i serwerem bez dużych nakładów

programistycznych. Biblioteka ta składa się z części działającej w przeglądarce oraz części działającej na serwerze w środowisku Node.js. Podobnie jak środowisko Node.js, funkcje tej biblioteki wywoływane są zdarzeniami (ang. *event-driven architecture*) [8].

3.3. AngularJS

AngularJS jest platformą programistyczną napisaną w języku JavaScript stworzoną i rozwijaną przez Google, służącą do budowania aplikacji webowych po stronie klienta na pojedynczej stronie. Dzięki takiemu podejściu aplikacja internetowa sprawia wrażenie spójnej całości, która szybko reaguje na polecenia użytkownika, w przeciwieństwie do tradycyjnych stron internetowych składających się z wielu osobnych podstron. Dodatkową zaletą tej platformy jest zastosowanie deklaratywnego paradygmatu programowania [9]. W takim podejściu, w przeciwieństwie do programów napisanych w paradygmacie imperatywnym, opisuje się to co chcemy osiągnąć, a nie szczegółową sekwencje kroków, które do niego prowadzą [10]. AngularJS umożliwia także rozszerzenie języka HTML o nowe dyrektywy i atrybuty, czym wyróżnia się od innych platform tego typu [11].

3.4. Twitter Bootstrap

Bootstrap jest zestawem stylów CSS i skryptów JavaScript rozwijanym przez zespół programistów Twittera. Framework ten wykorzystywany jest do szybkiego zbudowania układu (*layoutu*) strony internetowej oraz zapewnia wiele gotowych komponentów, takich jak przyciski, listy i inne elementy interfejsu. Głównym założeniem biblioteki Bootstrap jest zastosowanie systemu wierszy (rzędów) i kolumn (ang. *grid system*). System ten dzieli stronę internetową na dwanaście kolumn oraz dowolną liczbę rzędów. Elementy strony rozmieszcza się poprzez ustawienie ich w odpowiednim rzędzie oraz ustalenie jak wiele kolumn mają zajmować. Dzięki temu stworzenie układu graficznego strony jest łatwe i nie wymaga ręcznego pisania styli CSS [12].

3.5. WebRTC

WebRTC jest standardem rozwijanym w ramach standardu HTML5, którego głównym celem jest umożliwienie komunikacji z wykorzystaniem dźwięku i obrazu w czasie rzeczywistym, z wykorzystaniem przeglądarek internetowych [13]. W tym celu standard ten udostępnia funkcje JavaScript (wbudowane w przeglądarki) umożliwiające:

– dostęp do kamery

– przesyłanie obrazu i dźwięku między przeglądarkami w czasie rzeczywistym

– przesyłanie dowolnych danych bezpośrednio pomiędzy dwoma przeglądarkami na zasadzie *P2P* (ang. *peer–to–peer*)

– nagrywanie obrazu i dźwięku z kamery do pliku, bezpośrednio po stronie przeglądarki

Wszystkie funkcje realizowane są bez użycia dodatkowych wtyczek takich jak Adobe Flash. Standard WebRTC obsługiwany jest obecnie przez wszystkie nowoczesne przeglądarki internetowe, jednakże różni się stopień zaimplementowania poszczególnych funkcji.

3.6. Serwer przetwarzający gesty

Do przetwarzania obrazu i rozpoznawania gestów wykorzystywany jest serwer opracowany w ramach innej pracy dyplomowej [2], wykorzystujący technologie opracowane w ramach projektu „Wirtualny Tłumacz Komunikacji Migowej" [1]. Serwer ten działa w środowisku Matlab.

Do wykrywania obiektów w obrazie (ekstrakcji cech), wykorzystano metodę nazywaną „przepływem optycznym" (ang. *optical flow*), należącą do metod wizyjnych. W rozpoznawaniu języka migowego wykorzystuje się między innymi technologie opracowane do rozpoznawania mowy. Na etapie rozwoju, zanim serwer będzie mógł być wykorzystany do rozpoznawania gestów, przeprowadzane jest trenowanie modelu języka migowego. Innymi słowy, cechy uzyskane z nagrań prawidłowo wykonanych gestów zostają użyte jako wejście dla algorytmów uczenia maszynowego. Wykorzystano podejście znane jako Ukryte Modele Markowa. Po wytrenowaniu takich modeli, serwer może być wykorzystany do rozpoznawania gestów i uzyskania procentowego wyniku poprawności ich wykonywania.

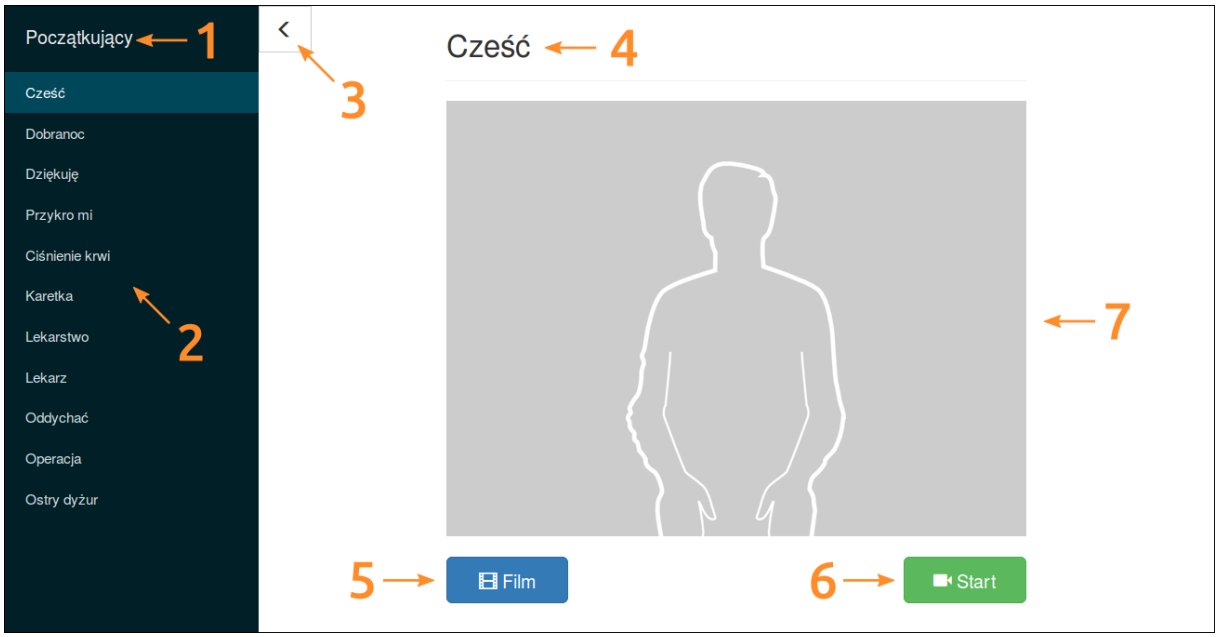
W aplikacji tej, wykorzystany serwer korzysta z modeli, które na obecnym etapie wymagają dokładnego ustawienia się przed kamerą, a uzyskane wyniki są zależne od osoby wykonującej gest (ang. *user dependent model*). Niestety ze względu na duże trudności z rozpoznawaniem obrazu i nielinearnością języków migowych, a także ze względu na mniejsze zapotrzebowanie, technologie tego typu są znacznie słabiej rozwinięte w stosunku do technologii przetwarzania mowy [14].

4. Interfejs użytkownika

Rozdział ten poświęcony został układowi graficznemu oraz interfejsowi projektowanej aplikacji. Przy projektowaniu układu graficznego głównym założeniem była jego prostota, tj. ograniczenie zbędnych elementów i zastosowanie bardzo popularnych obecnie płaskich form graficznych (ang. *flat design*).

4.1. Elementy interfejsu użytkownika

Układ graficzny projektowanej aplikacji internetowej składa się z dwóch zasadniczych części: po lewej znajduje się tzw. panel boczny (ang. *sidebar*), natomiast po prawej znajduje się zasadnicza część interfejsu. Na rys. 4.1 główne elementy układu graficznego oznaczono strzałkami.



Rysunek 4.1: Zrzut ekranu z wyróżnionymi elementami interfejsu aplikacji

Panel boczny składa się z nagłówka (nr **1**) oraz listy z gestami (nr **2**), których można się

nauczyć. Obecnie dostępna jest tylko jedna lista gestów, dlatego nagłówek (nr **1**) spełnia w tej chwili rolę wyłącznie informacyjną i jest elementem statycznym, którego nie można kliknąć. Lista gestów (nr **2**) umożliwia wybranie słowa, którego w danym momencie chcemy się nauczyć. Wybrany gest podświetlany jest na liście (nr **1**) oraz wyświetlany w nagłówku (nr **4**) znajdującym się na głównej części strony. Przycisk (nr **3**) umożliwia schowanie panelu bocznego na żądanie użytkownika, a strzałka znajdująca się na tym przycisku przyjmuje zwrot przeciwny, tak żeby funkcja przycisku pozostawała jasna dla użytkownika. Ponadto, jeżeli szerokość przeglądarki będzie mniejsza od 768 pikseli to panel boczny zostanie automatycznie schowany.

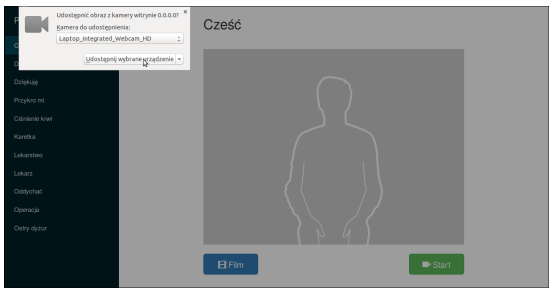
Główna część interfejsu użytkownika składa się ze wspomnianego wcześniej nagłówka (nr **4**) wyświetlającego wybrany gest, przycisków służących do sterowania aplikacją (nr **5** i **6**) oraz centralnego elementu wideo (nr **7**). Przycisk (nr **5**) służy do przełączania pomiędzy wyświetlaniem filmu pokazującego właściwe wykonanie gestu, a widokiem z kamery użytkownika. Jeżeli aplikacja znajduje się w trybie wyświetlania filmu instruktażowego, to prawy przycisk (nr **6**), służy do ponownego odtworzenia tego filmu. Gdy wyświetlany jest obraz z kamery, to wtedy prawy przycisk umożliwia rozpoczęcie lub zatrzymanie nagrywania użytkownika.

Jeżeli opisywana strona internetowa zostanie załadowana w nieobsługiwanej przeglądarce lub pojawi się problem dotyczący kamery, to główna część interfejsu zostanie ukryta, a wyświetlona zostanie jedynie ramka informująca o błędzie. Przykładowy zrzut ekranu z wyświetlonym opisem błędu przedstawiono na rys. 4.2.

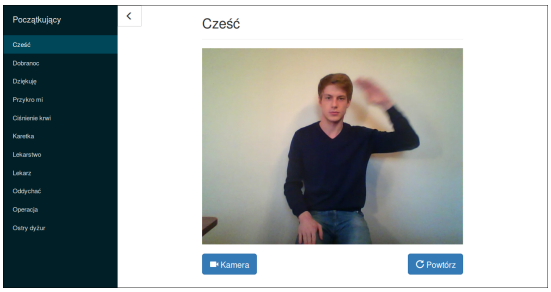
Rysunek 4.2: Zrzut ekranu aplikacji w stanie błędu

4.2. Przebieg standardowego użycia aplikacji

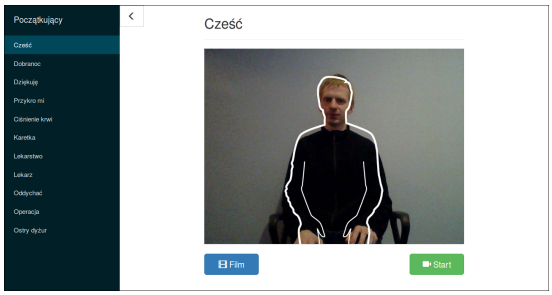
Kolejne etapy typowego użycia zaprojektowanej aplikacji przedstawiono na rys. 4.3.



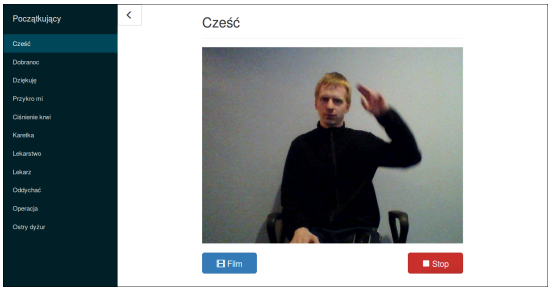
(a) Zezwolenie na dostęp do kamery



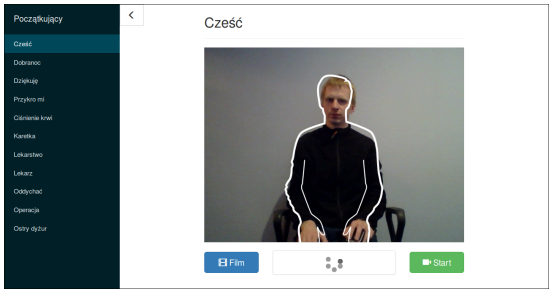
(b) Wyświetlenie filmu instruktażowego



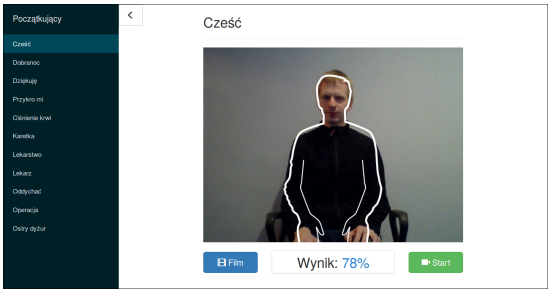
(c) Ustawienie użytkownika przed kamerą



(d) Nagrywanie gestu



(e) Przetwarzanie przez serwer Witkom



(f) Wyświetlenie wyniku

Rysunek 4.3: Typowy sposób użycia aplikacji przedstawiony na zrzutach ekranu

Po załadowaniu aplikacji, przeglądarka wyświetla monit o udostępnienie obrazu z kamery (rys. 4.3a) (gdyby użytkownik odmówił udostępnienia kamery to wyświetlony zostałby błąd podobny do przedstawionego na rys. 4.2). Następnie użytkownik wybiera z listy interesujący go gest i klika na przycisk FILM, aby dowiedzieć się jak poprawnie wykonać gest (rys. 4.3b). W tym momencie prawy przycisk POWTÓRZ umożliwia ponowne obejrzenie filmu przykładowego, a lewy przycisk KAMERA powrót do wyświetlania obrazu z kamery.

Po przełączeniu do widoku z kamery (rys. 4.3c), prawy przycisk START umożliwia rozpoczęcie nagrywania. Ponieważ obecna wersja serwera Witkom do poprawnego rozpoznawania gestów wymaga odpowiedniego ustawienia nagrywanej osoby, na obraz z kamery nakładana

jest schematyczna sylwetka mająca pomóc użytkownikowi, aby odpowiednio ustawił się on przed kamerą. Przyjęto założenie, że uczenie się gestów będzie bardziej intuicyjne, jeżeli film instruktażowy oraz obraz z kamery będzie wyświetlany w odbiciu lustrzanym. Dzięki temu użytkownik powtarza gest z filmu tak jakby uczył się przed lustrem.

Po odpowiednim ustawieniu, użytkownik rozpoczyna wykonanie gestu po naciśnięciu na przycisk START. W tym momencie (rys. 4.3d), rysunek schematycznej sylwetki znika, a na prawym przycisku pojawia się napis STOP.

Po zakończeniu nagrywania (rys. 4.3e), film wysyłany jest na serwer i przetwarzany. W tym momencie między przyciskami pojawia się animacja informująca o przetwarzaniu (tzw. *spinner*).

Następnie, wynik otrzymany z serwera zostaje wyświetlony między przyciskami za pomocą punktów procentowych (rys. 4.3f). Teraz użytkownik może powtórzyć wykonanie tego samego gestu, lub wybrać z listy inny i powtórzyć cały proces.

4.3. Obsługa skrótów klawiaturowych

W dzisiejszych czasach przy projektowaniu różnego rodzaju interfejsów użytkownika zawsze pamięta się o użytkownikach początkujących i interfejsach dotykowych. Zapomina się jednak często, o użytkownikach, którzy potencjalnie mogą z naszej aplikacji korzystać bardzo często i wtedy wygodniejsze może być dla nich użycie klawiatury.

Skróty klawiszowe zaimplementowane w tej aplikacji zaprezentowano w tab. 4.1. Do poruszania się po liście gestów, znajdującej się na panelu bocznym służą klawisze strzałek (góra i dół). Przełączanie widoku pomiędzy filmem instruktażowym, a obrazem z kamery obsługiwane jest za pomocą przycisku F. W trybie filmu instruktażowego przycisk R umożliwia jego ponowne wyświetlenie, natomiast w trybie obrazu z kamery, klawisz spacji umożliwia rozpoczęcie oraz zatrzymanie nagrywania.

Tabela 4.1: Lista skrótów klawiaturowych

Czynność	**Skrót klawiaturowy**
Kolejny gest z listy	strzałka w dół
Poprzedni gest z listy	strzałka w górę
Przełączanie widoku kamera/film	litera F
Powtórzenie filmu instruktażowego	litera R
Start/Stop	spacja

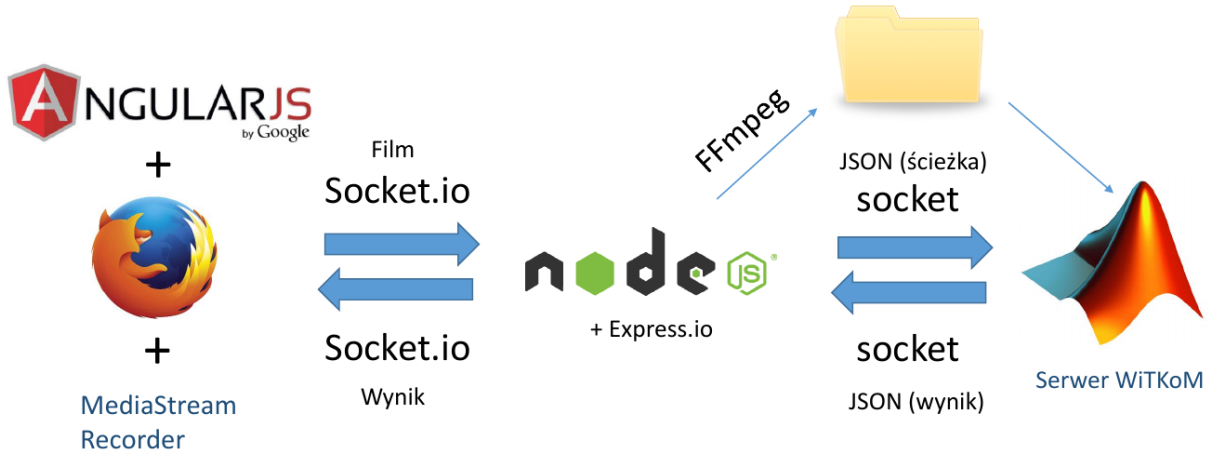
5. Architektura i działanie systemu

W tym rozdziale opisane zostaną techniczne szczegóły działania aplikacji. Na rys. 5.1 przedstawiono ogólny schemat architektury opisywanej aplikacji. Cała aplikacja składa się z trzech zasadniczych części:

1. Interfejsu użytkownika zrealizowanego za pomocą dynamicznej strony internetowej działającej w przeglądarce

2. Serwera Node.js, pełniącego rolę pośrednika pomiędzy interfejsem, a serwerem rozpoznającym gesty

3. Serwera rozpoznającego gesty (działającego na platformie Matlab)



Rysunek 5.1: Schemat architektury systemu

W ramach tej pracy inżynierskiej rozwijane są pierwsze dwie części systemu. Szczegółowe rozwiązania przedstawiono w kolejnych podrozdziałach. Kod źródłowy aplikacji znajduje się w załączniku nr 1.

5.1. Opis aplikacji po stronie przeglądarki

Opisywana strona internetowa napisana jest z wykorzystaniem języków HTML5, CSS3 oraz JavaScript. Układ graficzny strony zrealizowano za pomocą frameworku Bootstrap, natomiast jej funkcjonalność za pomocą biblioteki programistycznej AngularJS (patrz rozdz. 3.3). Strona internetowa do działania nie potrzebuje wtyczek zewnętrznych. Na chwilę obecną aplikacja działa poprawnie jedynie w najnowszych wersjach przeglądarki Mozilla Firefox.

5.1.1. Szczegóły techniczne interfejsu

Do skonstruowania strony internetowej posiadającej panel boczny wykorzystano szkielet napisany przez Davida Millera [15]. Szkielet ten grupuje wszystkie elementy strony w dwóch częściach – do wspomnianego panelu bocznego oraz do części zawierającej główne elementy interfejsu. Co ważne, w szkielecie zaimplementowana została funkcja automatycznego ukrywania panelu, gdy szerokość przeglądarki jest mniejsza od 768 pikseli. Funkcjonalność ta zrealizowana jest za pomocą funkcji *media-query* należącej do języka CSS3. Aby użytkownik miał możliwość schowania panelu również niezależnie od szerokości przeglądarki, dodany został odpowiedni przycisk. Zarówno w przypadku realizowanym za pomocą funkcji *media-query* jak i po naciśnięciu przycisku, ukrycie panelu następuje poprzez ustawienie lewego marginesu wewnętrznego (ang. *padding-left*) całej strony (dokładnie elementu `div` otaczającego wszystkie elementy strony) tak, żeby panel boczny został przesunięty poza obszar przeglądarki.

Dla lepszego efektu wizualnego domyślne kolory panelu (skala szarości) zastąpione zostały kolorami pochodzącymi z kompozycji *solarized dark* będącej częścią edytora Atom. Kolejnym etapem rozwoju było dostosowanie arkuszy stylów opisujących panel boczny w taki sposób, żeby możliwe stało się podświetlanie wybranego gestu na liście. Podświetlany jest również element listy nad którym aktualnie znajduje się kursor myszy.

Elementy głównej części interfejsu (nagłówek, element wideo, przyciski) rozmieszczone zostały za pomocą systemu rzędów i kolumn biblioteki Bootstrap. W najwyższym rzędzie umieszczono przycisk do ukrywania panelu oraz nagłówek zawierający nazwę aktualnie wybranego słowa. W drugim rzędzie umieszczony jest główny element, w którym wyświetlany jest obraz z kamery lub film. Najniższy rząd zawiera przyciski do sterowania aplikacją oraz miejsce na wyświetlenie wyniku.

Ikony znajdujące się na przyciskach dostarczone zostały przez Bootstrap i pochodzą z projektu Glyphicon [16]. Ponadto, przed wyświetleniem wyniku, wyświetlana jest animacja zaczerpnięta z projektu CSS Spinners [17]. Animacja ta, napisana jest w całości za pomocą CSS3 i renderowana jest bezpośrednio przez przeglądarkę.

Ważną częścią interfejsu są także skróty klawiszowe opisane w rozdz. 4.3. Do ich obsługi zastosowano moduł angular-keyboard, który umożliwia bardzo łatwe powiązanie konkretnych elementów języka HTML z funkcjami AngularJS [18].

5.1.2. Obsługa stanu aplikacji

Jak wspomniano wyżej, stan aplikacji i jej funkcjonalność realizowana jest przez bibliotekę AngularJS. Dzięki temu cała aplikacja jest zbudowana na pojedynczej stronie. Innymi słowy, zmiana wybranego gestu czy naciśnięcie jednego z przycisków nie spowoduje załadowania innej strony HTML, ani przeładowania bieżącej, zmienia się tylko stan aplikacji napisanej z użyciem biblioteki AngularJS.

Dużą zaletą tej biblioteki jest także możliwość łatwego ukrywania elementów HTML oraz ustawiania klas CSS w zależności od wartości zmiennych w aplikacji. Do obsługi różnych stanów aplikacji wykorzystano następujące zmienne:

- `toggled` (*wart. logiczna*) – zmienna przechowująca stan panelu bocznego, jej zmiana powoduje ukrycie lub pokazanie panelu bocznego, tak jak zostało opisane to w poprzednim podrozdziale
- `selectedGesture` (*wart. tekstowa*) – zmienna zawierająca nazwę aktualnie wybranego gestu, dzięki czemu możliwe jest podświetlenie odpowiedniego elementu z listy, wyświetlenie go w nagłówku oraz wysłanie nazwy wybranego gestu na serwer
- `source` (*wart. tekstowa*) – zmienna przełączająca obraz wyświetlany w elemencie wideo, może to być widok z kamery lub film pobrany z serwera, prezentujący poprawne wykonanie gestu
- `state` (*wart. tekstowa*) – przechowuje informację o tym czy obraz z kamery jest w danej chwili nagrywany
- `showOutline` (*wart. logiczna*) – odpowiada za wyświetlenie obrysu sylwetki, sylwetka wyświetlana jest tylko gdy pokazywany jest obraz z kamery, gdy użytkownik włączy nagrywanie, obrys sylwetki przestaje być widoczny
- `loading` (*wart. logiczna*) – zmienna odpowiadająca za wyświetlenie animacji podczas przetwarzania gestu przez serwer, przyjmuje wartość prawda, tylko po naciśnięciu przycisku STOP, zanim serwer zwróci wynik
- `ready` (*wart. logiczna*) – zmienna odpowiadająca za wyświetlenie ramki z wynikiem, po zwróceniu go przez serwer

5.1.3. Nagrywanie obrazu za pomocą WebRTC

Aby możliwe było skorzystanie z aplikacji, do urządzenia na którym wyświetlona zostanie opisywana strona internetowa musi być podłączona kamera, skonfigurowana tak, by była widoczna w systemie operacyjnym. Dostęp do kamery w przeglądarce realizowany jest za pomocą funkcji `getUserMedia()`, będącej częścią projektu WebRTC. Zgodnie z zaleceniami stylu pisania aplikacji w AngularJS, nie powinno używać się funkcji JavaScript spoza biblioteki AngularJS bez wykorzystania mechanizmu „wstrzykiwań zależności" (ang. *dependency injection*). Aby być w zgodzie z zaleceniami wykorzystano rozwiązanie opisane przez Minko Gecheva [19].

Do nagrywania obrazu z kamery wykorzystano mechanizm `MediaRecorder`, będący częścią standardu WebRTC, ale obsługiwanego jedynie przez przeglądarki Mozilla Firefox od wersji 25 oraz Google Chrome od wersji 47. Niestety z powodu różnic implementacyjnych aplikacja działa obecnie jedynie w standardowej przeglądarce Firefox. Mechanizm `MediaRecorder` jest stosunkowo prosty w użyciu, ponieważ udostępnia funkcję `start()` rozpoczynającą nagrywanie oraz funkcję `stop()` zatrzymującą nagrywanie. Wspomniane funkcje wywoływane są przez przycisk START/STOP (nr 6 na rys. 4.1). `MediaRecorder` udostępnia również mechanizm zdarzeniowy (ang. *event*) o nazwie `ondataavailable`, który umożliwia wywoływanie funkcji po zakończeniu nagrywania i wykorzystanie nagranych danych.

5.1.4. Komunikacja z serwerem za pomocą biblioteki Socket.IO

Ponieważ czas przetwarzania danego filmu przez serwer nie jest po stronie przeglądarki znany, stwierdzono, że najlepszym sposobem komunikacji przeglądarki z serwerem będzie użycie biblioteki Socket.IO. Dzięki niej możliwa jest, bazująca na zdarzeniach, komunikacja w czasie rzeczywistym. Podobnie jak w przypadku opisanym w poprzednim podrozdziale, aby poprawnie użyć funkcji Socket.IO w środowisku AngularJS, wykorzystano odpowiedni moduł, napisany przez Briana Forda [20].

Po naciśnięciu przycisku Stop, gdy wywoływane jest zdarzenie `ondataavailable`, dostępne dane przesyłane są na serwer za pomocą funkcji `emit()`, należącej do Socket.IO. Biblioteka Socket.IO udostępnia funkcję o nazwie `on`, która umożliwia zareagowanie na odpowiednie zdarzenie przychodzące, zainicjowane przez serwer. Dzięki temu, gdy serwer wykona obliczenia i zwróci wynik, otrzymane dane mogą zostać w odpowiednim momencie wyświetlone w przeglądarce.

5.2. Opis aplikacji po stronie serwera

Aplikacja po stronie serwera obsługiwana jest przez platformę Node.js. Jak już zostało wspomniane, serwer Node.js pełni rolę pośrednika pomiędzy interfejsem użytkownika, czyli stroną internetową załadowaną w przeglądarce, a serwerem rozpoznającym gesty działającym na platformie Matlab. Do głównych zadań opisywanego tutaj serwera Node.js można zaliczyć:

1. Uruchomienie środowiska Matlab
2. Odbieranie danych od przeglądarki
3. Konwersja filmu
4. Komunikacja ze środowiskiem Matlab
5. Wysyłanie wyniku do przeglądarki

Zgodnie z tym co napisano w podrozdziale 3.1, Node.js jest środowiskiem asynchronicznym. Dzięki temu poszczególne zadania serwera mogą wykonywać się niezależnie, bez blokowania siebie nawzajem. Są jednak takie zadania, których kolejność musi być ściśle określona. Określenie kolejności wybranych zadań można uzyskać poprzez zagnieżdżenie wywołań zwrotnych (ang. *callback nesting)*.

Aby aplikacja webowa mogła działać, serwer napisany w Node.js musi połączyć się z serwerem rozpoznającym gesty na platformie Matlab. W tym celu pierwszą czynnością jaką wykonuje serwer Node.js jest uruchomienie środowiska Matlab (poprzez utworzenie procesu potomnego), a w nim skryptu o nazwie `runMatlabServer.m`[1].

Skrypt napisany w Matlabie tworzy obiekt `tcpip`, w którym ustawione jest, że Matlab może komunikować się za pomocą gniazd sieciowych TCP/IP (ang. *TCP/IP Sockets*) na porcie 8000 spełniając rolę serwera. Po utworzeniu takiego obiektu, skrypt Matlaba informuje serwer Node.js, że jest gotowy, używając tekstu wypisanego na tzw. standardowe wyjście (ang. *stdout*) oraz czeka na połączenie, które może zainicjować klient (w tym wypadku jest to aplikacja Node.js).

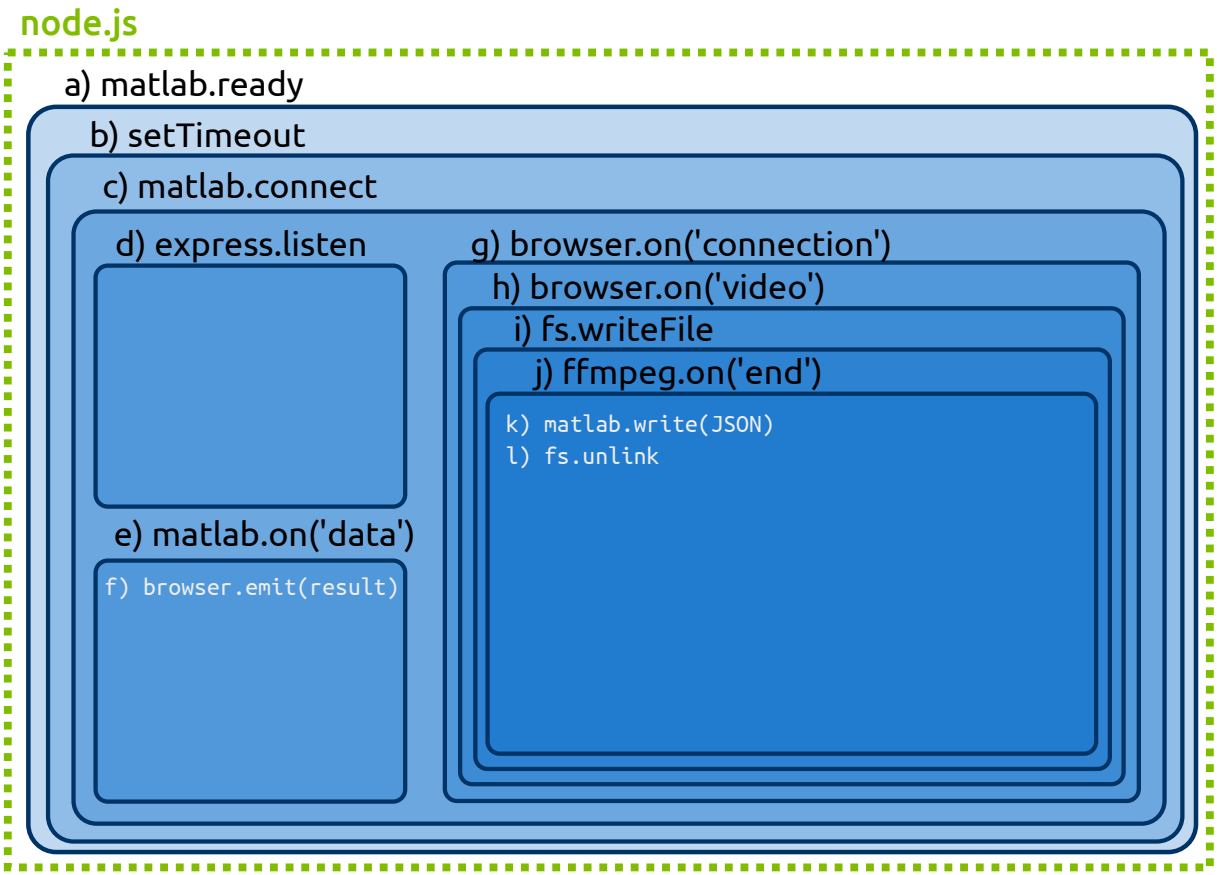
Gdy na standardowym wyjściu pojawi się informacja o tym, że środowisko Matlab jest gotowe (rys. 5.2a), aplikacja Node.js jest wstrzymywana[2] przez okres 100 ms (rys. 5.2b), a następnie nawiązywane jest połączenie z serwerem Matlab za pomocą wbudowanego modułu

[1] Na chwilę obecną Matlab uruchamiany jest automatycznie tylko w systemie operacyjnym Linux. W systemie Windows, należy najpierw ręcznie uruchomić środowisko Matlab ze skryptem `runMatlabServer.m`, a następnie uruchomić serwer Node.js.

[2] Ze względu na pewne problemy z użyciem funkcji `fopen()` w Matlabie.

`net` (rys. 5.2c). Są to tylko zadania przygotowawcze wywoływane tylko raz po uruchomieniu serwera Node.js.

Rys. 5.2 przedstawia zależności pomiędzy wywołaniami zwrotnymi w serwerze Node.js, opisanymi za pomocą uproszczonych nazw (pseudokodu).



Rysunek 5.2: Schemat przedstawiający działanie serwera za pomocą pseudokodu. Niebieskie prostokąty oznaczają wywołania zwrotne (ang. *callbacks*)

W tym momencie serwer Node.js wykonuje trzy główne zadania:

– czeka na połączenia na porcie 3000 i udostępnia stronę internetową na każde żądanie przeglądarek (za pomocą funkcji `listen` frameworku Express.js) (rys. 5.2d)

– łączy się z każdą kopią aplikacji webowej uruchomionej w przeglądarkach za pomocą biblioteki Socket.IO i czeka na dane przychodzące z przeglądarek (rys. 5.2g)

– oczekuje na wynik z serwera przetwarzającego gesty (rys. 5.2e) i przekazuje go do właściwej kopii uruchomionej aplikacji webowej (rys. 5.2f)

Każda uruchomiona kopia aplikacji webowej (strona internetowa otwarta na osobnej karcie, na innym komputerze, w innej sieci) dostaje osobny identyfikator przyznawany automatycznie

przez bibliotekę Socket.IO, dzięki czemu możliwe jest zwrócenie wyniku do właściwej strony. Dane przychodzące z aplikacji są przekazywane w formacie JSON posiadającym pole o nazwie `video` zawierające nagrany gest w formacie webm oraz pole `nameOfGesture` zawierające nazwę wykonanego gestu.

Za każdym razem, gdy dane przychodzą z przeglądarki, uruchamiane są wywołania zwrotne (rys. 5.2h) umożliwiające

1. zapisanie filmu po stronie serwera z nazwą zawierającą identyfikator strony internetowej (rys. 5.2i)

2. przekonwertowanie filmu z formatu *webm* do formatu *avi* (kodek *mpeg4*) (rys. 5.2j), aby możliwe było przetworzenie filmu przez serwer Matlab oraz usunięcie pliku *webm* (rys. 5.2 l)

3. wysłanie informacji w formacie JSON do serwera Matlab, zawierającej pole `fileName` z nazwą pliku *avi* oraz pole `gesture` zawierającej nazwę wykonanego gestu

Zgodnie z tym co napisano powyżej, po przetworzeniu gestu przez serwer zewnętrzny, wynik zwracany jest do serwera Node.js, również w formacie JSON, a następnie przekazywany do odpowiedniej kopii strony internetowej, na podstawie właściwego identyfikatora.

6. Testy interfejsu użytkownika

Testy interfejsów z użytkownikami (ang. *user experience*) są najważniejszą techniką badania i ulepszania użyteczności rozwijanej aplikacji czy strony internetowej [21]. Testy takie, zwane testami UX przeprowadzane są w celu zdobycia informacji o tym jak użytkownicy korzystają z badanego interfejsu [22]. Dzięki temu możliwe jest wychwycenie słabych punktów i nieprawidłowości danego interfejsu.

Dobrą praktyką jest testowanie produktu na każdym etapie jego rozwoju. Testuje się finalną wersję strony, a wcześniej jej prototypy i wersje testowe. Pozwala to na ograniczenie kosztów i zaoszczędzenie czasu, ponieważ wprowadzanie zmian w końcowym produkcie jest już często bardzo trudne i kosztowne [21, 22].

Aplikacja webowa rozwijana w ramach tej pracy inżynierskiej znajduje się obecnie w fazie prototypowej. W ramach projektu rozwinięto stronę docelową (ang. *landing page*), która została poddana badaniom. Postanowiono przeprowadzić dwa niezależne badania:

1. bardzo szybki „test pięciu sekund", przeprowadzony na 24 osobach
2. moderowany test z udziałem użytkowników podczas korzystania ze strony, przeprowadzony na 4 osobach

6.1. Test pięciu sekund

6.1.1. Metodyka testu

Jednym z najszybszych badań interfejsu jest tzw. „Test 5 sekund". Metoda ta polega na pokazaniu użytkownikowi badanej strony internetowej przez dokładnie pięć sekund i sprawdzenie co użytkownik zapamiętał oraz jakie jest jego pierwsze wrażenie.

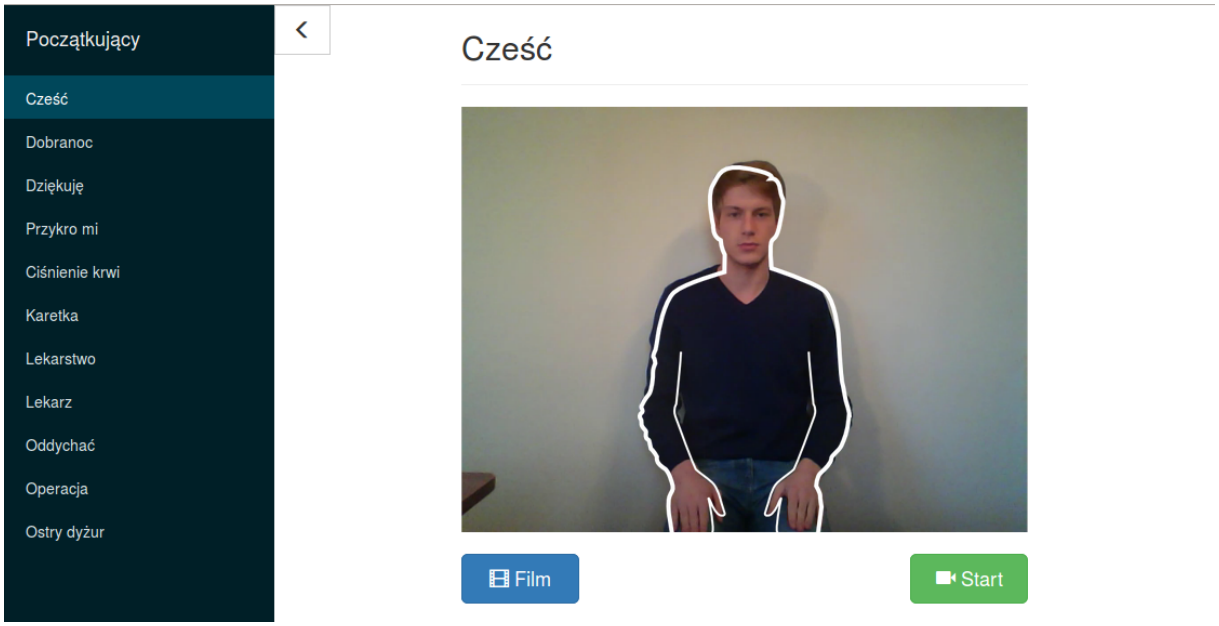
Aby test dostarczył interesujących wyników potrzebne jest przemyślane przeprowadzenie badania. Po pierwsze najważniejsze jest wybranie osób, które nigdy danej strony nie widziały nawet przez chwilę. Ponieważ pięć sekund to bardzo mało potrzebne jest ukierunkowanie uwagi użytkownika. Aby to zrobić, przed pokazaniem strony określa się kontekst sytuacji. Konieczne

jest także uprzedzenie użytkownika, że strona będzie pokazywana tylko przez pięć sekund oraz podkreślenie (ważne przy wszystkich testach UX), że testowana jest strona czy interfejs, a nie respondent [23].

Po ukryciu strony wyświetla się kilka zadań, które musi wykonać użytkownik. Do zadań tych należy wypisanie wszystkich elementów jakie zapamiętał, podanie ogólnego wrażenia czy opisania jaki może być cel przedstawionej strony.

Opisany powyżej test został zaprojektowany i przeprowadzony za pomocą serwisu UsabilityHub [24]. Kontekst sytuacyjny został określony za pomocą zdania: „**Wyobraź sobie, że chcesz nauczyć się języka migowego i natrafiłeś na tę stronę**”.

Następnie zaprezentowany został przez pięć sekund następujący zrzut ekranu z badanej strony internetowej:



Rysunek 6.1: Wygląd interfejsu zaprezentowany użytkownikowi przez 5 sekund

Po ukryciu zrzutu ekranowego aplikacji wyświetlone zostały cztery pytania. Test przeprowadzono zdalnie, na 24 osobach. Każda z tych osób widziała układ graficzny strony po raz pierwszy.

6.1.2. Wyniki

Odpowiedzi na każde z pytań przedstawiono za pomocą chmury najczęściej występujących słów. Im większe słowo, tym częściej pojawiało się w odpowiedziach.

1. Jak myślisz, co umożliwia pokazana strona? Co można za jej pomocą zrobić?

Rysunek 6.2: Chmura najczęściej występujących słów występujących w odpowiedziach na pytanie pierwsze

2. Jakie wrażenie zrobił na Tobie wygląd strony w skali od 1 (bardzo brzydka) do 5 (bardzo ładna).

Odpowiedzi na to pytanie sformułowane były w postaci liczbowych ocen. Średnia ze wszystkich odpowiedzi: **4,15**

3. Co najbardziej zwróciło Twoją uwagę? Jaki element strony zapamiętałeś?

Rysunek 6.3: Chmura najczęściej występujących słów występujących w odpowiedziach na pytanie trzecie

4. Gdzie znajduje się menu strony?

Rysunek 6.4: Chmura najczęściej występujących słów występujących w odpowiedziach na pytanie czwarte

6.2. Test moderowany

Test moderowany jest badaniem użyteczności przeprowadzonym na użytkownikach w obecności moderatora. Zadaniem moderatora jest wytłumaczenie celu przeprowadzanych testów i stworzenie warunków, w których respondenci mogą swobodnie wyrażać swoje myśli. Bardzo ważne jest zaznaczenie, że badany jest interfejs, a nie respondent. Moderator przeprowadza test według scenariusza, zawierającego konkretne zadania, które ma wykonać użytkownik. Najlepiej jest, gdy moderator nie jest twórcą badanego interfejsu czy układu graficznego oraz gdy obserwacją użytkowników i notowaniem tego co się dzieje zajmuje się jeszcze inna osoba – obserwator [25].

6.2.1. Metodyka testu

Jak wspomniano badanym interfejsem jest prototyp aplikacji webowej do nauki języka migowego. Na tym etapie na stronie internetowej brakuje informacji dla użytkownika, odnoszących się do tego w jaki sposób należy skorzystać z aplikacji. Powyższy fakt uwzględniono podczas przeprowadzania testów, tłumacząc użytkownikowi do czego służy badana strona internetowa.

Według zaleceń do testów użyteczności najlepiej jest poprosić pięciu respondentów [26]. Ponieważ badana strona internetowa jest dopiero na etapie wczesnego projektu i ma stosunkowo

nie wiele funkcji, to badania postanowiono przeprowadzić z pomocą czterech respondentów, w celu zaoszczędzenia czasu[1].

Wybrano osoby o różnej biegłości posługiwania się komputerem[2]. Każda z wybranych osób widziały testowaną stronę internetową po raz pierwszy i nie były związane z jej projektowaniem. Nikt spośród testujących nie znał języka migowego. Trzy spośród badanych osób były w wieku ok. 50 lat, natomiast jedna znajdowała się w wieku licealnym. Przyjęto założenie, że zróżnicowana biegłość w obsłudze komputera oraz różne wykształcenie pozytywnie wpłyną na różnorodność błędów, które będą mogły zostać wychwycone podczas przeprowadzania testów. Ponieważ osoby młode są zazwyczaj bardziej sprawne w obsłudze interfejsów komputerowych stwierdzono, że lepiej będzie wybrać kilka osób w średnim wieku oraz jednego nastolatka. Jest to zgodne z ewentualną grupą docelową aplikacji – potrzebę nauki języka migowego mogą mieć osoby w szerokim przedziale wiekowym, jeżeli w ich rodzinie znajdzie się osoba głucha lub niedosłysząca.

Przeprowadzony test składał się z następujących etapów:

1. Wypełnienie pre-kwestionariusza, w którym znajdowały się pytania dotyczące respondenta (wiek, wykształcenie, sposób korzystania z internetu i aplikacji webowych), patrz załącznik nr 2

2. Poinformowanie testującego o tym, że

   – wyświetlany wynik jest nieistotny, testowany jest jedynie interfejs, a nie poprawność rozpoznawania gestów

   – narysowany kontur na obrazie z kamery służy po to, żeby ustawić się zgodnie z wymaganiami aplikacji

   – film przykładowy oraz widok z kamery są w odbiciu lustrzanym

3. Załadowanie strony internetowej i zachęcenie testującego do używania strony następującymi zadaniami:

   (a) Spróbuj nauczyć się trzech wybranych przez siebie gestów

   (b) Załóżmy przez chwilę, że nie trzeba odsuwać się od kamery. Spróbuj teraz użyć aplikacji za pomocą klawiatury.

   (c) Spróbuj schować panel boczny

[1] Kolejne badania użyteczności przy pomocy większej ilości osób powinny być przeprowadzane wraz z rozwojem aplikacji i dodawaniem nowych funkcji

[2] Każda z osób podała inną biegłość w używaniu komputera, były to: słaba, średnia, dobra i bardzo dobra

4. Wypełnienie post-kwestionariusza, jednego pobranego z przewodnika firmy Ideacto [21], oraz własnego, patrz załącznik nr 3

Na bieżąco obserwowano testujących i ich działania, zachęcano do głośnego wypowiadania swoich opinii co było na bieżąco notowane.

6.2.2. Wyniki

Dane użytkowników biorących udział w testach, uzyskane za pomocą kwestionariusza zebrano w tabeli poniżej:

Tabela 6.1: Dane osób testujących

Cecha	**Osoba 1**	**Osoba 2**	**Osoba 3**	**Osoba 4**
Wiek	ok. 50 lat	ok. 50 lat	ok. 50 lat	ok. 17 lat
Zawód	operator linii	informatyk (ale nie zajmujący się stronami internetowymi)	nauczyciel chemii	uczeń liceum
Umiejętności komputerowe (samoocena)	Słabe	Bardzo dobre	Średnie	Dobre
Korzystanie z internetu (w godz. na dzień)	1 h	5 h	2 h	7 h
Znajomość języka migowego	brak			
Inne informacje	użytkownik nie korzysta z aplikacji webowych	użytkownik korzysta czasem z kursów e-learningowych	użytkownik zna osobę niesłyszącą	użytkownik uczy się języków obcych za pomocą Internetu

Wyniki obserwacji przeprowadzonych podczas testu moderowanego zebrano w tabeli 6.2.

Tabela 6.2: Wyniki z obserwacji podczas testów użyteczności

Obserwacja	**Osoba 1**	**Osoba 2**	**Osoba 3**	**Osoba 4**
Sprawność korzystania z aplikacji	słaba, dosyć chaotyczne działania	wysoka	niewielkie problemy podczas korzystania	wysoka
Udostępnienie obrazu z kamery	bez problemu		przeoczenie	
Przechodzenie pomiędzy trybem filmu i kamery	błędne użycie przycisku do ukrywania panelu	użytkownik sugeruje zmianę rozmieszczenia przycisków	z pewnymi problemami	łatwe, bez problemu
Nagrywanie gestu	błędne przejście do filmu przykł. podczas nagrywania	trochę zbyt późne zatrzymywanie nagrywania	nie jest jasne, że należy samemu zatrzymać nagrywanie	czasem zbyt późne zatrzymywanie nagrywania
Przycisk do ukrywania panelu	potraktowanie jako przycisku wstecz	użycie prawidłowe	brak użycia	zaskoczenie funkcją przycisku
Poruszanie się po liście klawiaturą	użycie strzałek było dla każdego intuicyjne			
Inne skróty klawiaturowe	niezauważenie podpowiedzi	oczekiwanie listy skrótów pod przyciskiem F1 lub H	niezauważenie podpowiedzi	niezauważenie podpowiedzi
Ocena wygląd	brak uwag	użytkownik docenia prostotę	sugestia innych kolorów	użytkownik wysoko ocenia kolorystykę
Obrys sylwetki	cel jest zrozumiały, użytkownicy ustawiają się prawidłowo			
Odbicie lustrzane	użytkownicy intuicyjnie, nawet bez poinformowania, powtarzają gest prawidłowo			
Animacja oczekiwania	funkcja jest zrozumiała, brak uwag			

6.2.3. Ilościowa ocena ogólnych aspektów strony

W tabeli poniżej, przedstawiono średnie wyniki, będące odpowiedziami na pytania z kwestionariusza Ideacto [21]. Wartość **1** oznacza stwierdzenie „Zdecydowanie się nie zgadzam", natomiast wartość **5** oznacza stwierdzenie „Zdecydowanie się zgadzam".

Tabela 6.3: Średnia z odpowiedzi na pytania z kwestionariusza

Pytanie	**Średnia**
Czy poruszanie się po stronie było intuicyjne?	3,25
Czy strona była dla Ciebie zrozumiała?	4
Uważam, że serwis jest zbyt skomplikowany	1,75
Myślę, że serwis jest prosty w obsłudze	4
Uważam, że funkcje w serwisie są dobrze ze sobą zintegrowane	4,5
Uważam, że w serwisie jest za dużo niekonsekwentnych rozwiązań	1,5
Wydaje mi się, że osoby korzystające z serwisu nie będą musiały uczyć się jego obsługi	4,25
Czułem się bardzo pewnie poruszając się po serwisie	3
Musiałbym spędzić więcej czasu na nauczenie się, w jaki sposób poruszać się po serwisie	2

Wyniki zebrano w celach informacyjnych, trzeba jednak pamiętać, że aby liczbowe wyniki można było uznać za miarodajne, testy użyteczności należałoby przeprowadzić na znacznie większej grupie osób.

6.3. Wnioski

Wykonanie testów użyteczności okazało się bardzo przydatne. Część problemów była znana autorowi strony już wcześniej, ale dzięki badaniom użyteczności ujawniono wiele innych. Dzięki temu możliwa jest zmiana pewnych elementów strony, jeszcze zanim projekt stanie się zbyt złożony. W podrozdziale tym przedstawione zostaną wnioski po przeprowadzeniu opisanych wyżej badań UX.

6.3.1. Ogólny układ graficzny strony

Wnioski na temat układu graficznego strony można wyciągnąć już na podstawie testu pięciu sekund. Badana strona internetowa jest już stroną docelową, uprawnione jest więc założenie,

że użytkownik, który na nią trafi, chce nauczyć się języka migowego. Odpowiedzi na pytanie pierwsze (rys. 6.2) jednoznacznie sugerują, że funkcjonalność zaprojektowanej strony już po pierwszym spojrzeniu jest dla użytkowników jasna. Wygląd strony w teście 5 sek. jest oceniany na **4,15** (w skali od 1 do 5). Także osoby biorące udział w teście moderowanym wygląd uznawały za estetyczny i nieprzeładowany. Jednak ze względu na różne upodobania, osoby testujące wyrażały bardzo odmienne zdanie na temat istniejącej kolorystyki. Sugerować może to, żeby w finalnym serwisie umożliwić wybór kolorystyki strony.

Ze względu na prostotę wyglądu, większość osób nie miało problemu ze znalezieniem i zapamiętaniem (już po pięciu sekundach) gdzie znajduje się lista słów o czym świadczą jednoznaczne odpowiedzi zebrane na rys. 6.4. Z odpowiedzi na pytanie „Co najbardziej zwróciło Twoją uwagę?" (rys. 6.3) wnioskować można, że najbardziej zwraca uwagę obrys sylwetki. Skoro jest to element mocno przyciągający uwagę, jego użycie musi być dobrze wytłumaczone użytkownikowi. Wiele osób zauważało również bez problemu listę z gestami oraz przycisk START (prawdopodobnie dzięki wyróżnieniu kolorem zielonym). Można zatem wnioskować, że użycie tak prostego interfejsu jest uzasadnione.

6.3.2. Sugerowane zmiany na podstawie wyników testu moderowanego

Mimo prostoty układu graficznego, użycie aplikacji webowej nie jest dla każdego użytkownika oczywiste. Konieczne jest zatem stworzenie strony, na której w sposób zrozumiały wytłumaczone będzie jak należy używać aplikacji. Innym z możliwych rozwiązań jest opracowanie interaktywnego samouczka uruchamianego przy pierwszej wizycie. Działanie takiego samouczka mogłoby polegać na wskazaniu odpowiednich elementów interfejsu we właściwej kolejności, wraz z opisem.

Poza opracowaniem odpowiedniej pomocy, jeszcze ważniejsze jest wyeliminowanie błędów funkcjonowania interfejsu. Najważniejsze zmiany jakie należy wprowadzić to:

1. Zablokowanie możliwości korzystania z aplikacji bez udostępnionej kamery i poinformowanie o tym użytkownika.

2. Wprowadzenie określonego czasu nagrywania. Nagrywanie powinno kończyć się samoczynnie (bez konieczności używania przycisku STOP).

3. Uniemożliwienie wyświetlenia filmu przykładowego podczas nagrywania.

4. Przeprojektowanie przycisku ukrywania panelu bocznego tak, żeby nie mylił się z przyciskiem wstecz. Być może funkcjonalność ręcznego ukrywania panelu powinna zostać całkowicie zlikwidowana.

Kolejne zmiany jakie należy wprowadzić odnoszą się do wygody użytkowania:

1. Po naciśnięciu przycisku START nagrywanie powinno rozpocząć się po kilku sekundach opóźnienia, podczas którego wyświetlany byłby komunikat o ustawieniu się zgodnie z wyświetlanym konturem.

2. Przyciski POWTÓRZ i KAMERA w trybie wyświetlania filmu przykładowego powinny zostać zamienione miejscami.

3. Etykiety przycisków powinny mieć mniej techniczne brzmienie. Etykietę KAMERA można zamienić na NAUKA, a etykietę FILM na PRZYKŁAD.

4. Informacja o skrótach klawiszowych powinna być łatwiej dostępna.

Podczas testów potwierdzono również, że **wyświetlanie konturu sylwetki** oraz **zastosowanie odbicia lustrzanego** w wyświetlanych filmach i w obrazie z kamery jest wygodne i bardzo pomocne dla użytkowników.

7. Rozwój aplikacji

Stworzenie w pełni kompletnej i wygodnej platformy e-learningowej to bardzo duży i ambitny projekt, wykraczający poza ramy projektu inżynierskiego. w tym rozdziale przedstawione zostaną koncepcje i możliwości przyszłego rozwoju opisywanej aplikacji.

7.1. Rozwój interfejsu użytkownika

Przy rozwoju interfejsu użytkownika i układu graficznego strony należy przede wszystkim wziąć pod uwagę przedstawione w podrozdziale 6.3, wnioski z przeprowadzonych testów użyteczności i wyeliminować wszystkie zauważone błędy oraz niedogodności.

Jak już wspomniano, na obecnym etapie serwer dokonujący rozpoznawania gestów wymaga, aby osoba migająca znajdowała się w odpowiedniej odległości od komputera. Problemem staje się rozpoczynanie i zatrzymywanie nagrywania. Poza proponowanym rozwiązaniem opóźnienia rozpoczęcia nagrywania (patrz podrozdz. 6.3.2), rozwiązaniem może być zaimplementowanie rozpoznawania dźwięku, tak, żeby komenda głosowa *Start* rozpoczynała nagrywanie, bez konieczności naciśnięcia przycisku START. Rozwiązanie takie jest jednak uprawnione tylko wtedy, gdy docelową grupą aplikacji będą osoby słyszące chcące nauczyć się języka migowego.

Aby aplikacja do nauki języka migowego była prawdziwie multiplatformowa, konieczne jest przystosowanie jej do urządzeń mobilnych. Obecnie czcionki i układ strony nie dostosowują się do małych ekranów urządzeń mobilnych oraz występuje błąd przy przesyłaniu nagrania z przeglądarki Firefox Mobile. Na dalszym etapie rozwoju aplikacja powinna być dostosowana do przeglądarek mobilnych lub zostać przepisana tak, by mogła być uruchamiana natywnie w mobilnych systemach operacyjnych.

7.2. Konta użytkowników

Aplikacja do nauki języka powinna umożliwiać utworzenie konta, zalogowanie się i zapamiętywać indywidualny postęp każdego użytkownika. Zmiana ta wymaga wdrożenia bazy danych oraz zmodyfikowania interfejsu użytkownika, tak, żeby wyświetlać postęp i poprzednie wyniki. Sugerowaną bazą danych może być baza MongoDB, często wykorzystywana przy tworzeniu aplikacji z użyciem Node.js, Express.js i AngularJS w ramach zestawu oprogramowania MEAN [27].

7.3. Zaimplementowanie algorytmu SuperMemo

Na obecnym etapie aplikacja umożliwia wybranie konkretnego gestu i jego naukę, ale nie jest zaimplementowany żaden algorytm powtarzania. Dowiedziono, że do nauki słownictwa bardziej efektywne od tradycyjnego powtarzania jest uczenie metodą *SuperMemo* opracowaną przez Piotra Woźniaka i Edwarda Gorzelańczyka [28]. Metoda ta opiera się na założeniu, że nauczone słownictwo trzeba powtarzać, ale tracenie czasu na powtarzanie już dobrze utrwalonych słów jest nieefektywne. W tym celu opracowane są algorytmy uwzględniające szybkość zanikania śladów pamięciowych. Czas powtórek jest dobierany dla każdego słowa osobno na podstawie tego ile dana osoba popełniła pomyłek dla danego słowa oraz w jakich odstępach czasu. Zaimplementowanie algorytmu typu *supermemo* w aplikacji do nauczania języka migowego znacznie zwiększyłoby atrakcyjność aplikacji oraz szybkość uczenia się gestów.

7.4. Wykorzystanie strumieniowania

W obecnej wersji aplikacji, gest wykonywany przez użytkownika nagrywany jest w całości po stronie przeglądarki. Dopiero po zakończeniu nagrywania film przesyłany jest na serwer.

Lepszym rozwiązaniem technicznym, planowanym w dalszym rozwoju aplikacji byłoby wykorzystanie technologii strumieniowania jaką również oferuje standard WebRTC. Dzięki temu, już podczas nagrywania gestu, dane wysyłane byłyby na serwer na bieżąco.

8. Podsumowanie

W ramach projektu inżynierskiego, opisywanego w tej pracy, udało zrealizować się postawione na początku cele. Stworzono działającą aplikację webową służącą do nauki języka migowego. Aplikacja została zrealizowana za pomocą nowoczesnych technologii, które w ostatnim czasie zdobywają co raz większą popularność. Przeprowadzono także badania użyteczności interfejsu z udziałem użytkowników, dzięki czemu przekonano się jakie są jego mocne i słabe strony. Zrealizowano także połączenie z serwerem przetwarzającym gesty opracowywanym w ramach oddzielnego projektu.

Trzeba mieć na uwadze, że opracowana aplikacja, jest dopiero prototypem, posiadającym pewne błędy. Autor pracy jest przekonany, że aplikacja ma duży potencjał rozwoju, a jej szkielet może zostać wykorzystany także do połączenia z innymi algorytmami, w celu zaimplementowania zupełnie odmiennej funkcjonalności.

9. Bibliografia

[1] Akademia Górniczo-Hutnicza w Krakowie, VoicePIN.com. *Projekt WiTKoM*. 2015. URL: http://witkom.info/.

[2] Przemysław Węgrzynowicz. "Serwer usług rozpoznania gestów języka migowego." Praca inżynierska. Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie, 2016. Promotor: dr inż. Jakub Gałka.

[3] Piotr Tomaszewski i in. *Nauczmy się rozumieć nawzajem. Poradnik dla słyszących o niedosłyszących i głuchych*. Polski Związek Głuchych Zarząd Główny, 2010. URL: http://www.pzg.org.pl/rokdownloads/SC/poradnik_PZG.pdf.

[4] Katarzyna Łukaszewska. "Matematyka po chińsku, czyli jak wygląda sytuacja Głuchych w Polsce". W: *Uniwersytet Warszawski - pismo uczelni* (lip. 2007). ISSN: 1640-2758.

[5] Node.js Foundation. *Node.js*. Wer. 5.2.0. Lic. MIT. URL: https://nodejs.org/.

[6] Alexandru Vladutu. *Node.js Tutorial – Step-by-Step Guide For Getting Started*. 2014. URL: https://www.airpair.com/javascript/node-js-tutorial.

[7] StrongLoop. *Express.js*. Wer. 4.13.3. Lic. MIT. URL: http://expressjs.com/.

[8] Guillermo Rauch i in. *Socket.IO*. Wer. 1.3.7. Lic. MIT. URL: http://socket.io/.

[9] Google. *AngularJS*. Wer. 1.4.8. Lic. MIT. URL: https://angularjs.org/.

[10] Autorzy Wikipedii. *Programowanie deklaratywne*. 2015. URL: https://pl.wikipedia.org/wiki/Programowanie_deklaratywne.

[11] Rzepiński M. *AngularJS #1 – Wprowadzenie*. 2013. URL: http://mrzepinski.pl/angularjs-1-wprowadzenie.html.

[12] Twitter. *Bootstrap*. Wer. 3.3.6. Lic. MIT. URL: https://getbootstrap.com/.

[13] World Wide Web Consortium. *WebRTC*. URL: https://webrtc.org/.

[14] Ulrich von Agris i in. "Recent developments in visual sign language recognition". W: *Universal Access in the Information Society* (2007). DOI: 10.1007/s10209-007-0104-x.

[15] David Miller. *Start Bootstrap - Simple Sidebar*. Wer. 1.0.4. Lic. Apache 2.0. URL: https://github.com/IronSummitMedia/startbootstrap-simple-sidebar.

[16] Jan Kovarik. *Sharp and clean symbols – Glyphicons.com*. 2015. URL: https://glyphicons.com/.

[17] John W. Long, Alexander Kohlhofer. *CSS Spinners*. Wer. 2.2.0. Lic. MIT. URL: https://github.com/jlong/css-spinners.

[18] RightScale. *angular-keyboard*. Wer. 0.1.0. Lic. MIT. URL: http://code.gampleman.eu/angular-keyboard/.

[19] Minko Gechev. *Multi-User Video Conference with WebRTC*. 2014. URL: http://blog.mgechev.com/2014/12/26/multi-user-video-conference-webrtc-angularjs-yeoman/.

[20] Brian Ford. *Socket.IO component for AngularJS*. Wer. 0.7.0. Lic. MIT. URL: https://github.com/btford/angular-socket-io.

[21] Ideacto, red. *Testy użyteczności z użytkownikami*. 2015. URL: http://szkoleniausability.pl/narzedziownik-uzytecznosci-czesc-1.pdf.

[22] A. Czoska i in. *Badania UX*. 2015. URL: http://kck.wikidot.com/_zajecia:08:ux.

[23] Ewa Sobula. "Test 5 sekund". W: *UXbite* (28 paź. 2010). URL: http://uxbite.com/2010/10/test-5-sekund/.

[24] UsabilityHub. *The original five second test*. 2015. URL: https://usabilityhub.com/five-second-test.

[25] Ewa Sobula. "Super szybki przewodnik po testach użyteczności". W: *UXbite* (23 sierp. 2011). URL: http://uxbite.com/2011/08/super-szybki-przewodnik-po-testach-uzytecznosci/.

[26] Jakob Nielsen. "How Many Test Users in a Usability Study?" W: *Nielsen Norman Group* (4 lip. 2012). URL: https://www.nngroup.com/articles/how-many-test-users/.

[27] Amos Haviv. *MEAN.JS Open-Source Full-Stack Solution*. 2016. URL: http://meanjs.org/.

[28] E. J. Gorzelańczyk P. A. Woźniak. *SuperMemo*. 2016. URL: https://www.supermemo.com/.

Zał. 1a. Kod HTML aplikacji webowej

```
<!DOCTYPE html>
<html ng-app="witkomApp">
<head>
  <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
  <link rel="stylesheet" href="bower_components/bootstrap/dist/css/
     bootstrap.css">
  <link rel="stylesheet" href="css/simple-sidebar.css">
  <link rel="stylesheet" href="css/learning.css">
  <link rel="stylesheet" href="css/hexdots.css" type="text/css">
  <title>WiTKoM Learning</title>
  <script src="bower_components/jquery/dist/jquery.min.js"></script>
  <script type="text/javascript" src="js/adapter.js"></script>
  <script src="/socket.io/socket.io.js"></script>
  <script type="text/javascript" src="bower_components/angular/
     angular.js"></script>
  <script src="bower_components/angular-animate/angular-animate.js"></s
     cript>
  <script src="js/angular-keyboard.js"></script>
  <script src="bower_components/angular-socket-io/socket.js"></script>
  <script type="text/javascript" src="js/angularApp.js"></script>
  <script type="text/javascript" src="js/controllers.js"></script>
  <script type="text/javascript" src="js/services.js"></script>
</head>
<body>
  <div ng-controller="LearningController" id="wrapper" ng-class="{'
     toggled': toggled}">
    <!-- Sidebar -->
    <div id="sidebar-wrapper">
      <ul class="sidebar-nav">
        <li class="sidebar-brand">
          <a href="#">Początkujący</a>
        </li>
        <li ng-repeat="gesture in gestures">
```

```
30          <keyboard-shortcut keyboard-shortcut="down"
31                             keyboard-action="setNextGesture()"
32                             keyboard-prevent-default>
33          </keyboard-shortcut>
34          <keyboard-shortcut keyboard-shortcut="up"
35                             keyboard-action="setPreviousGesture()"
36                             keyboard-prevent-default>
37          </keyboard-shortcut>
38          <a ng-class="{'selected-true': selected(gesture)}"
39             ng-click="setGesture(gesture)">{{gesture.fullName}}
40          </a>
41        </li>
42      </ul>
43    </div>
44    <!-- /#sidebar-wrapper -->
45    <!-- Page Content -->
46    <div ng-controller="RecordingController" id="page-content-wrapper">
47      <div class="container-fluid">
48        <div ng-hide="error">
49          <div class="row">
50            <div class="col-md-3">
51              <!-- Przycisk do chowania menu bocznego-->
52              <a id="toggle-button"
53                 type="button"
54                 class="btn btn-default btn-lg"
55                 aria-label="Toggle left sidebar"
56                 ng-click="toggleSidebar()">
57                <span class="glyphicon"
58                      id="sidebarButton"
59                      ng-class="toggled ? 'glyphicon-menu-right' : '
                           glyphicon-menu-left'"
60                      aria-hidden="true">
61                </span>
62              </a>
63            </div>
64            <div ng-class="toggled ? 'col-md-6' : 'col-md-8
                 col-md-offset-2'">
65              <div class="page-header" id="selected-gesture">
66                <h2>{{selectedGesture.fullName}}</h2>
67              </div>
68            </div>
69          </div>
70
```

```
71          <div class="row">
72            <div ng-class="toggled ? 'col-md-6 col-md-offset-3' : '
                 col-md-8 col-md-offset-2'">
73              <div class="embed-responsive embed-responsive-4by3">
74                <div class="video-overlay fade"
75                     ng-show="showOutline"></div>
76                <video id="video-player"
77                       class="embed-responsive-item"
78                       ng-src="{{getLocalVideo()}}" autoplay>
79                </video>
80              </div>
81            </div>
82          </div>
83
84          <div class="row">
85            <div ng-class="toggled ? 'col-md-2 col-md-offset-3' : '
                 col-md-2 col-md-offset-2'">
86              <a type="button"
87                 class="main-buttons btn btn-lg btn-block btn-primary"
88                 ng-click="changeSource()"
89                 keyboard-shortcut="f"
90                 keyboard-title="Przełącz widok">
91                <i class="glyphicon"
92                   ng-class="source === 'camera' ? 'glyphicon-film' : '
                      glyphicon-facetime-video' ">
93                </i> {{source === 'camera' ? 'Film' : 'Kamera'}}
94              </a>
95            </div>
96
97            <div ng-class="toggled ? 'col-md-2' : 'col-md-4'">
98              <div ng-show="LearningCtrlVariables.loading ||
                   LearningCtrlVariables.ready"
99                   class="panel panel-default score">
100               <div class="panel-body"
101                    ng-class="{'score-body' :
                        LearningCtrlVariables.ready}">
102                 <strong class="h2"
103                         ng-show="LearningCtrlVariables.ready">
104                   Wynik:
105                   <span>{{result | number:0}}%</span>
106                 </strong>
107                 <i ng-show="LearningCtrlVariables.loading"
108                    style="transform: scale(0.5)"
```

```
109                     id="spinner"
110                     class="hexdots-loader ng-animate-disabled"></i>
111                 </div>
112               </div>
113             </div>
114
115             <div ng-class="toggled ? 'col-md-2' : 'col-md-2'"
116                  ng-show="source==='camera'">
117               <a type="button"
118                  class="main-buttons btn btn-lg btn-block"
119                  ng-class="state==='idle' ? 'btn-success' : 'btn-danger'
                        "
120                  ng-click="startStop()"
121                  keyboard-shortcut="space"
122                  keyboard-title="Rozpocznij/zatrzymaj nagrywanie"
123                  keyboard-prevent-default="true"
124                  shortcut-if="source==='camera'">
125                 <i class="glyphicon"
126                    ng-class="state==='idle' ? 'glyphicon-facetime-video'
                          : 'glyphicon-stop'"></i> {{state==='idle' ? '
                       Start' : 'Stop'}}
127               </a>
128             </div>
129             <div ng-class="toggled ? 'col-md-2' : 'col-md-2'"
130                  ng-show="source==='film'">
131               <a type="button"
132                  class="main-buttons btn btn-lg btn-block btn-primary"
133                  ng-click="videoReplay()"
134                  keyboard-shortcut="r"
135                  shortcut-if="source=='film'"
136                  keyboard-title="Odtwórz ponownie">
137                 <i class="glyphicon glyphicon-repeat"></i> Powtórz
138               </a>
139             </div>
140           </div>
141         </div>
142         <div class="alert alert-danger" role="alert" ng-hide="!error"
              ng-bind="error"></div>
143       </div>
144     </div>
145   </div>
146 </body>
147 </html>
```

Zał. 1b. Kod CSS aplikacji webowej

```css
.sidebar-nav a, #toggle-button {
    outline: none!important;
}

.sidebar-nav .selected-true {
    text-decoration: none;
    color: #fff;
    background: #00475a;
    transition: background 0.15s ease;
}

.sidebar-nav li:hover:not(.sidebar-brand) {
    background: #002f3c;
}

.keyboard-selected {
    background: #002f3c;
}

#toggle-button{
    position: fixed;
    margin-left: -35px;
    margin-top: -20px;
    border-radius: 0;
    z-index: 9999;
    background: rgba(255, 255, 255, 0.8);
}

.video-overlay{
    opacity: 1;
    position: absolute;
    width: 100%; height: 100%;
    z-index: 100;
```

```
34      background: rgba(0,0,0,0.2);
35      background-image: url(../pictures/Przemek.png);
36      background-position: center bottom;
37      background-size: 38%;
38      background-repeat: no-repeat;
39      transition: 0.5s linear opacity;
40  }
41
42  #video-player{
43      position: absolute;
44      z-index: 99;
45      transform: scaleX(-1);
46  }
47
48  .main-buttons{
49      margin-top: 20px;
50  }
51
52  @media(max-width:768px) {
53      #sidebarButton{
54          transform: rotate(180deg);
55          -webkit-transform: rotate(180deg);
56          -moz-transform: rotate(180deg);
57          -o-transform: rotate(180deg);
58      }
59  }
60
61  .score{
62      margin-top: 18px;
63      text-align: center;
64
65  }
66
67  .score-body{
68      padding-top: 7px;
69      padding-bottom: 7px;
70  }
71
72  .score span{
73      color: rgb(38, 127, 216);
74  }
75
76
```

```
77 .fade.ng-hide {
78   transition: 0.1s linear opacity;
79   opacity: 0;
80 }
81 
82 #spinner {
83     margin-bottom: 2.5px;
84 }
85 
86 #selected-gesture{
87     margin-top: -20px;
88 }
```

Zał. 1c. Kod JavaScript aplikacji po stronie klienta

angularApp.js

```
var witkomApp = angular.module('witkomApp', [
    'witkomControllers',
    'witkomServices'
]);
```

services.js

```
(function () {
    'use strict';

    var witkomServices = angular.module('witkomServices', ['btford.
        socket-io']);

    witkomServices.factory('SocketIO',
        function (socketFactory) {
            return socketFactory();
        });

    witkomServices.factory('VideoStream', function ($q) {
        var stream;
        return {
            get: function () {
                if (stream) {
                    return $q.when(stream);
                } else {
                    var d = $q.defer();
                    navigator.getUserMedia(
                        {
                            video: true,
                            audio: false
                        },
```

```
                        function (s) {
                            stream = s;
                            d.resolve(stream);
                        },
                        function (e) {
                            d.reject(e);
                        }
                    );
                    return d.promise;
                }
            }
        };
    });
}());
```

controllers.js

```
(function () {
    'use strict';

    var witkomControllers = angular.module('witkomControllers', ['
        angular-keyboard', 'ngAnimate'])
            .config(function ($animateProvider) {
                $animateProvider.classNameFilter(/^(?:(?!ng-animate-
                    disabled).)*$/);
            });

    witkomControllers.controller('LearningController', ['$scope', '$http
        ', 'SocketIO', '$document', function ($scope, $http, SocketIO,
        $document) {
        // Get gestures from server (JSON file)
        $scope.gestures = [];
        $scope.LearningCtrlVariables = {
            ready : false,
            loading : false
        };

        $scope.selectedGesture = null;
        $http.get('js/gestures.json').success(function (data) {
            $scope.gestures = data;
            $scope.selectedGesture = $scope.gestures[0];

        });
```

```
23          // Selection of gesture
24          $scope.setGesture = function (gesture) {
25              $scope.selectedGesture = gesture;
26              $scope.LearningCtrlVariables.ready = false;
27              $scope.LearningCtrlVariables.loading = false;
28          };
29
30          $scope.setNextGesture = function () {
31              var index = $scope.gestures.indexOf($scope.selectedGesture);
32              if (index < $scope.gestures.length - 1) {
33                  index += 1;
34              } else {
35                  index = 0;
36              }
37              $scope.selectedGesture = $scope.gestures[index];
38              $scope.LearningCtrlVariables.ready = false;
39          };
40
41          $scope.setPreviousGesture = function () {
42              var index = $scope.gestures.indexOf($scope.selectedGesture);
43              if (index > 0) {
44                  index -= 1;
45              } else {
46                  index = $scope.gestures.length - 1;
47              }
48              $scope.selectedGesture = $scope.gestures[index];
49              $scope.LearningCtrlVariables.ready = false;
50          };
51
52          $scope.selected = function (gesture) {
53              return $scope.selectedGesture === gesture;
54          };
55
56          // Toggle left sidebar
57          $scope.toggled = false;
58          $scope.toggleSidebar = function () {
59              $scope.toggled = !$scope.toggled;
60          };
61
62          $scope.showOutline = true;
63
64          SocketIO.on('results', function (data) {
65              $scope.LearningCtrlVariables.ready = true;
```

```
66              $scope.LearningCtrlVariables.loading = false;
67              $scope.result = data.result;
68          });
69
70          $scope.videoReplay = function () {
71              var curVid = $document.find('#video-player');
72              curVid.get(0).src = 'gestures/' + $scope.selectedGesture.
                    name + '.webm';
73          };
74      }]);
75
76      witkomControllers.controller('RecordingController', ['$scope', '$sce
            ', 'VideoStream', 'SocketIO', function ($scope, $sce,
            VideoStream, SocketIO) {
77          if ((typeof MediaRecorder === 'undefined') || !navigator.
                getUserMedia) {
78              $scope.error = 'Moduł MediaRecorder nie jest wspierany przez
                     Twoją przeglądarkę :( Spróbuj użyć najnowszej przeglą
                    darki Firefox.';
79              return;
80          }
81          $scope.state = 'idle';
82          $scope.source = 'camera';
83          var stream;
84          VideoStream.get().then(function (s) {
85              stream = s;
86              var mediaRecorder = new MediaRecorder(stream);
87              mediaRecorder.ondataavailable = function (e) {
88                  var videoObject = {
89                      'video' : e.data,
90                      'nameOfGesture' : $scope.selectedGesture.name
91                  };
92                  SocketIO.emit("video", videoObject);
93              };
94
95              $scope.startStop = function () {
96                  if ($scope.state === 'idle') {
97                      $scope.state = 'recording';
98                      mediaRecorder.start();
99                      $scope.showOutline = false;
100                     console.log("recorder started");
101                     $scope.LearningCtrlVariables.ready = false;
102                     $scope.LearningCtrlVariables.loading = false;
```

```
103                 } else {
104                     $scope.state = 'idle';
105                     mediaRecorder.stop();
106                     $scope.showOutline = true;
107                     console.log("recorder stopped");
108                     $scope.LearningCtrlVariables.loading = true;
109                     $scope.LearningCtrlVariables.ready = false;
110                 }
111             };
112             stream = URL.createObjectURL(stream);
113         },  function () {
114             $scope.error = 'Brak dostępu do kamery. Odśwież stronę i
                zezwól na udostępnienie obrazu z kamery.';
115         });
116 
117         $scope.changeSource = function () {
118             if ($scope.source === 'camera') {
119                 $scope.source = 'film';
120                 $scope.showOutline = false;
121             } else {
122                 $scope.source = 'camera';
123                 $scope.showOutline = true;
124             }
125         };
126 
127         $scope.getLocalVideo = function () {
128             var filmURL;
129             if ($scope.source === 'film') {
130                 if ($scope.selectedGesture === null) {
131                     filmURL = stream;
132                 } else {
133                     filmURL = 'gestures/' + $scope.selectedGesture.name
                        + '.webm';
134                 }
135             } else {
136                 filmURL = stream;
137             }
138             return $sce.trustAsResourceUrl(filmURL);
139         };
140     }]);
141 }());
```

Zał. 1d. Kod JavaScript aplikacji po stronie serwera

```
1  'use strict';
2
3  var debug = require('debug')('http');
4  var express = require('express');
5  var app = express();
6  var server = require('http').Server(app);
7  var fs = require("fs");
8  var ffmpeg = require('fluent-ffmpeg');
9
10 var mainServer = function () {
11     var net = require('net');
12     var matlabNetSocket = net.connect({port: 8000},
13         function () {
14             console.log('Connected to MATLAB :)');
15
16             app.use(express.static('public'));
17             var server = app.listen(3000, function () {
18                 var host = server.address().address;
19                 var port = server.address().port;
20                 console.log('WiTKoM app is listening at http://%s:%s',
                       host, port);
21             });
22
23             var browserSocketIO = require('socket.io')(server);
24             // Take video from browser using socket.io:
25             browserSocketIO.on('connection', function (socket) {
26                 var fileName;
27                 var newFilePath;
28                 socket.on('video', function (videoObject) {
29                     var videoData = videoObject.video;
30                     console.log('Name of gesture: ' + videoObject.
                           nameOfGesture);
31                     var path = __dirname + '/matlab/Records';
```

```
32                      // Name of recording without extension:
33                      fileName = Math.random().toString(16).substr(2) + '_
                            ' + socket.id;
34                      // Build whole filepath:
35                      var fileExtension = 'webm';
36                      var fileRootNameWithBase = path + '/' + fileName;
37                      var filePath = fileRootNameWithBase + '.' +
                            fileExtension;
38
39                      // Save file to webm, and then, in callback convert
                            it to avi (mpeg4):
40                      fs.writeFile(filePath, videoData, function (err) {
41                          if (err) {
42                              throw err;
43                          }
44                          debug('Original record path: ', filePath);
45                          // This regexp change file extension to .avi:
46                          newFilePath = filePath.replace(/\.[^\/.]+$/, ".
                                avi");
47                          // Conversion of video (using ffmpeg) to avi,
                                with mpeg4 codec and highest quality:
48                          var command = ffmpeg(filePath)
49                              .videoCodec('mpeg4')
50                              .on('end', function () {
51                                  debug('Converted record path: ',
                                        newFilePath);
52                                  var jsonForMatlab = {
53                                      'fileName' : fileName + '.avi',
54                                      'gesture' : videoObject.
                                            nameOfGesture
55                                  };
56                                  matlabNetSocket.write(JSON.stringify(
                                        jsonForMatlab) + '\n');
57                                  fs.unlink(filePath, function () {
58                                      debug('Webm deleted!');
59                                  });
60                              })
61                              .on('error', function (err) {
62                                  console.log('an error happened: ' + err.
                                        message);
63                              })
64                              .outputOptions('-q:v 1')
65                              .save(newFilePath);
```

```
66                      });
67                  });
68              });
69
70              matlabNetSocket.on('data', function (data) {
71                  var recog = JSON.parse(data);
72                  debug(recog.fileName);
73                  console.log('Score: ' + recog.result);
74                  browserSocketIO.sockets.sockets.forEach(function (s) {
75                      if (recog.fileName.endsWith(s.id + '.avi')) {
76                          s.emit('results', recog);
77                          debug('Result sent to browser :)');
78                      }
79                  });
80              });
81          });
82  };
83
84  if (process.platform === 'linux') {
85      console.log('Starting MATLAB...');
86      var spawn  = require('child_process').spawn,
87          matlabProcess = spawn('matlab', ['-nodisplay', '-nosplash', '-
              nodesktop', '-r', 'runMatlabServer'], {cwd : 'matlab'});
88
89      matlabProcess.stdout.on('data', function (data) {
90          debug('MATLAB: ' + data.toString());
91          setTimeout(function() {
92              if (data.toString().includes('Matlab_ready')) {
93                  mainServer();
94              }
95          }, 100);
96
97      });
98  } else {
99      mainServer();
100 }
```

Załącznik nr 2. Pre-kwestionariusz do moderowanych testów użyteczności

1. Wiek
□ poniżej 16 lat
□ 16-20 lat
□ 20-24 lat
□ 25-30 lat
□ 31-40 lat
□ 41-50 lat
□ 51 i więcej lat

2. Wykształcenie
□ średnie
□ wyższe
inne _____

3. Wykonywany zawód / poziom edukacji
_____

4. Jak długo już korzystasz z Internetu?
_____

5. Ile godzin przeciętnie korzystasz z Internetu w ciągu dnia?
_____

6. Na jakich urządzeniach korzystasz z Internetu? (zaznacz na liście wszystkie elementy, z których korzystasz)
□ Komputer stacjonarny
□ Laptop
□ Tablet
□ Netbook
□ Smartfon
□ Zwykły telefon komórkowy (nie smartfon)
Inne:_____

7. Czy korzystasz z aplikacji webowych typu Microsoft Office Online, Google Docs itp.? Jeśli tak to z jakich?
_____

8. Jak często używasz kamerki internetowej?
□ Nigdy
□ Kilka razy w roku
□ Kilka razy miesięcznie
□ Kilka razy w tygodniu
□ Codziennie

9. Czy zdarzyło Ci się używać kamer internetowej na stronach internetowych?
_____

10. Czy korzystasz z gier przeglądarkowych? Jeśli tak to z jakich?
_____

11. Czy uczysz się języków obcych za pomocą Internetu? Jeśli tak to z jakich stron internetowych korzystasz?

_____

12. Czy korzystasz z interaktywnych kursów e-learningowych (takich jak Coursera, CodeAcademy, CodeSchool, KhanAcademy)?

_____

13. Czy znasz oprogramowanie do nauki języków typu super-memo? Jeżeli tak, to czy korzystasz z niego?

_____

14. Czy znasz jakiś język migowy? Jeżeli tak to jaki?

_____

15. Czy masz w rodzinie osobę głuchą lub niesłyszącą?

_____

16. Jak oceniasz swoje umiejętności w posługiwaniu się komputerem?

☐ Bardzo słabo ☐ Słabo ☐ Średnio ☐ Dobrze ☐ Bardzo dobrze

Załącznik nr 3. Post-kwestionariusz do moderowanych testów użyteczności

1. Czy masz jakieś uwagi odnośnie listy gestów znajdującej się z lewej strony?

_____

2. Jak podoba Ci się wygląd serwisu?

_____

3. Czy obrys sylwetki okazał się pomocny, żeby poprawnie się ustawić, czy zaciemniał obraz?

_____

4. Co sądzisz o zastosowaniu odbicia lustrzanego w filmikach instruktażowych i obrazie z kamery?

_____

5. Czy wygodniejsze jest korzystanie z klawiatury czy z myszy? Napisz dlaczego.

_____

6. Czego oczekiwałbyś/oczekiwałabyś od strony do nauki języka migowego?

_____