POLSKO-JAPOŃSKA WYŻSZA SZKOŁA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Bartosz Marganiec

Nr albumu 12769

System internetowy do pracy nad korpusem Polskiego Języka Migowego

Praca magisterska napisana pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, czerwiec 2016

Streszczenie

Praca dotyczy narzędzi wykorzystywanych do obróbki i analizy klipów filmowych zawierających materiał językowy w Polskim Języku Migowym (PJM). Analizuje je i przedstawia propozycję rozwiązania technicznego wspomagającego proces obróbki i systematyzowania danych językowych zawartych w klipach, które w założeniu ma być dostępne zdalnie i bez potrzeby instalacji specjalistycznego oprogramowania oraz posiadania odpowiedniego sprzętu komputerowego. Języki migowe stanowią przedmiot zainteresowań lingwistów, którzy, ze względu na brak formy pisanej tych języków, zmuszeni są korzystać z nowotworzonych komputerowych programów do analizy danych. Ze względu na wielowymiarowość języka migowego analiza automatyczna jest niemożliwa i musi być wykonywana przez człowieka. Niewiele jest narzędzi do opisu i systematyzacji języka, są one też przeznaczone tylko do tego celu. Praca przedstawia prototyp systemu, który może zostać wykorzystany w celu stworzenia systemu do anotacji korpusu albo systemu do tworzenia napisów, a także w innych celach.

Spis treści

1. Wstęp

Prace lingwistyczne przeprowadzane nad polskim językiem migowym są prowadzone stosunkowo od niedawna i jest niewiele narzędzi wspierających lingwistów. Autor pracy przez kilka lat czynnie uczestniczył w prowadzeniu badań w Pracowni Lingwistyki Migowej na Uniwersytecie Warszawskim. Jest to dziedzina, która w Polsce rozwija się intensywnie dopiero od około 2010 roku. Ze względu na bliski związek Autora z polskim językiem migowym i z powszechnie występującymi stereotypami na jego temat, Autor uznał za wskazane przybliżyć Czytelnikowi pokrótce związane z nim zagadnienia, czemu poświęca niniejszy Wstęp. Przestawiony jest w nim krótki opis polskiego języka migowego, społeczności, która się nim posługuje (a do której Autor sam należy), oraz krótką historię badań lingwistycznych.

1.1. Polski Język Migowy

W niniejszym podrozdziale przedstawione są podstawowe informacje na temat Polskiego Języka Migowego, który był główną motywacją powstania niniejszej pracy.

1.1.1 Podstawowe informacje o polskim języku migowym i społeczności głuchych

Polski Język Migowy jest językiem, którym powszechnie posługują się Głusi w Polsce. Ze względu na międzynarodową konwencję jego pełna nazwa jest właśnie taka, a nie inna. Przykładowo, w USA Głusi posługują się Amerykańskim Językiem Migowym (American Sign Language - ASL), francuscy Głusi Francuskim Językiem Migowym (Langue des Signes Française - LSF). Nie jest to zwyczajny zbiór gestów, pantomima, czy „wskazywanie palcem". Jest to taki sam język, jak inne języki foniczne, np. język polski, angielski czy francuski. Podstawowa różnica leży w sposobie przekazu – wszystkie języki foniczne korzystają z kanału audytywno-werbalnego, natomiast języki migowe z kanału wizualnego. Badania wykazały, że pomimo innego kanału komunikacji, język migowy jest przetwarzany w tym samym ośrodku mózgu, co język foniczny [1]. Odmienny kanał komunikacji i niezależne od języka fonicznego powstanie języka skutkują zupełnie innym zestawem reguł gramatycznych i innym słownikiem. Jak to w przypadku innych języków bywa, nie wszystkie słowa są możliwe do dosłownego przetłumaczenia i nie wszystkie reguły gramatyczne występują we wszystkich językach. Języki migowe nie są tu wyjątkiem.

Społeczności głuchych istniały od dawna, nie wiadomo jednak, czy korzystały one z już wykształconego języka migowego. Przyjmuje się, że jego intensywny rozwój i upowszechnienie nastąpiło w XVIII wieku, gdy powstały szkoły specjalne z internatem dla osób głuchych. Były to miejsca, w których powstały warunki, aby język migowy mógł się rozwijać. Przyjmuje się, że Polski

Język Migowy powstał wraz z utworzeniem pierwszej szkoły dla głuchych w Polsce – Instytutu Głuchoniemych i Ociemniałych, który został założony z inicjatywy ks. Jakuba Falkowskiego w 1817 roku. Nie ma dokładnych statystyk co do tego, ile osób posługuje się Polskim Językiem Migowym. Szacuje się, że może być to 20 do 50 tys. osób.

1.1.2 Język migowy jako obszar zainteresowań lingwistyki

Pierwsze badania lingwistyczne nad językiem migowym rozpoczęły się w USA. Willam Stokoe w latach 50 XX wieku zainteresował się amerykańskim językiem migowym. Był to punkt zwrotny w historii lingwistyki migowej. Początkowo podchodzono do tych badań krytycznie i dopiero w latach 70 i 80 przeprowadzone badania psycho- i neurolingwistyczne wykazały, że podstawy neurobiologiczne języków fonicznych i migowych są takie same [2]. Obserwacja ta była ważnym krokiem na drodze do uznania języków migowych za samodzielne i w pełni rozwinięte systemy komunikacyjne.

W chwili obecnej badania lingwistyczne nad językami migowymi są prowadzone w ponad 100 ośrodkach akademickich na całym świecie, m. in. w Australii, Japonii, Brazylii, Peru, Hiszpanii, Francji, Włoszech. Bada się zarówno języki migowe licznych społeczności, np. ASL (American Sign Language), BSL (British Sign Language), ale także tych mniejszych, np. izraelski język migowy, jordański język migowy, a nawet społeczności mieszczących się w pojedynczej wiosce, czy używanych przez jedno plemię, jak np. beduiński język migowy z wioski Al-Sajid na pustyni Negew, czy język kata kolok z balijskiej wioski Bengkala [2].

Pierwotne prace lingwistyczne skupiały się na wykazaniu, że języki migowe w istocie niczym się nie różnią od języków fonicznych, tj. są pełnowartościowymi językami naturalnymi. Obecnie badania mają charakter bardziej interdyscyplinarny i skupiają ekspertów z zakresu psychologii, neurobiologii, socjologii. Coraz większą rolę odgrywają nauki informatyczne w badaniach lingwistycznych. Ich zadaniem jest wspieranie analizy i przetwarzania zbiorów danych wizualnych, nagrań korpusowych, tekstów języka migowego.

Polskie badania językoznawcze rozpoczęły się na Wydziale Polonistyki Uniwersytetu Warszawskiego. W latach 1999 – 2002 w ramach projektu naukowo-badawczego „Studia nad kompetencją językową i komunikacją niesłyszących” (KBN 1 H01D 012 16). Obecnie na Uniwersytecie istnieje już Pracownia Lingwistyki Migowej stworzona i prowadzona przez dr Pawła Rutkowskiego. Nadrzędnym celem Pracowni jest opracowanie korpusu Polskiego Języka Migowego. Jest on rozwijany dzięki grantowi: *Wielopoziomowa anotacja lingwistyczna korpusu polskiego języka migowego (PJM)*; grant w ramach modułu 1.2 III edycji Narodowego Programu Rozwoju Humanistyki (NPRH) Ministerstwa Nauki i Szkolnictwa Wyższego (0111/NPRH3/H12/82/2014).

1.2. Cel pracy

Celem pracy jest analiza i przedstawienie rozwiązania informatycznego pozwalającego na przeprowadzanie badań językowych bez konieczności użycia wyspecjalizowanego sprzętu bądź oprogramowania. Rozwiązanie to może być wykorzystane również do stworzenia oprogramowania mającego za cel naukę języka migowego, tworzenia napisów dla osób niesłyszących.

1.3. Rozwiązanie przyjęte w pracy

Prototyp systemu jest modułową aplikacją internetową podzieloną na frontend i backend. Zastosowane technologie to JavaScript w połączeniu z RequireJS tworzące od strony frontendu aplikację do zarządzania opisem klipów filmowych. Do części backendowej wykorzystany został framework PHP Phalcon 3.0, baza Postgresql i nierelacyjna baza Redis. Backend został podzielony na dwa moduły. Pierwszy z nich zajmuje się obsługą frontendu, wyświetleniem widoków i logowaniem użytkownika, drugi umożliwia komunikację RESTową i zajmuje się przechowywaniem i serwowaniem danych do aplikacji frontowej.

1.4. Rezultaty pracy

Rezultatem pracy jest zaprojektowanie budowy modułowej systemu internetowego do pracy nad klipami video, oraz wykonanie jego prototypu. Zadaniem systemu jest możliwość indywidualnej pracy lingwistycznej i analizy językowej materiału filmowego bez dodatkowego wymaganego sprzętu i oprogramowania.

Projekt modularyzacji systemu uwzględnia możliwość samodzielnego napisania wtyczek zarówno do klienta systemu, jak i do backendu bez naruszania dotychczasowych funkcjonalności. Pozwala to na rozszerzanie możliwości systemu i dostosowanie do własnych potrzeb. Projektowana modularyzacja dotyczy dwóch obszarów – klienta napisanego w JavaScript i jego możliwości pracy nad klipem video, oraz backendu zapisującego wyniki pracy w wybranym miejscu.

Projekt zawiera wytyczne dotyczące budowy pluginów, które muszą być spełnione, aby mogły prawidłowo działać w systemie. Prototyp jest implementacją przyjętego projektu budowy systemu i prezentuje dwa pluginy do frontendu i dwa repozytoria do zapisu i odczytu danych w backendzie.

1.5. Organizacja pracy

W rozdziale drugim Autor prezentuje istniejące narzędzia do analizy korpusowej języków migowych. Zestawia je ze sobą i prezentuje ich możliwości, wady, oraz zalety.

W rozdziale trzecim Autor prezentuje koncepcje wynikające z rozważań w rozdziale drugim. Opisywane są założenia, jakie powinna spełniać aplikacja.

Rozdział czwarty prezentuje technologie i koncepcje wykorzystane przy tworzeniu prototypu. W przypadku mniej popularnych rozwiązań Autor podaje przykłady zastosowań i odnośniki do dokumentacji.

Rozdział piąty jest opisem prototypu aplikacji. Opisano architekturę prototypu i podział na moduły. Szczególną uwagę zwrócono na generyczność rozwiązań.

Rozdział szósty stanowi podsumowanie całej pracy. Przedstawione są w nim wady i zalety prototypu, oraz możliwości jego dalszego rozwijania.

2. Narzędzia do anotacji korpusowej języków migowych

Programy używane do anotacji korpusowej języków migowych służą do pracy nad materiałem filmowym. Mają one zestaw wspólnych funkcji. Przy anotowaniu materiału filmowego pozwalają na utworzenie tzw. „warstw", na których umieszczane są tokeny. Jeden film może mieć dowolną ilość warstw, różnie nazwanych, otagowanych, itd.. Każdy pozwala na zaznaczenie odpowiedniego fragmentu filmu i nadanie mu tokenu na wybranej warstwie. Ponieważ każda warstwa może mieć inne funkcje, taki film można anotować na różnych poziomach – gramatycznym, składniowym, można analizować ruchy tylko jednej ręki, głowy, czy innych części ciała, można również stworzyć warstwę z tłumaczeniem na język pisany. Programy posiadają możliwość przeglądania filmu z różnym tempem odtwarzania, można również przeglądać film klatka po klatce. Programy oferują możliwość wyszukiwania po tokenach wszystkich fragmentów filmu, w których występują. W poniższych podrozdziałach Autor opisuje, czym jest anotacją korpusowa, przedstawia najpopularniejsze aplikacje, ich działanie, oraz wady i zalety.

2.1. Anotacja korpusowa języków migowych

Przy analizie korpusowej analizowane są klipy filmowe z informatorami prezentującymi spontaniczne wypowiedzi na ustalone tematy. Otrzymują oni zadania komunikacyjne, takie jak spytanie się rozmówcy o drogę, czy opowiedzenie bajki. Nagrania są zapisywane i konwertowane do formatów dających się odczytać przez program wykorzystywany do anotacji.

Anotacja jest to opis danych językowych zawartych w materiale filmowym [3]. W jej ramach dokonuje się podziału materiału na tokeny i opisuje je metadanymi. Mogą to być np. znaczniki lingwistyczne, definiujące, czy dane wystąpienie tokena jest czasownikiem, czy np. przymiotnikiem, ale także dane o informatorze. W literaturze międzynarodowej wykorzystuje się termin *annotation,* którego bezpośrednim tłumaczeniem na język polski jest *adnotacja,* jednak w polskich pracach lingwistycznych przyjęto tłumaczenie *anotacją* [4].

Zanotowanie korpusu jest czasochłonną pracą i kolejnym etapem po zebraniu materiału korpusowego od informatorów, który może trwać jeszcze przez wiele lat. Powstało kilka programów wspomagających ten proces. Są to Eudico Linguistic Annotator, iLex, oraz Sign Stream. Mają one zestaw wspólnych funkcji. Podstawowe z nich to:

- Okno podglądu filmu
- Okno opisu lingwistycznego

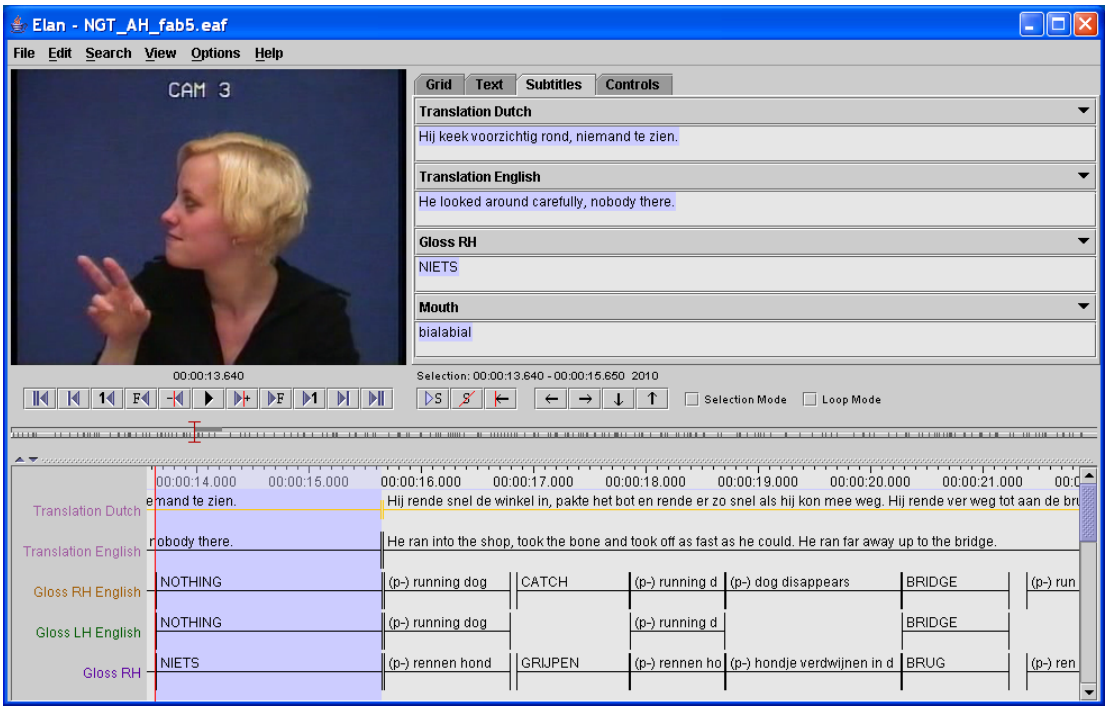
- Przeszukiwanie opisanych danych

2.2. EUDICO Linguistic Annotator

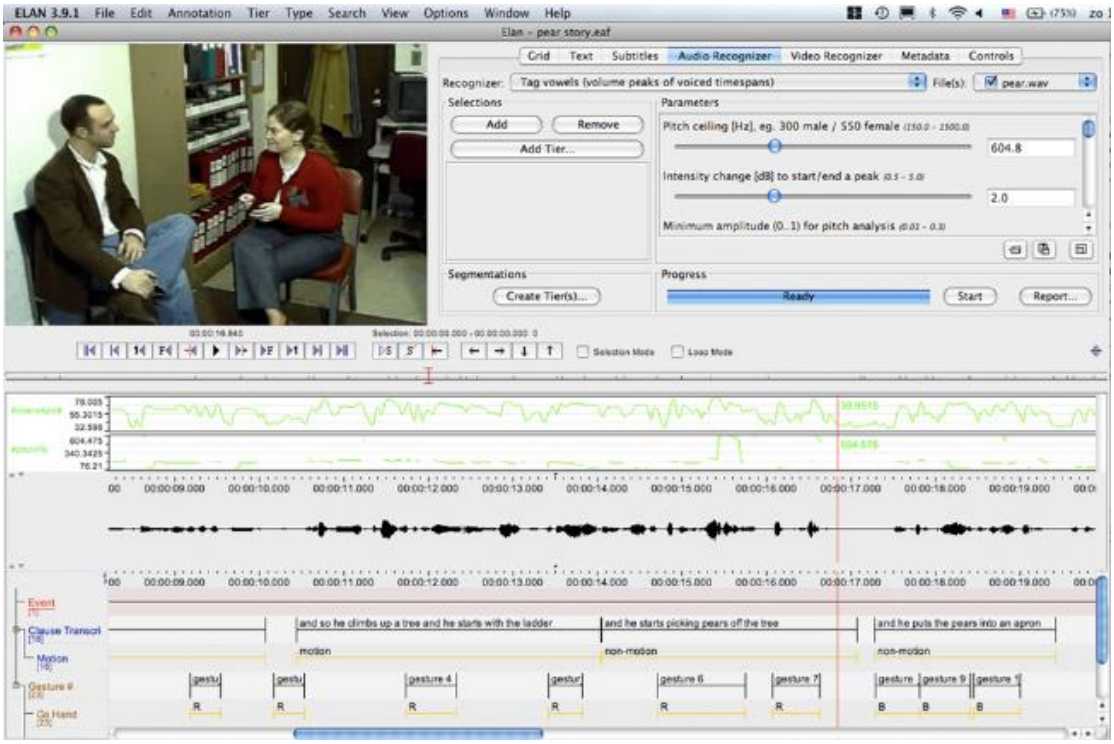
ELAN powstał w Instytucie Maxa Plancka w Nijmegen w 2005 roku [5]. Jest on bezpłatny i regularnie aktualizowany. Jego podstawowe zastosowanie to anotowanie tekstów. Mimo, że nie był tworzony na potrzeby badań nad językami migowymi, jego możliwości są na tyle szerokie, że możliwa jest praca z nim nad anotacją korpusu języka migowego. Jest to aplikacja desktopowa, dostępna na PC, Mac OS X i Linuxa.

Na rysunku **Błąd! Nie można odnaleźć źródła odwołania.** widoczny jest główny ekran aplikacji z otwartą anotacją holenderskiego korpusu języka migowego. Ze względu na to, że ELAN-a nie używa się w polskich pracach badawczych, Autor prezentuje jego wykorzystanie w zagranicznych korpusach. Widoczny jest podział na warstwy anotacji, każda ma inne przeznaczenie. W tym przypadku mamy warstwy z tłumaczeniem na angielski, holenderski, warstwy z glosami prawej i lewej ręki, warstwę mouthingu. Każda warstwa może mieć inny podział na osi czasu. Warstwy mogą mieć ustaloną hierarchię. Można je porządkować, kolorować, co ułatwia organizację pracy anotacyjnej.

Odtwarzanie plików filmowych można kontrolować – oglądając je klatka po klatce, lub z przyspieszeniem albo spowolnieniem.

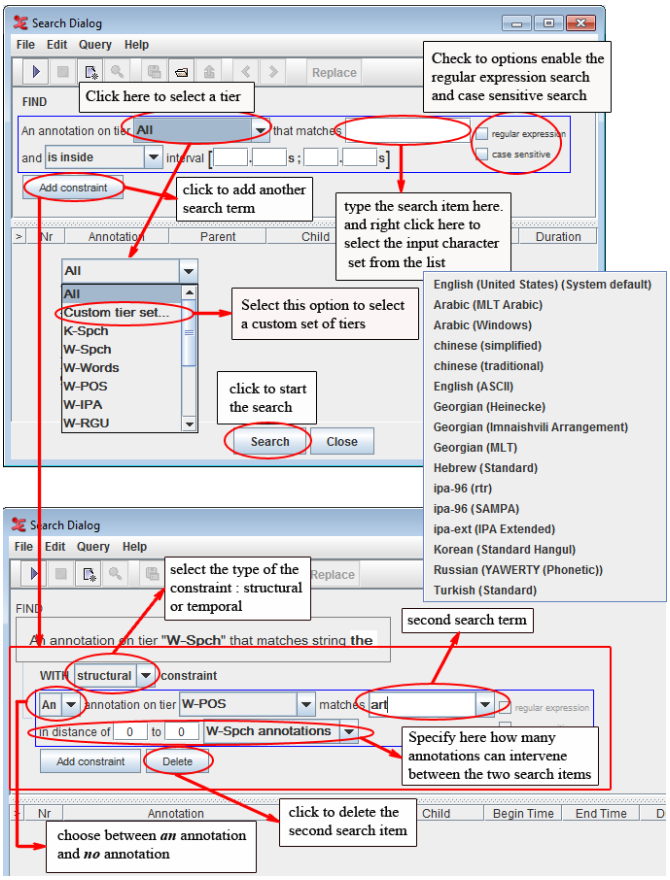


Rysunek 1. Główne okno programu ELAN



Rysunek 2. ELAN w wersji na Mac OS X

W pracy anotacyjnej ważne jest wyszukiwanie zanotowanych tokenów. ELAN udostępnia bogate opcje wyszukiwania, które widać na rysunku Rysunek 3. Okno wyszukiwania w ELAN.



Rysunek 3. Okno wyszukiwania w ELAN. Źródło:

http://www.mpi.nl/corpus/html/elan/ch04.html#Sec_Searching_in_a_single_annotation_file

Program jest wciąż rozwijany. ELAN pracuje na plikach na dysku twardym i tam umieszcza metadane dotyczące ich obróbki. Może on pracować na plikach *.mpg i *.wav. Twórcy programu opracowali format zapisu EAF – ELAN Annotation Format, oparty na XMLu.

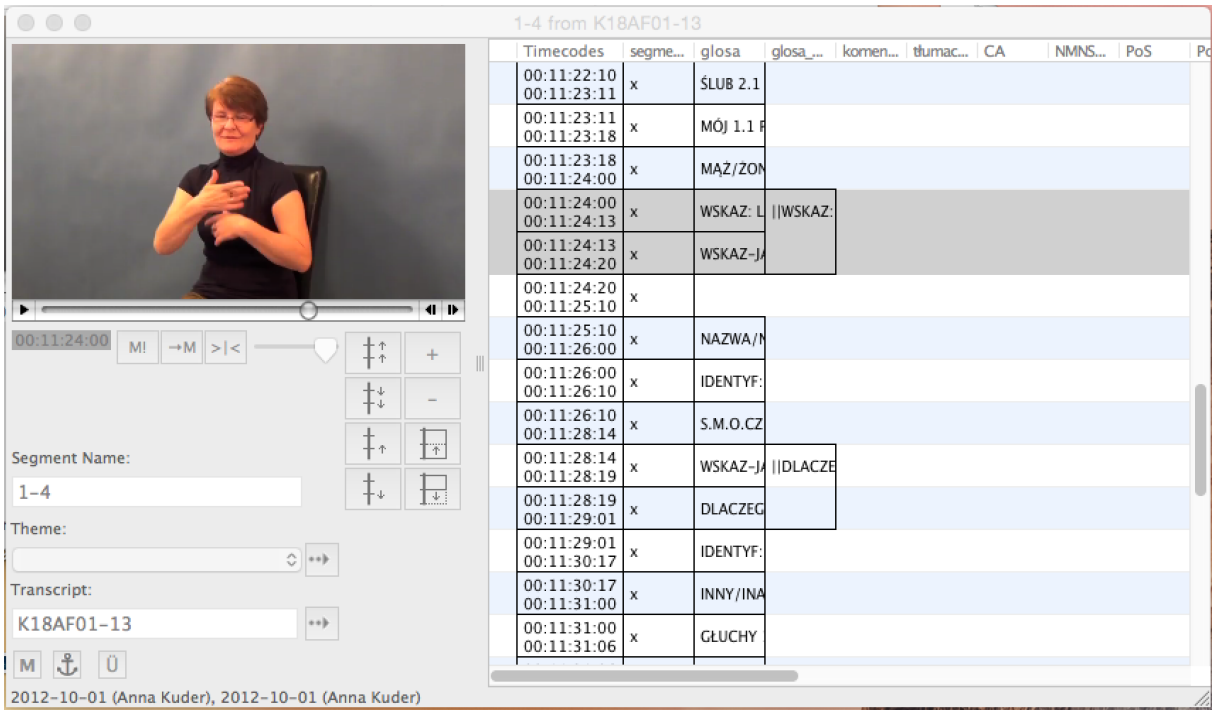
Szczegółowa instrukcja obsługi ELAN [6] została opracowana przez Instytut Maxa Plancka i jest stale aktualizowana.

2.3. iLex

iLex powstał w wyniku wieloletniego projektu badawczego prowadzonego przez Instytut Niemieckiego Języka Migowego i Komunikacji Głuchych w Uniwersytecie w Hamburgu. jako odpowiedź na brak wyspecjalizowanych narzędzi do anotacji korpusu językowego [3]. Twórcy iLexa stwierdzili, że dotychczasowe narzędzia nie spełniają ich wymagań i napisali własne narzędzie. Główną zaletą tego programu jest możliwość równoczesnego dostępu wielu anotatorów do danych umieszczonych na serwerze. Wadą natomiast jest to, że jest to program działający wyłącznie w środowisku Apple. Z dotychczasowych doświadczeń wynika, że możliwe jest uruchomienie iLexa na

maszynie wirtualnej pod środowiskiem Windows, ale jest to rozwiązanie, które generuje więcej kłopotów, niż przynosi pożytku.

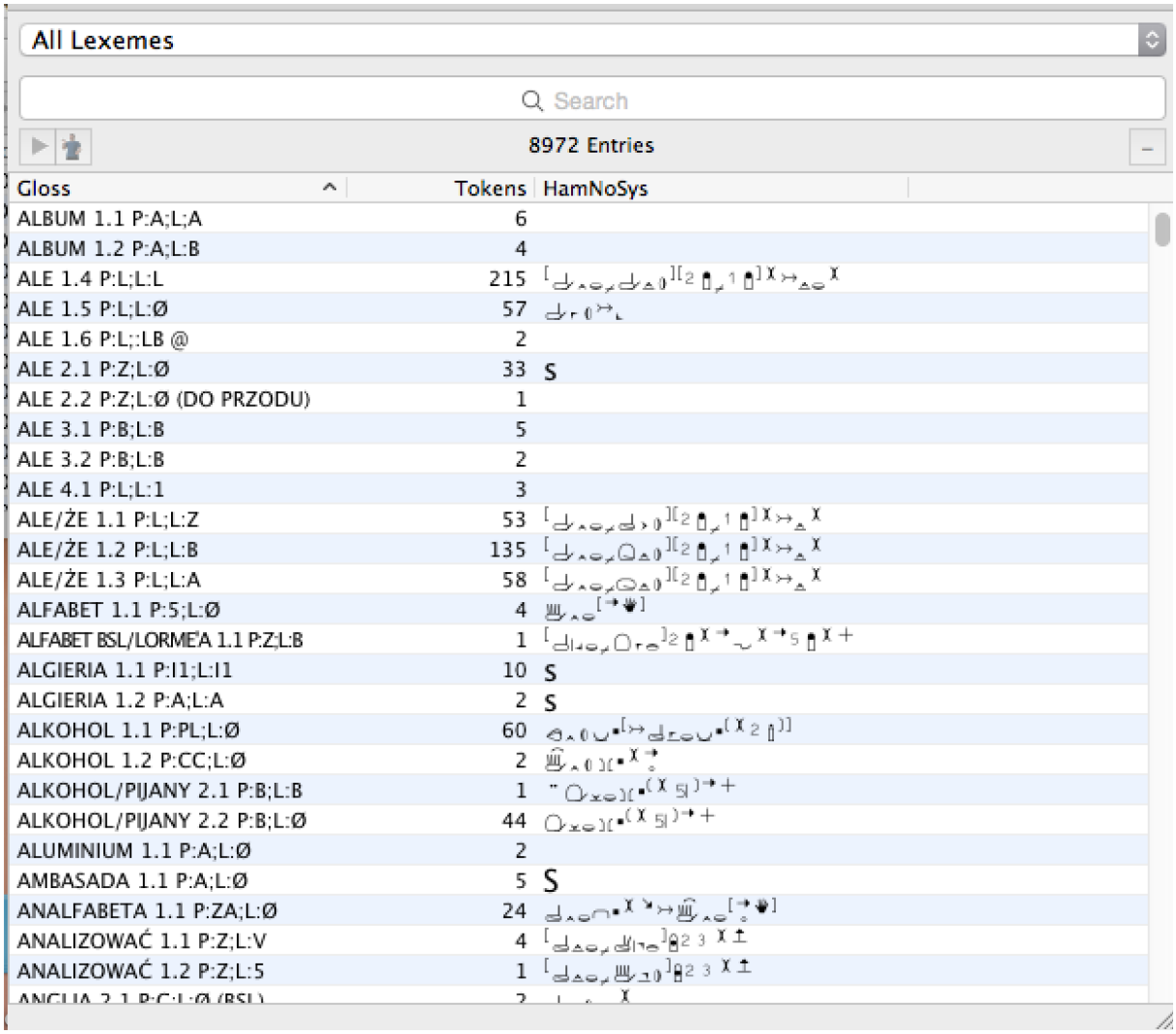
Możliwości narzędzia są podobne, jak w ELAN. Możliwe jest tworzenie warstw i cięcie ich oraz opisywanie. Przykład takiej pracy z iLexem widać na rysunku Rysunek 4. Okno anotacji iLexa. Okno programu opiera się o te same komponenty. Widoczne jest okno z klipem filmowym z osobą migającą, kontrolki do przewijania filmu, przyspieszania i zatrzymywania. Po prawej stronie jest oś czasu podzielona na warstwy i pocięta na tokeny, do których przypisane są metadane.



Rysunek 4. Okno anotacji iLexa

iLex jest wykorzystywany do wszelkich zadań korpusowych, takich jak segmentacja, lematyzacja, tagowanie. Są to między innymi prace związane z obróbką surowych danych filmowych, przypisywaniu ścisłego znaczenia każdemu znakowi w postaci glosy, która służy odróżnieniu danego leksemu od innych, oraz opisywanie szczegółowe glos pod względem gramatycznym.

Interesującą funkcją iLexa jest możliwość zapisu znaków języka migowego w HamNoSys. Jest to hamburski system notacji znaków – Hamburg Sign Language Notation System. Jest dostępny jako czcionka TrueType i możliwy do wyświetlenia na stronie internetowej. HamNoSys jako odpowiednik międzynarodowego alfabetu fonetycznego (IPA). Składa się on z symboli i cyfr i, w zamyśle twórców, jest możliwy w nim zapis znaku z dowolnego języka migowego. Z tych powodów jest używany również w badaniach Polskiego Języka Migowego. Przykładowe zapisy symboli HamNoSys prezentuje Rysunek 5. Widać na nim również listę glos.



Rysunek 5. Okno iLexa z listą glos i zapisem HamNoSys

Dostępna jest opcja odtworzenia znaku języka migowego zapisanego w HamNoSysie za pomocą avatara generowanego przez program SiGML Service Player. Oczywiście animacja nie jest bardzo dokładna, jednak użytkownik języka nie powinien mieć problemu ze zrozumieniem znaku.

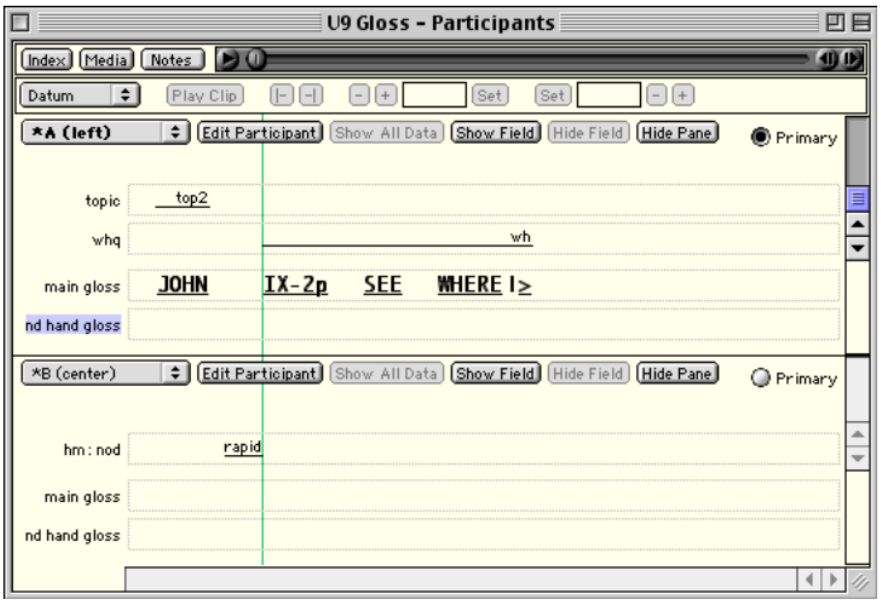
Metadane są przechowywane w bazie Postgres. Pliki filmowe mogą być przechowywane zarówno na komputerze użytkownika jak i w centralnym serwerze, a referencje do nich sa przechowywane w bazie danych [7].

iLex ma możliwość przeszukiwania całej bazy zawierającej dane ze wszystkich filmów, co umożliwia np. tworzenie list frekwencyjnych do słowników, prowadzenie badań statystycznych przez lingwistów, czy wyszukiwanie wystąpień danego znaku migowego w korpusie.

2.4. Sign Stream

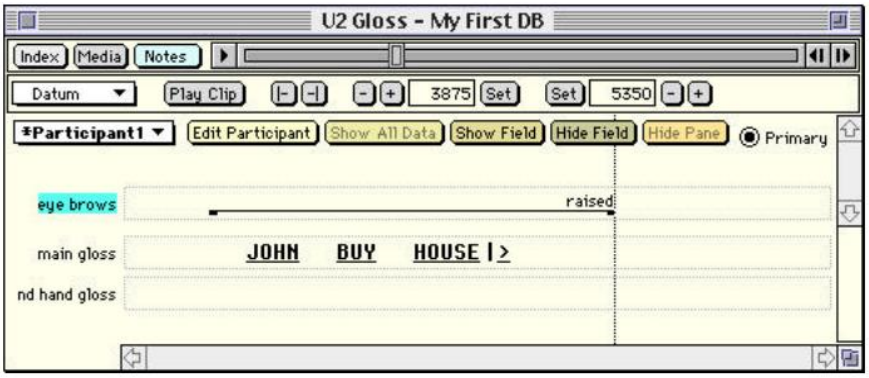
Sign Stream powstał w celu analizy Amerykańskiego Języka Migowego w ramach prac badawczych na Uniwersytecie w Bostonie. Jest on już nierozwijany, ostatnia wersja 2.2 została wydana w 2003 roku w maju. Rozpoczęto prace nad wersją 3, ale nie zostały one dokończone. Pierwszy prototyp powstał w 1994 roku. Jest to program open source, na licencji IBM Common Public License.

Praca w programie wygląda podobnie, pozostałych aplikacjach. Ma on możliwość przypisania do anotacji kilku filmów jednocześnie. Przydaje się to, gdy mamy informatora nagranego z kilku różnych pozycji – np. z przodu i z boku, aby móc dokładnie widzieć i zapisać dane dotyczące ruchu rąk, pozycji ciała czy głowy. Można w ten sposób zanotować klip filmowy, na którym widać dwóch rozmówców. Taki sposób pracy prezentuje Rysunek 6.

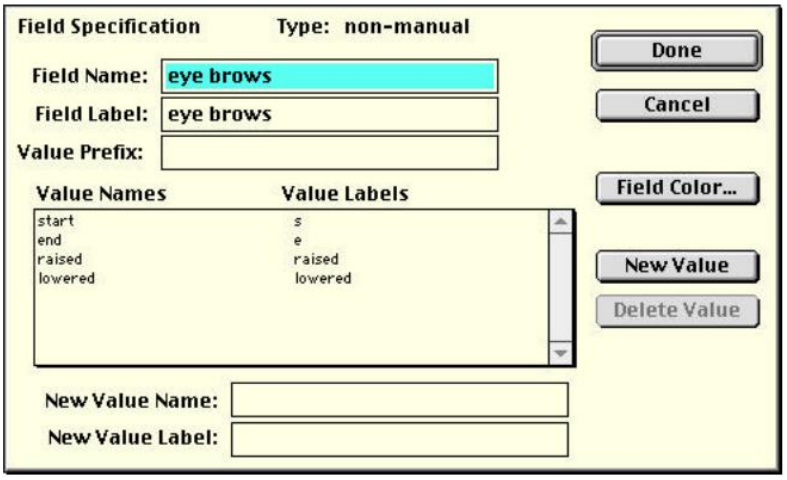


Rysunek 6. Okno anotacji kilku informatorów

Program również ma warstwy do anotacji, ma również osobne menu do ich definiowania. Warstwy mogą być zdefiniowane z zamkniętym bądź otwartym słownikiem wartości. Na przykład warstwa opisująca uniesienie brwi może mieć dwie wartości – uniesione, opuszczone, co reprezentuje np. zdanie twierdzące bądź pytające. Twórcy aplikacji dostarczają z góry zdefiniowane warstwy i wartości i opisują je w swoim przewodniku [8]. Rysunek 7 prezentuje przykładowe warstwy, a Rysunek 8 prezentuje okno definiowania warstwy.



Rysunek 7. Okno warstw Sign Stream



Rysunek 8. Definicja warstwy

Ciekawą funkcjonalnością jest wyszukiwanie, w którym zaimplementowano operatory typu boolean: AND, OR i NOT, oraz operatory temporalne: WITH, BEFORE, AFTER, STARTFRAME, ENDFRAME, FRAMED i UNFRAMED. Można z nich składać wyrażenia przeszukujące zanotowane dane. Operatory boolowskie są standardowymi operatorami działającymi na operandach typu boolean, albo wyrażeniach których wynikiem jest wartość typu boolean. Operatory temporalne działają na operandach temporalnych, to znaczy takich, które mają zapisane w sobie informacje dotyczące czasu rozpoczęcia. Przykładowo, operator BEFORE zwraca true, jeśli końcowa klatka pierwszej glosy jest przed startową klatką drugiej glosy, w przeciwnym wypadku zwraca false, a STARTFRAME zwraca klatkę startową glosy. Pozwala to na elastyczne układanie zapytań korpusowych i wyszukiwanie interesujących wyników.

```
PARTICIPANT *[eye brows[raised OR lowered]]
```

Przykładowe zapytanie wyszukuje wszystkie glosy, w których informator jest dowolny i glosy są opisane jako leżące na warstwie opisującej uniesienie brwi i mają wartość `raised` lub `lowered`.

```
primary PARTICIPANT *[ eye brows[raised] BEFORE GLOSS-FIELD[*]]
```

To zapytanie wyszukuje te glosy, w których dowolny informator oznaczony jako `primary`, ma przypisane glosy na warstwie opisującej uniesienie brwi z wartością `raised` i występujące przed dowolną glosą. Przy prowadzeniu badań statystycznych jest to bardzo pomocne narzędzie.

2.5. Wady i zalety dostępnych narzędzi

Istniejące narzędzia nie wykorzystują zalet nowoczesnej sieci i pracy w chmurze. Możliwe jest przechowywanie danych na zewnętrznych serwerach i dostęp do nich poprzez zwykłe aplikacje przeglądarkowe. Komfort takiej pracy nie ustępuje w niczym aplikacjom instalowanym na komputerach. Dodatkowe możliwości, jakie oferuje chmura, są nie do przecenienia. Są to równoczesny dostęp do danych, współdzielenie materiałów, możliwość tworzenia API dostępowych na potrzeby tworzenia różnych frontowych aplikacji mających dostęp do tych samych danych.

2.5.1 ELAN

Podstawową wadą programu ELAN jest nie sieciowy tryb pracy, co sprawia, że praca nad tak dużym przedsięwzięciem, jakim jest korpus PJM, jest utrudniona. Wymagane są dodatkowe narzędzia do zarządzania danymi, np. wspólny serwer FTP, bądź inne repozytorium danych. Dużą zaletą programu jest to, że jest on na licencji GNU Public License i ma udostępniony kod źródłowy.

2.5.2 iLex

Z wywiadów przeprowadzonych w Pracowni Lingwistyki Migowej wynika, że iLex jest wygodnym narzędziem, ale ma wady utrudniające pracę. Niemożliwa jest elastyczna i szybka współpraca przy anotacji. W razie potrzeby konsultacji danego znaku nie da się w łatwy i prosty sposób go pokazać osobie, która zdalnie pracuje w innym miejscu. Aby to zrobić, należy podać dokładną nazwę filmu, segment, i czas, aby współpracownik mógł sam odszukać znak i pomóc znaleźć jego znaczenie. Aby przyspieszyć tego rodzaju zadanie, wystarczyłoby spreparować odpowiedni link z danymi znaku zaszytymi w URLu, który program mógłby otworzyć.

Ponieważ praca nad korpusem jest wspólną praca wielu osób, zachodzi potrzeba sprawdzenia historii edycji danej glosy. W chwili obecnej jest możliwość sprawdzenia tylko ostatniej edycji.

Niestety iLex jest zamkniętym oprogramowaniem i nie udostępnia możliwości pisania własnych wtyczek i komponentów, co by było z pewnością przydatną opcją dla rozwijających się instytutów badawczych chcących wykorzystać oprogramowanie do własnych specyficznych celów.

2.5.3 Sign Stream

Program jest już starszą aplikacją i nie jest już rozwijany. Nie korzysta on z możliwości sieci. Pracując z nim trzeba uważać na lokalizacje przechowywanych plików, gdyż w danych anotacyjnych są zapisywane odnośniki do nich. Po przeniesieniu klipu filmowego w inne miejsce mogą wystąpić problemy. Program jest dopracowany jeśli chodzi o możliwości badawcze, a na szczególną uwagę zasługuje tu opracowany język zapytań.

3. Propozycja systemu do anotowania korpusu języka migowego

Analiza wad i zalet istniejących aplikacji podpowiada nam, jakie są obszary, które można poprawić, a które należy zachować. Wynika z niej, że istotny jest swobodny dostęp do aplikacji w ramach różnych platform, możliwość rozszerzenia aplikacji o dodatkowe funkcje i oraz sposoby przechowywania danych, aby móc przechowywać dane w lokalnej bazie bądź w chmurze. Realizacja takich wymagań powoduje konieczność odpowiedniego zaprojektowania systemu.

3.1. System jako aplikacja internetowa

Projektowany system jest pomyślany jako aplikacja internetowa. W niniejszym podrozdziale Autor przedstawia podstawowe informacje na temat aplikacji internetowych, ich architektur, oraz prezentuje koncepcje architektoniczne wykorzystywane w pracy.

3.1.1 Aplikacje internetowe w szerszym ujęciu

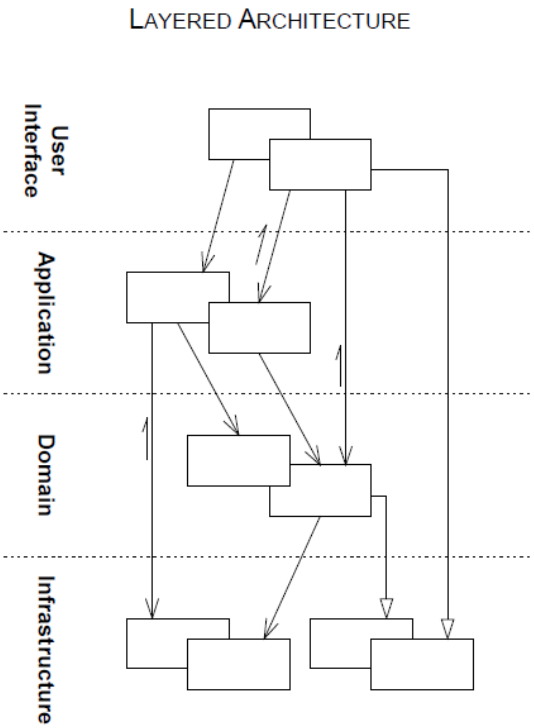
Pierwsze strony internetowe były statycznymi plikami HTML udostępnianymi przez serwer http. Dostęp do nich był szybki, lecz w przypadku zmiany danych opisywanych przez pliki, niezbędna była ich modyfikacja. Dopiero w 1993 roku pojawiła się koncepcja automatycznego generowania dokumentów przez serwer. Zakłada ona, że w odpowiedzi na żądanie http serwer generuje odpowiedź w sposób dynamiczny. Końcowo upowszechnił się standard CGI – Common Gateway Interface – za pośrednictwem którego serwery serwowały treści generowane przez dowolne języki, jak np. Perl, PHP, C, czy nawet język powłoki - bash.

Dynamiczne generowanie dokumentów HTML i uruchamianie na żądanie przez serwer http programów renderujących odpowiedź było głównym sposobem działania serwerów http. Skierowało to rozwój technologii internetowych w stronę aplikacji internetowych. Zaczęto konstruować serwisy internetowe składające się z programów komunikujących się ze sobą. Programy te działają w środowisku uruchomieniowym, zwanym serwerem aplikacji. Może być on częścią serwera http lub być z nim powiązanym. Przykładem może być Tomcat, JBoss w przypadku Javy. Programy mogą korzystać z zewnętrznych usług, dostarczanych przez odpowiednie oprogramowanie albo system operacyjny. Ten rodzaj technologii jest stosowany dziś praktycznie wszędzie. Wśród zastosowań można znaleźć np. systemy rezerwacji biletów lotniczych, sklepy internetowe, aukcje, systemy bankowe, czy fora internetowe.

3.1.1 Architektury aplikacji internetowych

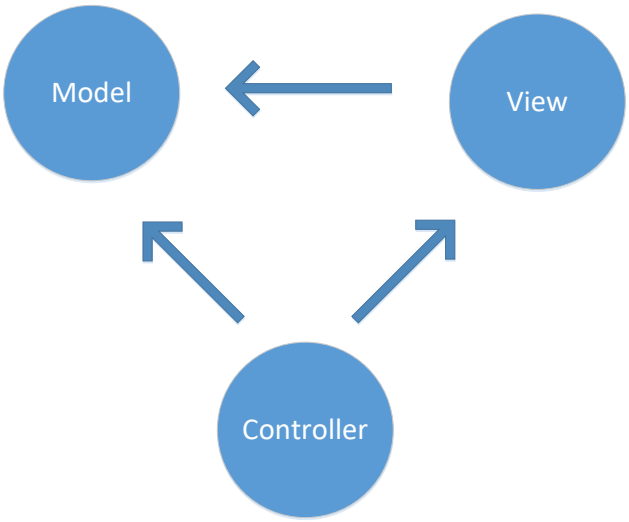
Eric Evans w swojej książce „Domain-Driven Design” stwierdza, że „istnieje wiele sposobów podziału systemu informatycznego”, a także, że „najbardziej skuteczne architektury używają pewnych

odmian następujących czterech warstw pojęciowych: Warstwa interfejsu użytkownika, warstwa aplikacji, warstwa dziedziny, warstwa infrastruktury" [9]. Przestrzega on przed mieszaniem kodu związanego z dostępem do danych z kodem logiki biznesowej. Taki sposób tworzenia oprogramowania w krótkiej perspektywie daje szybkie wyniki, lecz koszty jego utrzymania rosną bardzo szybko. Rozwiązaniem jest separacja odpowiedzialności poszczególnych elementów systemu pod względem funkcjonalności. Odpowiednie zaprojektowanie każdej warstwy i luźne jej powiązanie z innymi ułatwia utrzymanie kodu. Przyjmuje się zasadę, że „dowolny element w danej warstwie jest zależny jedynie od innych elementów w tej samej warstwie, lub w warstwach niższych".



Rysunek 9. Architektura warstwowa. Źródło: Evans, E. (2004)

Najpopularniejszą architekturą stosowaną dla aplikacji internetowych jest architektura Model-View-Controller. Umożliwia ona separację modelu od widoku i od kontrolera, co ułatwia rozwijanie kodu. Jednak warstwa modelu może być również bardzo złożona i należy ją starannie zaprojektować. Zasada separacji odpowiedzialności powinna się odnosić również do obiektów i klas, które w niej są utrzymywane [10].

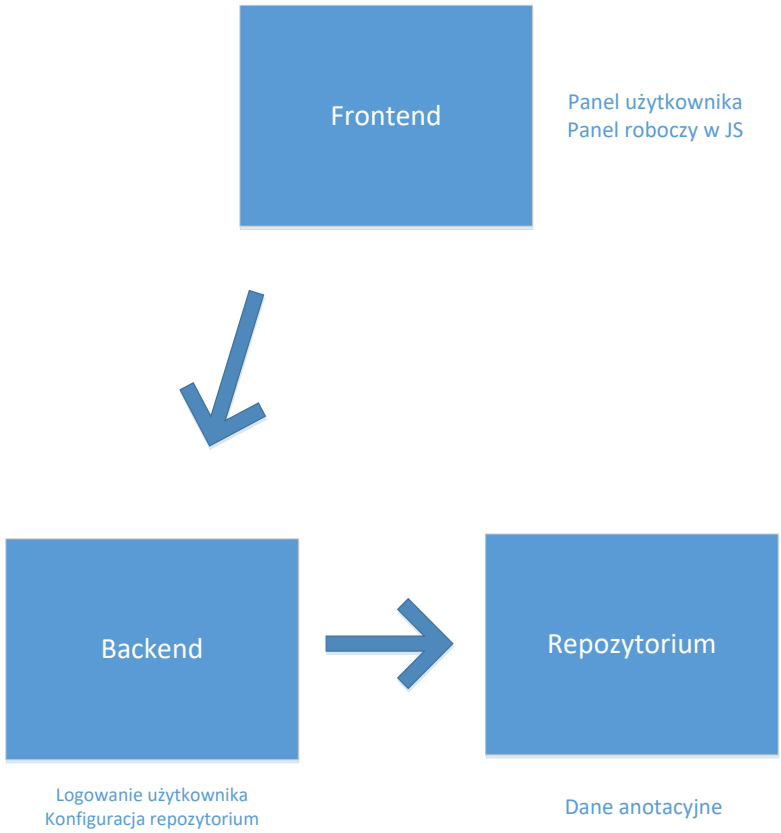


Rysunek 10. Architektura MVC

Sposobów podejścia do architektury jest kilka. System informatyczny może być zaprojektowany jako zestaw mikrousług, może mieć architekturę heksagonalną, warstwową z warstwą domeny. Wszystkie implementacje wiążą się z początkowym nakładem większej ilości pracy ze względu na potrzebę zakodowania odpowiednich interfejsów i mechanizmów. Autor pracy po przeanalizowaniu wymagań i istniejącego oprogramowania stwierdził, że projektowany system w istocie nie posiada zbyt rozbudowanej logiki biznesowej. W rzeczywistości jest to pewien zbiór metadanych niebiorących udziału w procesach biznesowych, czy wpływających w specyficzny sposób na siebie wzajemnie. Implementowanie architektury zorientowanej na budowę dużego systemu informatycznego byłoby tu bezcelowe. Jednak z uwagi na to, że zamierzeniem Autora jest zaprojektowanie systemu elastycznego, należałoby umożliwić przyszłe jego włączenie w większy projekt. Można tu zaczerpnąć z idei architektury zorientowanej na usługi, której jedną z głównych cech jest to, że komponenty są ze sobą luźno powiązane.

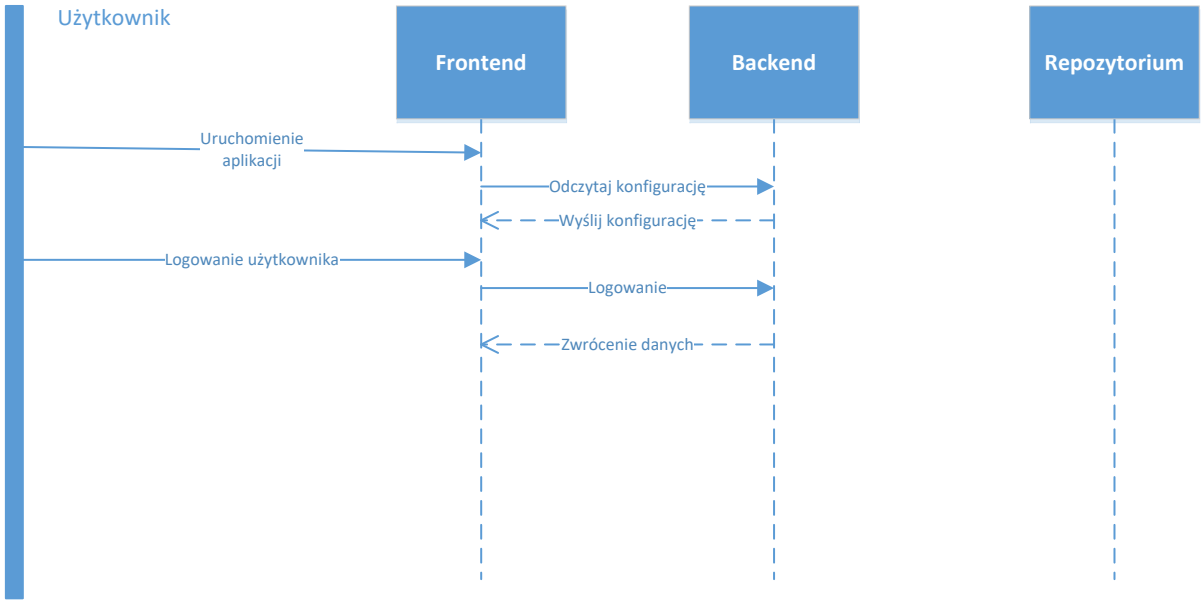
3.1.2 Proponowana architektura prototypu

Rysunek 11 przedstawia architekturę prototypu zakładającą elastyczność wyboru repozytorium do przechowywania danych. Aplikacja podzielona jest na trzy komponenty – Frontend, Backend i Repozytorium. Frontend składa się z aplikacji klienckiej reagującej na działania użytkownika. Część składa się z widoków serwowanych przez backend składających się na panel użytkownika i panel logowania, część jest aplikacją JavaScript, wykonującą to, do czego system jest przeznaczony – z jej pomocą użytkownik anotuje klipy filmowe. Backend powinien zawierać konfigurację Repozytorium. Repozytorium może być modułem w ramach Backendu, może być też zupełnie odrębną aplikacją, leżącą na innym serwerze.

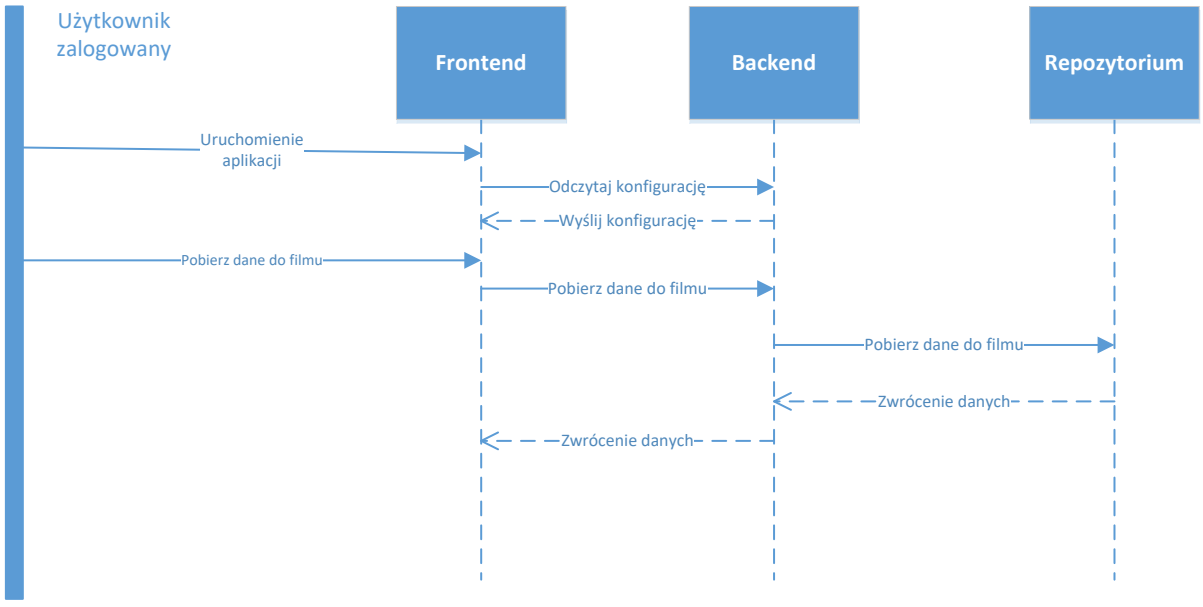


Rysunek 11. Architektura prototypu

Backend odpowiada na żądania wysyłane przez Frontend. Może to być żądanie zalogowania się, pobrania danych z panelu użytkownika, aktualizacja tych danych. Backend ma dostęp do repozytorium, i w razie potrzeby może się do nich odwołać. Wynika z tego możliwość rozszerzenia funkcji Backendu w miarę rozwoju aplikacji. Mogą to być np. metody do wyliczania danych statystycznych na podstawie danych z Repozytorium, które ostatecznie wyświetli Frontend. Rysunek 12 przedstawia akcję logowania się użytkownika. Backend jest pośrednikiem pomiędzy Frontendem a Repozytorium. Aplikacja JavaScript poprzez Backend pobiera dane i je zapisuje. Konfiguracja backendu wskazuje, gdzie jest Repozytorium. Proces odczytu danych prezentuje Rysunek 13.



Rysunek 12. Akcja logowania użytkownika



Rysunek 13. Proces odczytu danych z Repozytorium

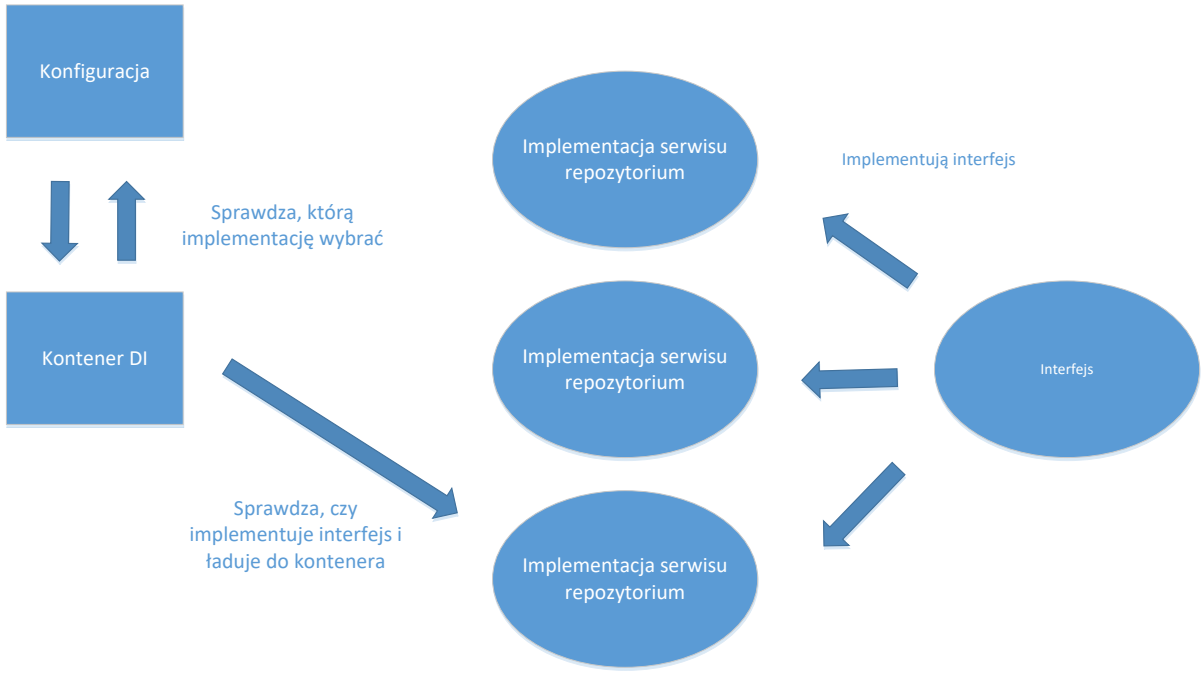
Wraz z rozwojem technologii internetowych kładziony jest duży nacisk na przetrzymywanie danych w chmurze, dostęp do nich przez przeglądarkę internetową, odciążanie klienta od potrzeby instalowania dodatkowych aplikacji. Za przykład mogą tu posłużyć aplikacje w wersji online z pakietu MS Office, jak np. Word Online, Excel Online. Przeniesienie ich do chmury sprawiło, że otworzyły się

dodatkowe możliwości współdzielenia pracy, jednoczesnego edytowania plików, etc. Są to możliwości, które można wykorzystać również w badaniach naukowych. Proponowana architektura sprawia, że można swobodnie wymienić sposób przechowywania danych. Wystarczy zapewnić odpowiednią wymienialność serwisów w Repozytorium.

3.1.3 Wstrzykiwanie zależności jako sposób na wymienialność serwisów

Nowoczesne techniki programowania obiektowego umożliwiają zrobienie wymienialności serwisów w prosty sposób. Aby rozluźnić powiązania obiektów stosuje się paradygmat Inversion of Control – odwrócenie sterowania. Wstrzykiwanie zależności jest jedną z możliwych implementacji tego wzorca [11]. Trzeba określić, jakie serwisy chcemy wymieniać i jakie powinny mieć metody.

Wymienialność obiektów można zaimplementować za pomocą kontenera Dependency Injection. Wystarczy uzależnić sposób ładowania danych do kontenera od konfiguracji aplikacji, np. można ustawić w niej, jaka klasa ma być zarejestrowana w kontenerze. Zmiana tej konfiguracji spowoduje, że pod tym samym identyfikatorem będzie obiekt innej klasy. Wykorzystywanie metod tej klasy nie będzie odbywać się bezpośrednio, tylko poprzez kontener, podmiana będzie więc zupełnie przezroczysta. Warunkiem jest tylko to, żeby wszystkie klasy ładowane za pomocą konfiguracji przez kontener miały te same metody. Aby to zrobić, kontener powinien sprawdzać, czy ładowana klasa implementuje zadany interfejs. W ten sposób uzyskamy pełną wymienialność serwisów repozytorium. Ten mechanizm obrazuje Rysunek 14.



Rysunek 14. Mechanizm wymienialności repozytorium

Jak wcześniej wspomniano, logika biznesowa nie jest zbyt rozbudowana. Odpowiedzialność repozytorium to zwykłe operacje zapisu i pobrania danych, więc interfejs powinien zakładać operacje typu CRUD.

3.1.4 Organizacja Frontendu

W skład frontendu wchodzi panel użytkownika oraz aplikacja umożliwiająca anotowanie filmu. Panel użytkownika powinien służyć do wyświetlania statystyk, umożliwiać logowanie się i być miejscem udostępniającym dodatkowe funkcjonalności. Autor zdecydował, że w prototypie zostanie zaimplementowane jedynie okno logowania oraz widok anotacji filmu. Opis ich implementacji wyczerpie tematykę związaną z projektowaniem i możliwościami implementacji panelu.

Aby zapewnić stopień interakcji aplikacji do anotacji porównywalny z aplikacjami okienkowymi, powinna być napisana w języku umożliwiającym dynamiczne generowanie formularzy i reagowanie na zdarzenia. Takim językiem jest JavaScript, w którym dostępne jest bardzo dużo bibliotek usprawniających pracę programisty. Aplikacja powinna umożliwić wyświetlenie filmu i udostępnić interfejs do edycji potrzebnych danych. Autor projektując interfejs wzorował się na aplikacjach przedstawionych w rozdziale 2. Wynika z nich, że w celach anotacji niezbędne są komponenty:

- Odtwarzacz klipów filmowych
- Kontrolki do odtwarzacza filmowego
- Moduł wyświetlający zanotowane dane
- Moduł do wpisywania anotowanych danych

iLex i ELAN różnią się między sobą pod względem modułu do wyświetlania danych i wpisywania ich. ELAN ma je rozdzielone, a iLex pozwala na edycję w ramach modułu wyświetlającego dane. Autor zdecydował, że interfejs będzie zbliżony do interfejsu iLexa.

W aplikacji do anotacji powinna być możliwość zmiany rodzaju anotowanych danych i sposobu ich uzupełniania. Umożliwi to zmianę działania aplikacji tak, aby można było dzięki niej tworzyć napisy do filmów. Korzystając z wzorca Rejestr można zaimplementować taką funkcjonalność. W rzeczywistości byłoby to rozwiązanie podobne do wykorzystania kontenera Dependency Injection. Na potrzeby aplikacji trzeba opracować szablon pluginu i zaimplementować moduł do zarządzania pluginami.

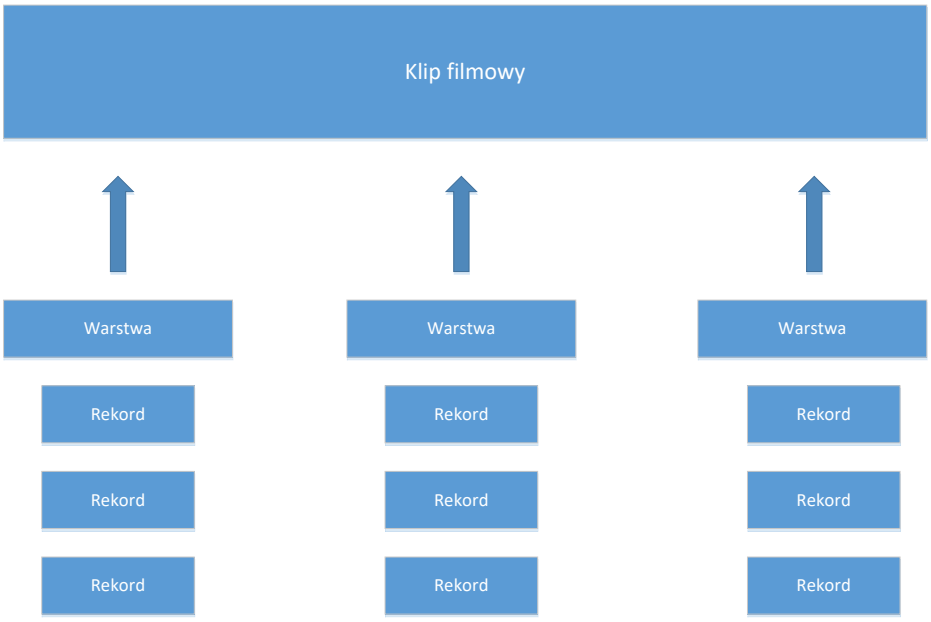
3.2. Przechowywanie danych

W przypadku zapisywania danych i ich odczytu operacje są proste i sprowadzają się właściwie do metod CRUD. W przypadku potrzeby zaimplementowania przetwarzania danych, np. obliczania statystyk, należy zapewnić możliwość pobrania wszystkich odpowiednich danych. Tego typu operacje już nie są logiką Repozytorium, tylko Backendu.

W przypadku baz danych bardzo łatwo jest zrobić przeliczanie odpowiednich danych po stronie silnika bazy danych, jednak jeśli chcemy zapewnić wymienialność repozytoriów, to nie możemy liczyć na to, że funkcjonalności udostępniane przez bazy danych będą dostępne wszędzie. Repozytorium danych powinno spełniać jedynie podstawowe funkcje, tak, aby można było je łatwo wymienić.

Za przykład mogą posłużyć agregacje w relacyjnych bazach danych, czyli funkcje typu `sum`, `max`. Agregacje takie mogą nie być dostępne np. w nierelacyjnych bazach, czy chociażby w przypadku plikowego przechowywania danych.

Analiza aplikacji wskazuje, że podstawową jednostkę danych można określić jako rekord zawierający datę rozpoczęcia, datę zakończenia, metadane, oraz przypisanie do warstwy. Warstwa jest kontenerem na rekordy określonego typu, definiowanego przez użytkownika. Tworzy się ją dla określonego klipu filmowego i nadaje jej pewną nazwę, dzięki której można określić, czemu ma służyć. Klip może mieć zdefiniowanych wiele warstw.



Rysunek 15. Organizacja danych w repozytorium

Autor nie narzuca tu konieczności definiowania wielu warstw, przypisywania im jakichś specyficznych danych. Jedyne, co warstwa posiada, to przypisanie do filmu, dla którego została utworzona, plugin, przez który została utworzona, oraz przypisanie do użytkownika – właściciela.

4. Opis narzędzi i technologii zastosowanych w pracy

W niniejszym rozdziale Autor przedstawia i opisuje technologie użyte podczas projektowania i implementacji prototypu. Ponieważ prototyp został zaprojektowany jako aplikacja internetowa, w większości są to technologie internetowe.

4.1. HTML5

HTML jest to język znaczników wywodzący się z uogólnionego języka SGML pozwalającego zapisywać różne dane w formacie tekstowym, dzięki czemu ułatwione było ich przenoszenie, wyświetlanie i drukowanie. Opisuje on strukturę danych zawartych na stronie internetowych, osadza na niej obiekty, takie jak tabele, linki, czy multimedia. Dzięki znacznikom przeglądarki internetowe potrafią wyświetlić stronę zgodnie z zamierzeniami twórcy. Nie jest to język programowania, gdyż nie posiada konstrukcji logicznych, instrukcji warunkowych, pętli, etc.

W miarę czasu rozwijano HTMLa dodając do niego lepszą obsługę błędów, nowe znaczniki, standaryzowano i doprecyzowano wiele niejasności w specyfikacji, co zaowocowało powstaniem wersji 5, nazywanej po prostu HTML5. Dodano do niego szereg funkcji, niedostępnych we wcześniejszych wersjach, a dających szerokie możliwości twórcom stron internetowych. Są to np. geolokalizacja, rysowanie obiektów 2D na elemencie canvas, dodano obsługę OpenGL – WebGL, API do metod dotyczących przeciągania i upuszczania, HistoryApi, Web Sockets, oraz, co w kontekście niniejszej pracy jest najważniejsze, API do odtwarzania plików audio i video [12]. W poprzednich wersjach korzystano z niezależnych wtyczek i aplikacji opartych o język Flash bądź aplety Javy. W chwili obecnej jest to możliwe z poziomu samego HTML5 i nie ma potrzeby doinstalowywania dodatkowego oprogramowania. Jest to ogromne ułatwienie dla twórców stron, którzy musieli liczyć się z tym, że ich strona może nie działać wszędzie poprawnie. Do dzisiaj są z tym problemy, ale wprowadzenie HTML5 i jednolitej obsługi błędów znacznie poprawiło tę sytuację.

4.2. JavaScript (ECMAScript)

JavaScript jest językiem wykonywanym po stronie klienta. Jest to język skryptowy, interpretowany. Zdobył ogromną popularność na stronach internetowych. Stworzony został przez firmę Netscape w latach 90 XX wieku. Bezpośrednio z niego wywodzi się ECMAScript będący jego następcą.

JavaScript umożliwia przeglądarkom internetowym manipulowanie stroną internetową, tworzenie ciasteczek, reagowanie na zdarzenia dokonywane na stronie [13]. Jedną z cech języka odróżniającą go od innych jest prototypowe dziedziczenie.

Osadzanie skryptów JavaScript odbywa się za pomocą znaczników `<script type=”text/javascript”></script>`. Mogą być one zawarte w treści takich znaczników, bądź w znacznikach mogą być odnośniki do wydzielonych plików z kodem. Przeglądarka internetowa ładując stronę internetową interpretuje zawartość tych znaczników i wykonuje kod JavaScript.

Przykład skryptu JavaScript zawartego w znacznikach jest widoczny na Listingu Listing 1. Przykład osadzenia skryptu JS

```
<html>
      <head></head>
      <body>
            <script type=”text/javascript”>
                  var today = new Dateime(),
                      year = today.getFullYear();

                  alert(‘Mamy dziś rok ’ + year);

            </script>
      </body>
</html>
```

Listing 1. Przykład osadzenia skryptu JS

Ten sam skrypt może być osadzony w oddzielnym pliku o rozszerzeniu, co prezentuje Listing 2.

```
<html>
      <head></head>
      <body>
            <script type=”text/javascript” src=”script.js”></script>
      </body>
</html>
```

Listing 2. Osadzenie skryptu JS za pomocą atrybutu src

4.3. jQuery

W miarę upowszechniania się Javascriptu powstało wiele bibliotek wspierających pracę z nim. Jedną z najpopularniejszych jest jQuery. Jest to biblioteka ułatwiająca trawersowanie po drzewie DOM, jego modyfikację, dodawanie animacji do elementów i modyfikowanie styli [14]. Udostępnia prosty interfejs do wywoływania zapytań AJAX.

4.4. AMD i RequireJS

Często budując zaawansowane systemy musimy mieć na uwadze zależności potrzebne do działania poszczególnych modułów systemu. Ręczne zarządzanie zależnościami jest niewygodne i mało odporne na błędy. Wystarczy sobie wyobrazić aplikację składającą się z kilkunastu lub kilkudziesięciu plików JavaScript, z których każdy musi być załadowany w odpowiedniej kolejności. Zależności występujące między poszczególnymi skryptami mogą być duże i mogą utrudniać programiście zorientowanie się w kodzie i jego rozwijanie [15].

Aby ułatwić zarządzanie kodem stworzono AMD API – Asynchronous Module Definition API, który definiuje w jaki sposób powinny być pisane moduły i ich zależności, aby mogły być ładowane asynchronicznie. Dzięki takiej architekturze możliwe jest ładowanie poszczególnych modułów wraz z ich zależnościami. Z architektury takiej korzysta np. DOJO i RequireJS. Specyfikacja AMD określa sposób definicji modułów zaprezentowany na Listingu Listing 3. Definicja modułu AMD

```
define(id?, dependencies?, factory);
```

Listing 3. Definicja modułu AMD

Polecenie `define` określa moduł o podanym `id`, podanych zależnościach, oraz podaje przepis, funkcję, która się uruchomi przy wywołaniu modułu. Dwa pierwsze parametry są opcjonalne. Określają one kolejno identyfikator modułu, który musi być unikalny, oraz zależności modułu, które są identyfikatorami innych modułów. Identyfikator może być typu „top-level”, lub być nazwą absolutną, nie relatywną. `factory` to funkcja wykonywana przy inicjalizacji modułu, lub obiekt zwracany przez moduł.

Na Listingu Listing 4 widoczna jest deklaracja modułu o identyfikatorze `lib/Car` zwracającego obiekt `that` zawierającego dwie metody.

```
define('lib/Car', [], function() {

     var that = {};

     tank = function() {
          alert('tankuję');
     };
     that.tank = tank;

     drive = function() {
          alert('jadę');
     }

     return that;
});
```

Listing 4. Definicja przykładowego modułu w AMD

Taki moduł jest już odpowiednio zarejestrowany i może być wykorzystywany przez inne moduły. Zwraca obiekt `that` zawierający dwie metody. Jego wykorzystanie prezentuje Listing 5, w którym zawarta jest deklaracja modułu z zależnością od `lib/Car` wykorzystującego jego metody:

```
define(['lib/Car'], function(car) {
     car.tank();
     car.drive();
});
```

Listing 5. Wykorzystanie zadekladowanego modułu AMD

Fabryka zwraca obiekt `car`, który przechowuje referencję do obiektu `that`, zdefiniowanego w pierwszym module. RequireJS zapewnia nam, że wywołanie drugiego modułu spowoduje automatyczne ściągnięcie przez przeglądarkę potrzebnego modułu.

Wykorzystanie RequireJS w aplikacji sprowadza się do dołączenia biblioteki w kodzie HTML i wystartowaniu aplikacji, która ładuje odpowiednie moduły. Przykład jest widoczny na Listingu Listing 6.

```
requirejs.config({
    baseUrl: 'js/app/',
    paths: {
        lib: '../lib',
        jquery: '../jquery'

    },
    urlArgs: "bust=" + (new Date()).getTime()
});

requirejs(['main']);
```

Listing 6. Bootstrap aplikacji z RequireJS

W podanym kodzie widzimy konfigurację biblioteki RequireJS, w której ustawiany jest URL bazowy dla wszystkich ładowanych bibliotek. Dzięki temu aplikacja wie, gdzie ma szukać modułów do załadowania. W parametrze `paths` stawiane są ścieżki do zewnętrznych bibliotek, oraz do biblioteki jquery. Ponieważ przeglądarki cache'ują odpowiedzi z serwera, dodany został parametr do URLa ustawiający przy każdym module dodatkowy timestamp, aby wymusić każdorazowe załadowanie modułu ze strony internetowej. Jest to funkcja przydatna w trakcie developmentu, która jednak powinna być wyłączona na etapie produkcji. Aplikacja startuje w momencie dołączenia modułu `main`. Biblioteka szuka odpowiedniego pliku `main.js` w katalogu `js/app`. Jeśli w pliku `main.js` został zdefiniowany moduł z odpowiednimi zależnościami, to one również zostaną ściągnięte i kod w nich zostanie wykonany.

4.5. JSON – JavaScript Object Notation

Jest to tekstowy format danych, zdefiniowany w oparciu o podzbiór języka JavaScript, chociaż w chwili obecnej jest on wykorzystywany przez wiele języków programowania, posiadających odpowiednie moduły do jego interpretowania. Jest bardzo dobrym formatem służącym do wymiany danych. Jest to format tekstowy, podobnie jak XML, jednak zajmuje mniej miejsca ze względu na brak znaczników. W języku JavaScript ma on postać tablicy asocjacyjnej, której klucze są łańcuchami typu string otoczonymi cudzysłowami, a wartości mogą być typami prostymi, tablicami albo obiektami. Struktura ta jest intuicyjna w odczycie i jest bardzo wygodna do zapytań AJAXowych.

```
{
      „car”: „Toyota”,
      „production_year”: 2008,
      „color”: „blue”,
      „broken”: false
}
```

Listing 7. Przykład struktury JSON

4.6. AJAX

AJAX jest to skrót od *Asynchronous JavaScript and XML*. Jest to metoda tworzenia aplikacji opierających się na dynamicznej komunikacji z serwerem za pośrednictwem wywołań po stronie klienta. Pozwala to na nieprzeładowywanie strony w trakcie użytkowania jej. Potrzebne dane mogą być załadowane na żądanie. Obsługą asynchronicznych wywołań zajmuje się klasa XMLHttpRequest, która nie jest zbyt wygodna w użyciu, dlatego powstało wiele bibliotek, które implementują tę funkcjonalność w sposób łatwy do wykorzystania przez programistę. Są to np. prototype, jQuery, AdvancedAJAX.

4.7. Baza SQL

Relacyjne bazy danych są podstawą działania wielu systemów i aplikacji. Przechowują dane w formie relacji, umożliwiając łatwe ich przeglądanie, wyszukiwanie i zarządzanie nimi. Dane mogą być w formie liczbowej, tekstowej, binarnej, mogą być przechowywane w formie LOB-ów (pliki graficzne, dokumenty) w formie JSON, itd. Bazy potrafią dane przeszukiwać, sortować. Dostęp do danych może być ograniczony w zależności od potrzeby. Bazy można skalować. Bazy zapewniają też szereg mechanizmów, które są nie bez znaczenia dla biznesu. Mowa tu o transakcyjności oraz dbaniu o spójność danych. Te cechy sprawiły, że relacyjne bazy danych stały się bardzo powszechne i powstało wiele systemów zarządzania bazami danych. Najpopularniejsze z nich to MySQL, MS-SQL, Postgresql, Oracle Database

Dostęp do danych odbywa się za pośrednictwem języka SQL – Structured Data Language. Pozwala on na modyfikację zawartości bazy, odczytywanie jej i kasowanie.

4.8. Nierelacyjna baza danych - Redis

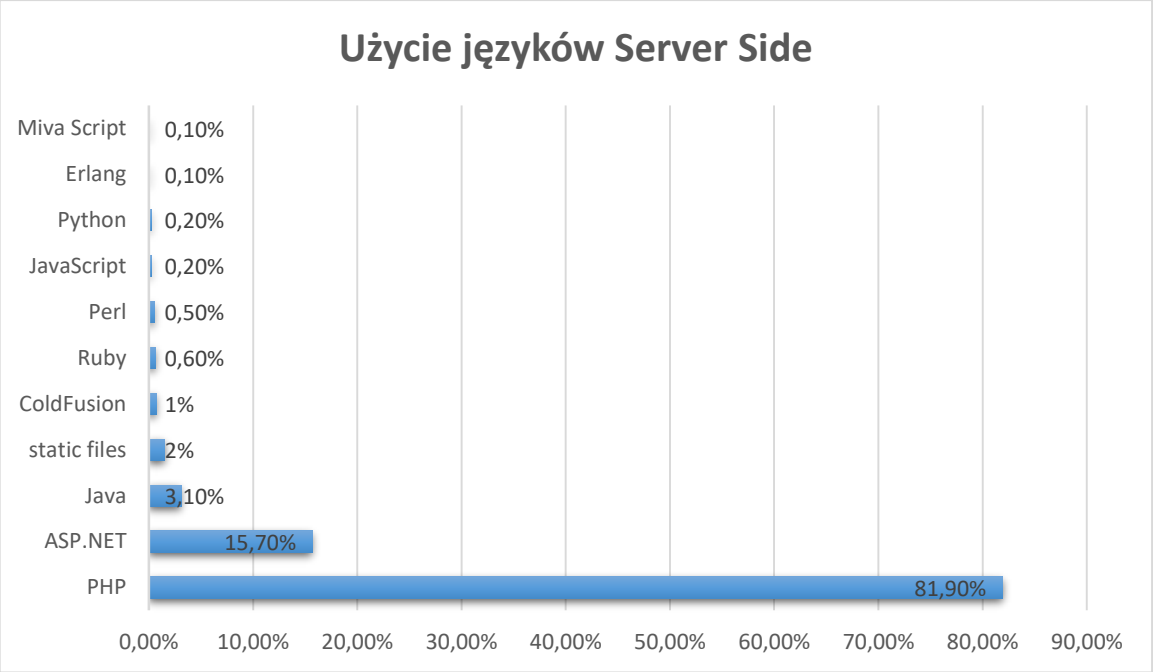
Choć relacyjne bazy danych znakomicie sprawdzają się w większości przypadków, to zdarzają się przypadki, gdy lepiej jest skorzystać z innych rozwiązań. Ze względu na konieczność zachowania spójności, może być trudno je skalować. Nierelacyjne bazy danych oferują duże możliwości skalowania kosztem spójności. Rezygnują z pewnych zalet baz relacyjnych, oferując w zamian inne. Przykładowo, silniki baz danych obsługują wyszukiwanie pełnotekstowe, jednak nie jest ono zbyt wydajne – baza danych nie do tego została zaprojektowana. W takich przypadkach znacznie lepiej jest skorzystać z dedykowanych rozwiązań, jak np. ElasticSearch, czy Solr. W przypadku potrzeby przechowywania dokumentów o nieustalonej, dynamicznej strukturze, można skorzystać z dokumentowej bazy – Mongo. Jeśli zależy nam na szybkim dostępie do danych, można skorzystać z bazy K-V, na przykład Redis.

Redis jest bazą typu key/value store, jej nazwa jest skrótem od Remote Dictionary Service. Jest bardzo wydajna. Stosuje się ją do przechowywania danych sesyjnych, cache'owania, zliczania statystyk, itd. Może również przechowywać dane na dysku. Jest bardzo łatwy do nauki i implementacji, jednak ze względu na swoją prostotę, może nie nadawać się do niektórych zastosowań [16].

Przechowywanie danych w nierelacyjnej bazie wygląda inaczej, niż w relacyjnej. Trzeba zaprojektować strukturę odpowiednio dopasowaną, tak, aby łatwo można było uzyskać dostęp do danych. W nierelacyjnej bazie nie ma złączeń, co powoduje, że struktura danych z relacyjnej bazy będzie nie do zaimplementowania w niej, albo w najlepszym przypadku bardzo niewygodna do użycia.

4.9. PHP

PHP jest internetowym skryptowym językiem programowania. Został on zaprojektowany do generowania stron internetowych. Nie jest językiem kompilowanym, tylko interpretowanym, co oznacza, że podczas wykonywania skryptu za każdym razem jest on przetwarzany przez interpreter. Jest językiem dynamicznie typowanym, choć jego najnowsza wersja umożliwia zdefiniowanie typów zwracanych przez funkcje oraz typy ich argumentów. Jest to dominujący język programowania w dziedzinie aplikacji internetowych, co jest widoczne na Wykresie 1.

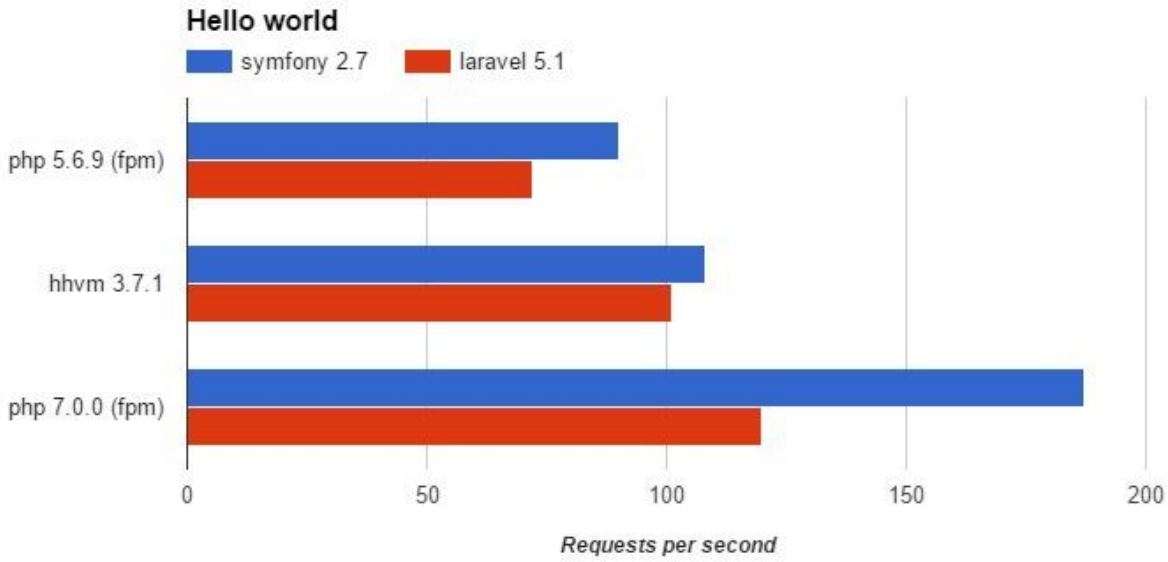


Wykres 1. Statystyki użycia języków server-side, źródło: W3Techs.com, 18 March 2016

PHP/FI (Personal Home Page/Forms Interpreter) powstał w roku 1994 jako zestaw skryptów Perla, którego celem było monitorowanie odwiedzalności strony internetowej. Zestaw ten, rozwijany przez Rasmusa Lerdorfa przekształcił się w język programowania PHP 3.0, gdy zainteresowali się nim dwaj izraelscy programiści: Zeev Suraski i Andi Gutmans. Ta wersja była pomyślana jako język modułowy, posiadał on zalążki obiektowości, ale najważniejsza była możliwość pisania własnych modułów i dołączania ich do języka. Późniejsze wydania PHP skupiały się na usprawnianiu silnika Zend Engine, wokół którego zbudowano PHP 4. Wersja 5 była przełomem – dodano wiele mechanizmów znanych z języków czysto obiektowych, jak np. dziedziczenie, referencje, obsługa wyjątków, przestrzenie nazw, wyrażenia lambda, domknięcia, cechy (trails). Wersja 6 została porzucona, gdyż była wersją eksperymentalną, która nie rozwijała się wystarczająco szybko, zaś następna wersja skupiała się na refaktoryzacji silnika Zend Engine i na usprawnieniu składni (Abstract Syntax Tree, Uniform Variable Syntax). W chwili obecnej dostępna jest wersja oznaczona numerem 7. Jedną z długo

oczekiwanych zmian jest type hinting, czyli wskazywanie typów wyników zwracanych przez funkcje i danych przyjmowanych w parametrach.

Dzięki refaktoryzacji ZE język doznał ogromnego przyspieszenia w stosunku do poprzedniej wersji, zużywa także mniej pamięci. W przedstawionym benchmarku (Wykres 2) porównywane są dwa frameworki – Symfony 2.7 oraz Laravel 5.1. Porównuje się 3 wersje PHP: 5.6.9, 7.0.0, oraz HHVM (HipHop Virtual Machnie), który jest dedykowanym rozwiązaniem Facebooka.



Wykres 2. Porównanie wersji PHP

Widać, że PHP7 wydajnością pokonuje pozostałe wersje języka.

Składnia języka PHP jest oparta o język C, Java i Perl. Kod musi się znajdować pomiędzy znacznikami `<?php` i `?>`. Przykładowy kod napisany w języku PHP z użyciem technik obiektowego programowania jest zaprezentowany na Listingu Listing 8.

```
<?php

class Car
{
      private $gas;

      public function drive()
      {
            if($this->gas >= 0) {
                  echo 'Jade!';
                  $this->gas--;
            } else {
                  echo 'Nie mam paliwa!';
            }
      }

      public function tank($amount)
      {
            echo 'Tankuję!';
            $this->gas += $amount;
      }
}

$car = new Car();
$car->tank(10);
$car->drive();
```

Listing 8. Przykład kodu w języku PHP

Do interpretera PHP można dołączać rozmaite moduły w zależności od potrzeb. Udostępniają one szereg dodatkowych funkcji możliwych do zastosowania w aplikacjach. Jest ich łącznie około 150. Są to m. in.:

- sterowniki do baz danych: MySQL, Postgres, Oracle, IBM DB2, Firebird, SQLite, MSSQL, Sybase, itd.
- parsery XML: SimpleXML, DOM XML
- LDAP
- JSON
- Funkcje daty i czasu
- Klient FTP
- ZIP
- Biblioteki graficzne GD, imageMagick
- OpenSSL
- Bzip2, ZIP

Ilość modułów sprawia, że możliwe jest napisanie rozmaitych aplikacji łączących się np. poprzez LDAP z firmowymi sieciami, crawlerów zbierających dane ze stron internetowych, automatycznych archiwizerów, a także aplikacji przetwarzających dane w formacie JSON.

4.10.Wzorzec Model-View-Controller

Jest to wzorzec architektoniczny opisujący, jak powinna zostać zbudowana aplikacja posiadająca graficzny interfejs użytkownika. Został zaprojektowany w 1979 roku przez Trygve Reenskauga, norweskiego programistę, pracującego wówczas w Xerox nad językiem Smalltalk. MVC nie jest to jedynym podejściem do tworzenia aplikacji i nie sprawdza się we wszystkich przypadkach. Są również podejścia typu MVVM, MVP. Aplikacje internetowe i frameworki w przeważającej większości implementują wzorzec MVC.

Wzorzec ten dzieli aplikację na trzy główne części:

1. Model – reprezentuje logikę aplikacji, przechowuje dane
2. Widok – wyświetla dane pochodzące z modelu
3. Kontroler – przyjmuje dane od użytkownika i modyfikuje model oraz odświeża widok

Wzorzec powstał, gdyż istniała potrzeba rozdzielenia kodu odpowiedzialnego za wyświetlanie danych od logiki biznesowej. Ułatwiało to prace nad aplikacją – widoki można było podmienić bez wpływu na logikę biznesową.

4.11.Framework Phalcon

Frameworki powszechnie określa się jako szkielety programistyczne. Definiują one strukturę aplikacji i udostępniają narzędzia oraz bibliioteki do wykorzystania przez programistę pozwalające na szybkie budowanie aplikacji. Są stosowane w wielu językach programowania. Dla PHP dostępne jest kilkanaście, jeśli nie kilkadziesiąt frameworków. Najbardziej popularne z nich to:

- Symfony
- Zend
- Yii
- Laravel

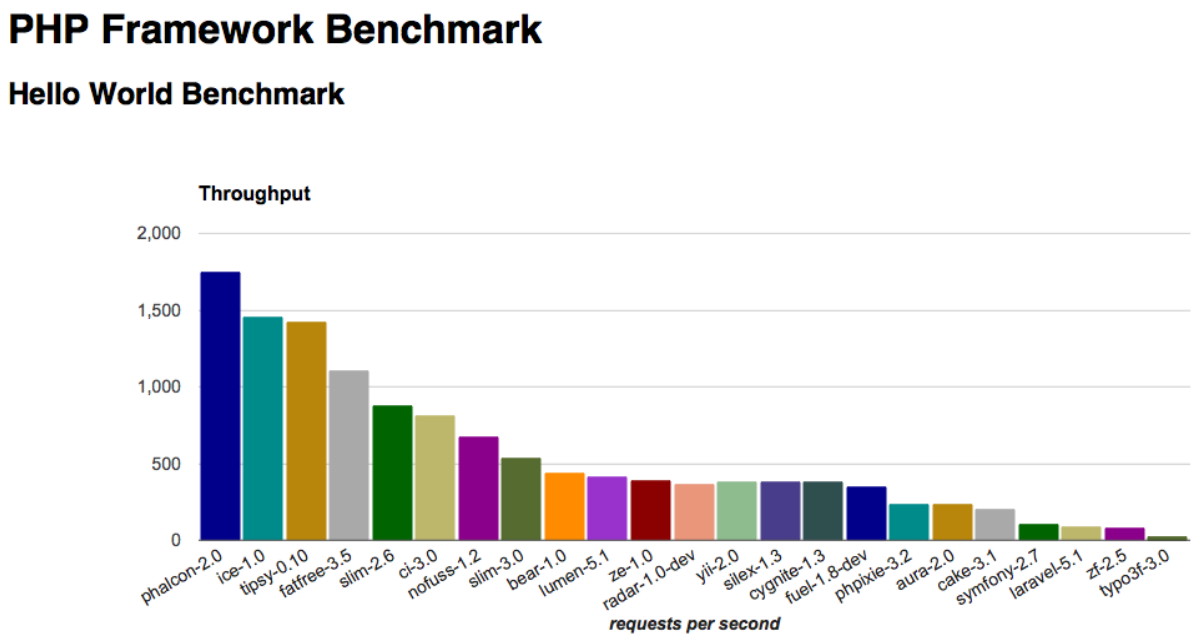
Wszystkie one sa frameworkami obiektowymi zorientowanymi na aplikacje MVC. Oprócz tego istnieją mniejsze frameworki, pozostawiające więcej swobody programiście, ale też mające mniej gotowych komponentów. W zamian za to są dużo szybsze i dobrze nadają się do budowania np. API.

Za obsługę backendu w systemie odpowiada framework Phalcon 3.0. Jest to nowoczesny obiektowy framework dostarczany jako rozszerzenie do PHP napisane w C [17]. Główną zaletą takiego

podejścia jest jego wydajność. Twórcy frameworka postawili sobie za cel zminimalizowanie wszystkich negatywnych cech frameworków. Są to:

- Duża ilość plików potrzebnych do załadowania, aby framework działał i udostępniał wszystkie potrzebne funkcje
- Ładowanie w ciągu każdego żądania http wszystkich funkcjonalności – duże wykorzystanie pamięci RAM
- Z powodu interpretowania kodu PHP w locie wszystkie pliki są przetwarzane za każdym razem na nowo

Implementacja frameworka jako rozszerzenia do PHP powoduje, że jest on jednorazowo wczytywany przy starcie serwera http wraz z modułem PHP. Dodatkowo nie jest on już interpretowany za każdym razem, gdyż jest już w postaci wykonywalnej. Powoduje to znaczące przyspieszenie działania aplikacji. Wszystkie benchmarki porównujące Phalcona z innymi popularnymi frameworkami wskazują, iż ten ma znaczącą przewagę wydajnościową, co pokazuje Wykres 3.



Wykres 3. Porównanie frameworków, źródło: https://github.com/kenjis/php-framework-benchmark

Frameworki powszechnie wykorzystywane w rozwiązaniach biznesowych, takie jak Symfony 2, Laravel 5, Zend Framework są dość mało wydajne, ale wiąże się to z ich możliwościami i rozbudowaną strukturą, pozwalającą programiście na szybkie tworzenie aplikacji. Inne, mniejsze frameworki, jak na przykład CakePHP, Yii, Code Igniter, również nie dorównują szybkością Phalconowi, a dobrze się nadają do małych i wydajnych aplikacji. Co więcej, Phalcon wydajnością pokonuje nawet mikroframeworki Silex, Lumen, które są zorientowane na wydajność i wykorzystanie innych komponentów z większych frameworków. Ich wadą jest to, że dołączenie w zależnościach

większych komponentów powoduje spowolnienie ich działania. Przyczyną jest potrzeba załadowania dodatkowych plików, których może być dużo. Phalcon jest samodzielnym frameworkiem i w jego przypadku nie ma potrzeby doładowywania dodatkowych modułów.

Wadą takiego rozwiązania jest to, że rozszerzenie Phalcona nie jest standardowo instalowane na popularnych hostingach.

Phalcon pozwala na łatwą implementację wzorca MVC w budowanych aplikacjach, posiada zintegrowany kontener Dependency Injection pozwalający na wstrzykiwanie odpowiednich zależności. W zamyśle twórców jest bardzo elastyczny i pozwala na różne sposoby zbudowania aplikacji MVC. W repozytorium github (https://github.com/phalcon/mvc) twórcy Phalcona w przykładach przedstawiają ich 18. Część wykorzystuje namespace'y, część nie, część jest pomyślana jako aplikacje jednomodułowe, część wielomodułowe, część prezentuje mikroaplikacje, część aplikacje z podziałem na warstwy. Jest to bardzo wygodny punkt startowy dla osób chcących rozpocząć przygodę z Phalconem, bądź zastanawiają się, w jaki sposób najwygodniej byłoby im zorganizować swoją aplikację. Dla bardziej zaawansowanych programistów interesujący będzie Zephir – język twórców Phalcona, który jest używany do pisania rozszerzeń do PHP, kompilowanych później do C.

4.12.REST – Representational State Transfer

Aplikacje internetowe bardzo często charakteryzują się tym, że są podzielone na część frontendową i część backendową. Często są to dwie oddzielne aplikacje charakteryzujące się swoją logiką. Najczęściej są pisane w innych językach i mają różne zadania. Udostępniane są wspólnie, jako jedna aplikacja, ale linia podziału na styku frontend – backend jest wyraźna. Frontend jest aplikacją, która jest uruchamiana i działa po stronie klienta. Może być to aplikacja napisana w Angularze, może być to zestaw formularzy HTML-owych, aplikacja w języku Flash. Backend natomiast jest aplikacją uruchamiana po stronie serwera i zapewniającą właściwe działanie logiki biznesowej.

Z podziałem aplikacji na dwie części wiąże się konieczność zapewnienia komunikacji pomiędzy nimi. Frontend powinien przesyłać wprowadzone dane do backendu, a ten powinien je walidować, zapisywać w jakimś repozytorium danych, przetwarzać i zwracać wyniki do wyświetlenia przez frontend.

REST jest wzorcem architektonicznym, który skupia się bardziej na podstawowej jednostce, jaką jest zasób i sposoby dostępu do niego. Nie określa on implementacji i szczegółów, a skupia się na relacjach między zasobami i ich rolach. Sieć WWW można określić jako usługę typu REST, w której mamy możliwość odwołania się do konkretnego zasobu za pomocą metod z protokołu HTTP.

Najczęściej właśnie ten protokół jest podstawą usług REST, jednak aplikację typu RESTful można zaimplementować korzystając z innych protokołów.

Ponieważ protokół HTTP jest bardzo elastyczny, z łatwością można wykorzystać go do zaimplementowania usługi REST. Aby podać przykład takiej usługi, możemy sobie wyobrazić, że pod adresem http://przyklad.pl/samochod mamy zasoby związane z samochodami. Możemy się do nich odwołać za pomocą metod HTTP, co prezentuje Tabela **Błąd! Nie można odnaleźć źródła odwołania.**.

Zasób	Metoda	Wynik
http://przyklad.pl/samochod	GET	Lista samochodów
http://przyklad.pl/samochod/1	GET	Samochód o id 1
http://przyklad.pl/samochod	POST Dane samochodu w ciele żądania	Dodanie samochodu
http://przyklad.pl/samochod/1	PUT Dane samochodu w ciele żądania	Aktualizacja danych samochodu o id 1
http://przyklad.pl/samochod/1	DELETE	Usunięcie samochodu o id 1

Tabela 1. Przykładowe metody REST

Zaletą architektury REST jest możliwość stworzenia API, które może być wykorzystywane przez rozmaite aplikacje bez konieczności ustalania szczegółowej specyfikacji dostępu.

5. Prototyp internetowego systemu do anotowania korpusu języka migowego

W niniejszym rozdziale Autor przedstawia opracowany prototyp systemu pozwalającego na anotację korpusu języka migowego. Prezentuje przyjęte założenia i opisuje bardziej szczegółowo niektóre ważne funkcjonalności i ich implementację.

5.1. Założenia architektoniczne

Celem Autora jest opracowanie systemu pozwalającego na stworzenie kolejnych pluginów do pracy nad klipami video, oraz umożliwiającego zmianę sposobu zapisu i przechowywania danych. Oznacza to konieczność zaprojektowania modularnej budowy systemu i interfejsów do komunikacji między modułami. Konieczna jest izolacja odpowiedzialności poszczególnych elementów, tak, aby zmiana jednego elementu na inny nie miała wpływu na działanie całego systemu.

Aplikacja jest podzielona na część frontową i część backendową korzystającą z repozytorium danych. Komunikacja pomiędzy nimi zapewniana jest przez REST API. Części te są od siebie zupełnie niezależne – implementacja zgodna z REST sprawia, że implementacja backendu nie jest zależna od aplikacji frontowej. Ten sam backend może być wykorzystany przy innej aplikacji przy założeniu, że posiada ona wiedzę na temat zasobów serwowanych przez niego, może to być np. aplikacja mobilna, albo prosta przeglądarka danych.

Rysunek 16. Źródło - opracowanie własne

Autor wykorzystuje wzorzec MVC.

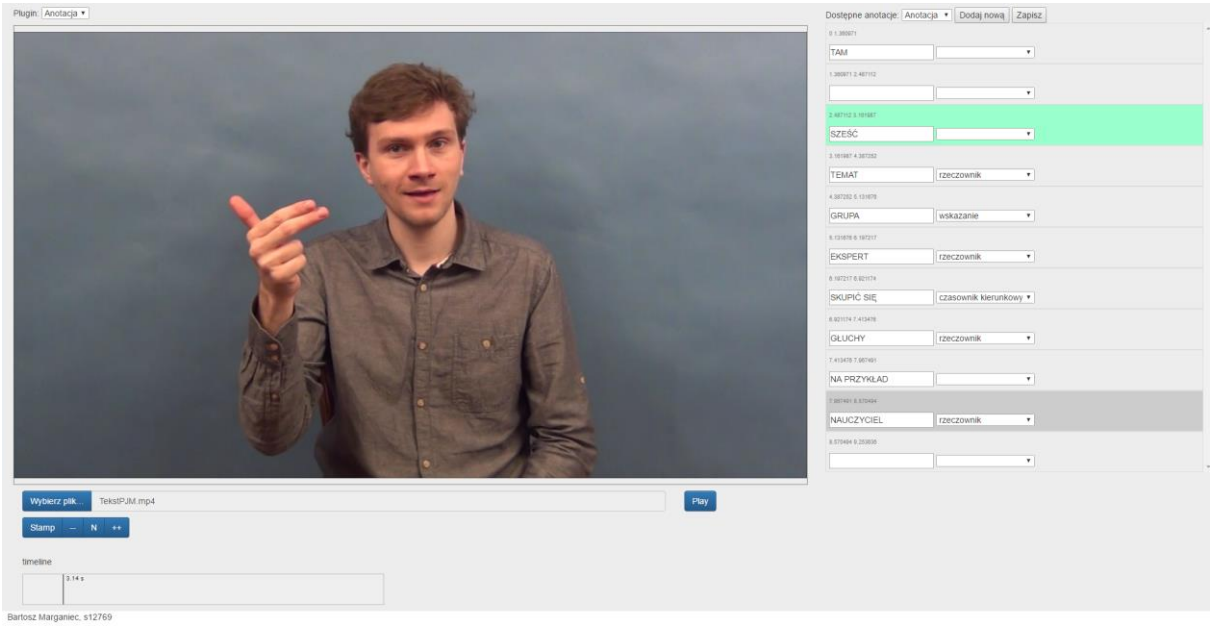
5.2. Frontend - moduły systemu i ich odpowiedzialności

Aplikacja frontendowa została napisana w JavaScript z wykorzystaniem wzorca AMD przy wsparciu biblioteki RequireJS. Główne wymaganie, jakie musi być spełnione, to możliwość pisania własnych pluginów wykorzystujących podstawowe funkcje systemu i dodające dodatkowe możliwości. Aby było to możliwe, należało zaprojektować system ładowania pluginów, zdefiniować interfejsy, które te pluginy muszą implementować, aby poprawnie one działały. Należało zdefiniować, które

komponenty systemu mają być podstawowe i dostępne dla wszystkich, a które mogą być ładowane przez użytkownika.

5.2.1 Ekran aplikacji

Główny ekran aplikacji prezentuje Rysunek 17. Głównym elementem jest okno klipu filmowego, który jest anotowany. Po prawej stronie zawarte są dane anotacyjne, które zostały opracowane. Jest na nich przedstawiony plugin do anotacji, który oprócz wpisania znaczenia danego znaku pozwala na wybranie z listy rozwijanej typu znaku.



Rysunek 17. Główny ekran aplikacji

Pod klipem video znajduje się miejsce na kontrolki playera, które mogą być różne w zależności od aktualnie załadowanego pluginu. Dla pluginu do anotacji jest to przycisk Stamp, oraz przyciski do przyspieszania i zwalniania odtwarzanego filmu. Przykład kontrolek pluginu do tworzenia napisów prezentuje Rysunek 18.

Rysunek 18. Kontrolki do pluginu do dodawania napisów

Górna część ekranu aplikacji umożliwia nam zmianę pluginu, oraz zmianę kontenera do przechowywania danych, zapisywanie i odczytywanie poprzednio zapisanych informacji na temat danego klipu filmowego.

Rysunek 19. Kontrolki do zmiany pluginu i zapisu/odczytu danych

5.2.2 Standardowa budowa modułu aplikacji na przykładzie modułu Store

Specyfika języka Javascript jest taka, że nie posiada on typowych klas znanych z innych języków, ani też nie umożliwia tworzenia metod publicznych i prywatnych [18]. Jednak jest wystarczająco elastyczny, aby taką funkcjonalność zaimplementować w inny sposób. Dzięki budowie modułowej można zapewnić, żeby każdy moduł udostępniał wybrane zmienne i funkcje, które deklaruje. Przykład implementacji takiego rozwiązania prezentuje Listing 9, na którym zawarty jest fragment kodu aplikacji. Jest to definicja komponentu Store, którego zadaniem jest przechowywanie w tablicy obiektów z danymi.

Komponent Store ma zdefiniowaną zależność od modułów Rejestr i Backend, który jest ładowany do zmiennej Registry. Przy ładowaniu modułu przez bibliotekę RequireJS uruchamiana jest fabryka, w której deklarowane są funkcje i zmienne pomocnicze. Każdy moduł aplikacji deklaruje funkcję init(), która jest na końcu wywoływana. Pełni ona rolę konstruktora obiektu. Funkcja ta może wykonać szereg działań potrzebnych do prawidłowego działania modułu, np. może zarejestrować potrzebne obiekty w Rejestrze, co się dzieje w przypadku komponentu Store. Rejestrowany jest w nim obiekt `that`, który przechowuje wszystkie zmienne i funkcje, które chcemy udostępnić na zewnątrz. Widzimy np. że funkcja `addRecord` została dodana do obiektu `that`, a `validateRecord` nie. Oznacza to, że po zarejestrowaniu `that` w Rejestrze pod kluczem ‘store’ będziemy mogli wywołać funkcję `addRecord` za pomocą polecenia `Registry.get('store').addRecord()`. Jest to niejako symulacja zachowania metod publicznych i prywatnych.

```
define(['Registry', 'Backend'], function(Registry, Backend) {

    //<editor-fold defaultstate="collapsed" desc="inicjalizacja zmiennych">
    var that = {},
        init,

        selected = null,
        records = [],

        actualStoreId = null,

        saveStore,
        getStore,
        getStoresForMovie,
        addStore,
        selectStore,
        clear,
        addRecord,
        validateRecord,
        unselectRecords,
        unmarkRecords,
        unmarkRecord,
        markRecordPlayerOnto,
        selectRecord,
        getSelectedRecord,
        findRecordByTime,
        openAddDialog,
        openAddErrorDialog,
        fillStoreSelect,
        bindStoreSelect,
        getDataFromRecordsToSave;
    //</editor-fold>

    //others...

    //<editor-fold defaultstate="collapsed" desc="addRecord">
    addRecord = function(record)
    {
        validateRecord(record);

        records.push(record);
        records.sort(sortByTime);

        $('#storage').append(record);
    };
    that.addRecord = addRecord;
    //</editor-fold>

    //<editor-fold defaultstate="collapsed" desc="validateRecord">
    validateRecord = function(record)
    {
        if(
            typeof record.data('start') === 'undefined' ||
            typeof record.data('end') === 'undefined' ||
            typeof record.data('new') === 'undefined' ||
            typeof record.data('edited') === 'undefined'
        ) {
            throw "Brak danych!";
        }
    };
    //</editor-fold>

    //<editor-fold defaultstate="collapsed" desc="bind">
    bind = function()
    {
```

```
        $('.js-layer-clear').on('click', function(e) {
            e.preventDefault();
            clear();
        });

        $('#store-save').on('click', function(e) {
            e.preventDefault();
            saveStore();
        });

        $('#store-add').on('click', function(e) {
            if(Registry.get('player').isVideoSetted()) {
                openAddDialog();
            } else {
                openAddErrorDialog();
            }
        });

        bindStoreSelect();
    };
    //</editor-fold>

    //<editor-fold defaultstate="collapsed" desc="init">
    init = function ()
    {
        Registry.add({id: 'store', object: that});

        bind();
    };
    //</editor-fold>

    init();

    return that;
});
```

Listing 9. Przykład modułu RequireJS

Moduł nie musi zwracać żadnego obiektu `that`. Wówczas jest po prostu komponentem, który przy uruchomieniu wywołuje metody, które programista chce, aby wykonał – np. załadowanie odpowiednich danych, ustawienie wyglądu okien, itd.

Wszystkie komponenty są zbudowane w ten sposób, z tym że pluginy dodatkowo muszą implementować odpowiednie metody, aby można było je zarejestrować w systemie. Wymusza to odpowiedni obiekt PluginSelect, o którym jest mowa w dalszej części pracy.

5.2.3 Bootstrap aplikacji

Aplikacja internetowa uruchamiana z przeglądarki musi wiedzieć, jakie moduły systemu powinna załadować i uruchomić, aby całość działała poprawnie. Bootstrap jest to punkt startowy aplikacji frontendowej. Ładuje on komponenty Store, Player, Timeline, PluginSelect, które są podstawowymi komponentami systemu.

```
define(
    ['object/Store', 'object/Player', 'object/Timeline', 'object/PluginSelect'],
    function(store, player, timeline) {
        player.load();
        timeline.load();
    }
);
```

Listing 10. Bootstrap aplikacji JS

Po załadowaniu bootstrap uruchamia metody `load` obiektów player i timeline. Metody te podpinają odpowiednie elementy HTML do obiektów, aby te mogły nimi zarządzać i nasłuchiwać zdarzeń.

5.2.4 Rejestr

Rejestr jest komponentem realizującym wzorzec Registry. Jego definicja znajduje się w `public/js/app/Registry.js`. Moduły Player, Timeline, Store podczas inicjalizacji rejestrują się w Rejestrze, aby zapewnić, że ich metody będą dostępne w obrębie całego systemu.

```
    init = function ()
    {
        Registry.add({id: 'store', object: that});

        bind();
    };
```

Listing 11. Inicjalizacja modułu

Rejestracja obiektu w rejestrze odbywa się przez przekazanie do metody add obiektu zawierającego pola id i object. Wówczas dostęp do tego obiektu jest możliwy przez Registry.get(id); Rejestr kontroluje dostęp do zmiennej globalnej REGISTRY_SERVICES, która przechowuje tablicę zarejestrowanych obiektów.

Taki sposób wykorzystania Rejestru jest w pewien sposób implementacją wzorca Dependency Injection, który zakłada usuwanie bezpośrednich zależności pomiędzy obiektami na rzecz rejestrowania i wstrzykiwania ich. Powoduje to, że kod jest łatwiejszy do testowania oraz bardziej odporny na błędy. Możliwe jest również podmienianie klas i mockowanie ich. Jest to realizacja paradygmatu Inversion of Control.

5.2.5 System ładowania pluginów

Aplikacja frontowa w zamierzeniu autora miała na celu udostępnić możliwość pisania wtyczek pozwalających na wykorzystywanie istniejących w niej elementów. Moduł `PluginSelect` zdefiniowany w `public/js/app/object/PluginSelect.js` służy do wyboru pluginów z listy dostępnych i sprawdza, czy spełniają one określone warunki – to znaczy, czy implementują odpowiednie

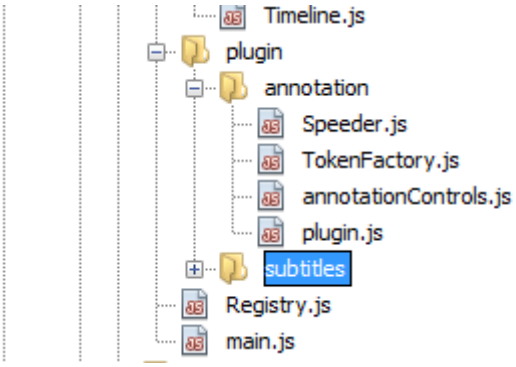
metody. Działa to na zasadzie kontraktu – jeśli te metody są implementowane, to zakłada się, że realizują one to, co powinny realizować.

Metody wymagane do zaimplementowania w pluginach to:

- load
- clean
- createRecord

Pluginy, aby były możliwe do załadowania, muszą być umieszczone w katalogu plugin, we własnym katalogu, w którym znajduje się plik `plugins.js`, zawierający implementację wymaganych metod.

Rysunek 20 przedstawia plugin do anotacji nazwany „annotation”. Jego struktura nie jest ograniczona w żaden inny sposób – można w `plugin.js` wywołać inne, utworzone przez siebie moduły pluginu, tak, aby uporządkować kod związany ze wtyczką.



Rysunek 20 Plugin do anotacji

Ładowanie pluginu odbywa się poprzez RequireJS. W momencie ładowania sprawdzana jest implementacja wymaganych funkcji (Listing 12), po czym plugin jest ładowany i rejestrowany w Rejestrze (Listing 13).

```
    checkPlugin = function(plugin)
    {
        if(plugin.hasOwnProperty('load') === false) {
            throw 'Plugin doesn\'t implement load method.';
        }

        if(plugin.hasOwnProperty('clean') === false) {
            throw 'Plugin doesn\'t implement clean method.';
        }

        if(plugin.hasOwnProperty('createRecord') === false) {
            throw 'Plugin doesn\'t implement createRecord method.';
        }

        return true;
}
```

Listing 12. Sprawdzenie, czy plugin implementuje wszystkie potrzebne metody

Ładowanie pluginu, sprawdzenie i rejestracja:

```
loadPlugin = function(plugin)
    {
        require(['plugin/' + plugin + '/plugin'], function(pluginToLoad)
        {
            if(checkPlugin(pluginToLoad)) {
                actualPlugin = pluginToLoad;
                actualPluginName = plugin;
                actualPlugin.load();
                Registry.add({id: 'plugin', object: pluginToLoad});
            }
        });
    };
```

Listing 13. Ładowanie pluginu

W czasie przeładowywania pluginów poprzedni plugin powinien zostać wyczyszczony, aby jego elementy nie kolidowały z elementami kolejnego pluginu. `PluginSelect` przy każdym takim przeładowaniu wykonuje metodę `clean` pluginu i wykonuje dodatkowe operacje.

```
cleanPlugin = function()
    {
        if(actualPlugin !== null) {
            actualPlugin.clean();
            actualPlugin = null;
            actualPluginName = null;
            Registry.remove('plugin');
        }
    };
```

Listing 14. Usunięcie pluginu

Autor stosuje zasadę rozdzielania odpowiedzialności. Oznacza to, że wiedzę na temat tego, jak plugin ma działać, wie tylko on sam. Moduł ładujący pluginy nie zna szczegółów działania tych pluginów – wie tylko, że muszą on zaimplementować trzy metody: `load`, `clean` i `createRecord`.

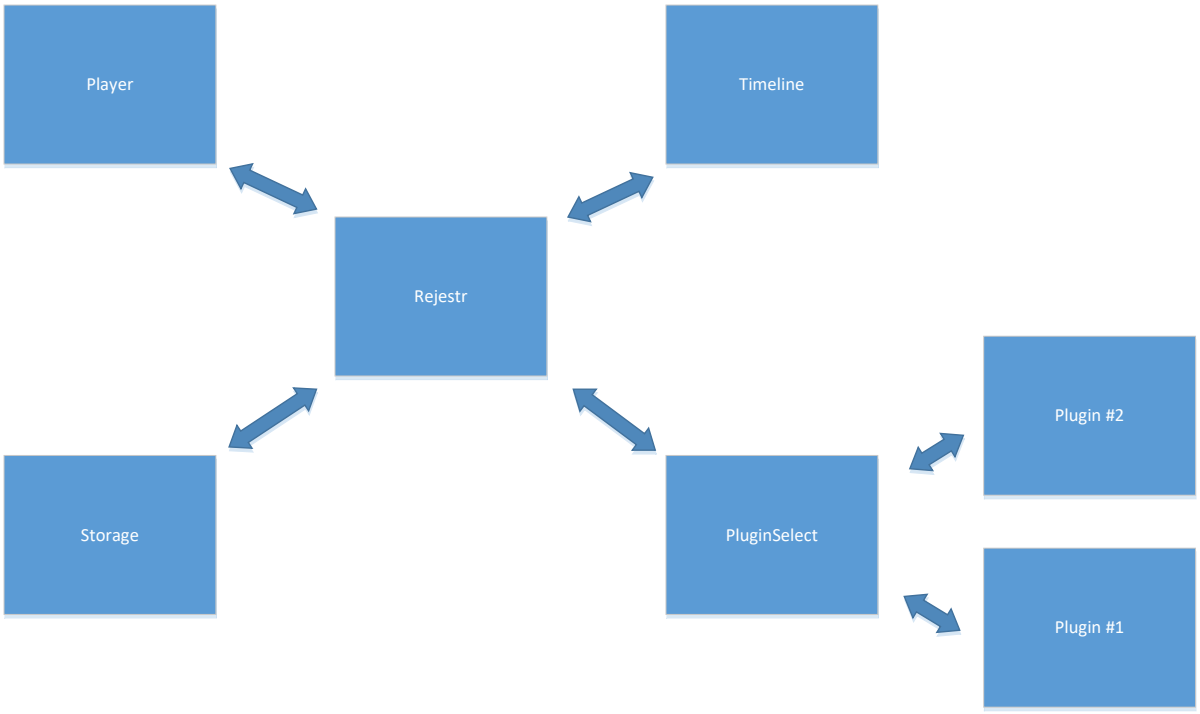
Powoduje to, że możliwości pisania pluginów są właściwie nieograniczone. Autor wtyczki może zdecydować np. że umieści przy ładowaniu przycisk, do którego podepnie akcję obliczania MD5 filmu, który wyświetli w utworzonym polu tekstowym. Przy czyszczeniu powinien on przywrócić aplikację do stanu wyjściowego, to znaczy usunąć utworzone przez siebie obiekty. Aplikacja z założenia ma służyć do tworzenia rekordów opisujących film – to znaczy napisów, albo tokenów anotujących język migowy, więc wymusza na pluginach implementację tworzenia takiego rekordu.

Rekordy z danymi mają z góry ustaloną strukturę. Zawierają czas rozpoczęcia, zakończenia, oraz metadane. Każdy plugin ma pełną dowolność w zarządzaniu metadanymi – może tam być tekst lub obiekt JSON, który będzie przy wczytywaniu interpretowany. Metadane przy zapisie są serializowane i przy odczycie deserializowane.

Pluginy operują na udostępnionych i stałych elementach aplikacji, to znaczy:

- Playera
- Timeline
- Storage

Dostęp do tych elementów jest umożliwiony poprzez Rejestr, w którym są zarejestrowane. Także pluginy są rejestrowane – za każdym razem pod tym samym identyfikatorem `plugin`. Nie może być dwóch jednocześnie działających pluginów. Powiązanie modułów prezentuje Rysunek 21.



Rysunek 21. Architektura aplikacji frontendowej

5.2.6 Timeline

Moduł ten jest odpowiednikiem osi czasu w playerze. Pozwala on na przewinięcie klipu do wybranego miejsca. Korzysta on z obiektu canvas, który, zgodnie ze specyfikacją HTML5 służy do renderowania kształtów i obrazów. Kod HTML jest widoczny na Listingu 15.

```
<div class="col-md-12" id="timeline">
    <p>timeline</p>
    <div id="timeline-wrapper">
        <canvas width='600' height='50' id="timeline-canvas"></canvas>
    </div>
</div>
```

Listing 15. Zdefiniowanie timeline

Obiekt Timeline udostępnia API, na podstawie którego między innymi można zdefiniować bardziej skomplikowane operacje. Można zdefiniować własne zdarzenia uruchamiane na timeline, co prezentuje Listing 16.

```
    addEventToCanvas = function(event, callback)
    {
        canvas.addEventListener(event, callback);
    };
    that.addEventToCanvas = addEventToCanvas;
```

Listing 16. Metoda do dodania zdarzenia

Tę funkcjonalność wykorzystuje plugin do dodawania napisów. Dodaje on możliwość zaznaczania na timeline czasu startu i czasu zakończenia napisów bez potrzeby wykorzystywania przycisków. Wykorzystuje do tego zdarzenia `mousedown` i `mouseup`.

API timeline prezentuje Tabela 2.

Metoda	Przeznaczenie
`addEventToCanvas`	Dodaje zdarzenie do canvas
`clear`	Czyści canvas timeline z naniesionych obiektów
`drawLine`	Rysuje linię pionową czasu
`drawText`	Pisze tekst na canvas
`calculateTime`	Oblicza czas na podstawie położenia na timeline
`width`	Zwraca szerokość osi czasu

`updateTimeline`	Aktualizuje canvas timeline, rysuje podstawowe obiekty

Tabela 2. API Timeline

5.2.7 Store

Store zajmuje się przechowywaniem danych generowaniem przez wtyczki, oraz komunikacją z backendem poprzez REST API. Udostępnia on metody do dodania rekordu, co prezentuje Listing 17.

```
    addRecord = function(record)
    {
        validateRecord(record);

        records.push(record);
        records.sort(sortByTime);

        $('#storage').append(record);
    };
    that.addRecord = addRecord;
```

Listing 17. Dodanie rekordu do Store

Metoda `addRecord` jest dołączana do obiektu `that`, co oznacza, że jest możliwa do wywołania z zewnątrz obiektu.

Zapis zawartości jest wywoływany poprzez przycisk Zapisz, który wywołuje AJAX-owe żądanie POST przetwarzane przez backend.

```
$.ajax({
    method: 'PUT',
    url: '/store/' + actualStoreId,
    dataType: 'json',
    data: {
        records: getDataFromRecordsToSave()
    },
    success: function() {
        selectStore(actualStoreId);
    }
});
```

Listing 18. Wysłanie żądania AJAX aktualizującego zawartość Store

Store jest modułem komunikującym się pomiędzy frontendem a backendem. Do niego powinny się odwoływać wszystkie wtyczki i w nim przechowywać swoje dane przeznaczone do zapisu w repozytorium danych.

API Store jest widoczne w Tabeli 3.

Metoda	Przeznaczenie

`getStoresForMovie`	Pobiera kontenery z danymi dla danego filmu
`addStore`	Dodaje nowy kontener na dane dla filmu
`saveStore`	Zapisuje kontener w repozytorium
`addRecord`	Dodaje rekord do kontenera
`unselectRecords`	Odznacza zaznaczone rekordy
`unmarkRecords`	Zdejmuje flagę z rekordów
`unmarkRecord`	Zdejmuje flagę z wybranego rekordu
`markRecordPlayerOnto`	Kładzie flagę na rekordzie, który aktualnie wyświetla player
`selectRecord`	Zaznacza dany rekord
`unselectRecord`	Odznacza dany rekord
`getSelectedRecord`	Pobiera zaznaczony rekord
`findRecordByTime`	Zwraca rekord zawierający podany czas
`clear`	Czyści aktualny kontener

Tabela 3. API Store

5.2.8 Player

Player jest modułem aplikacji odpowiedzialnym za odtwarzanie plików video. Jest to moduł, który udostępnia metody do odtwarzania play(), pauzowania pause(), ustawiania czasu, prędkości odtwarzania. Jest zarejestrowany w Rejestrze, a zatem jego metody publiczne są dostępne do użycia przez wszystkie pozostałe komponenty. Pozwala to na napisanie pluginu wykonującego dowolne udostępnione operacje na klipie video w odtwarzaczu.

Player jest podpięty pod obiekt DOM $(„#player”). Wykorzystuje on metody HTML5 do sterowania playerem.

API Playera jest widoczne w Tabeli Tabela 4. API playera

Metoda	Przeznaczenie
`setOnTime`	Ustawia klip video na określonym momencie
`getCurrentTime`	Pobiera aktualny czas playera

`getDuration`	Pobiera długość klipu filmowego
`getPlaybackRate`	Pobiera prędkość odtwarzania klipu
`setPlaybackRate`	Ustawia prędkość odtwarzania klipu
`isVideoSetted`	Sprawdza, czy w playerze jest ustawiony klip filmowy
`play`	Odtwarza klip filmowy
`pause`	Pauzuje klip filmowy
`clear`	Usuwa klip filmowy z playera
`getMovieName`	Pobiera nazwę klipu filmowego
`getMovieSize`	Pobiera rozmiar klipu filmowego
`getVideoPlayer`	Pobiera obiekt jQuery playera
`setSource`	Otwiera klip filmowy w playerze

Tabela 4. API playera

5.2.9 Pluginy

Autor pracy stworzył dwie wtyczki działające na różne sposoby. Jedna z nich służy do anotacji korpusu języka migowego. Pozwala ona na utworzenie rekordów zawierających dane dotyczące danej jednostki leksykalnej. Każdy taki rekord ma miejsce na wpisanie znaczenia tej jednostki w języku polskim, oraz przypisania jej odpowiedniego znaczenia gramatycznego. Dane te są zapisywane w JSON. Przykład:

```
{
	„id”: 132,
	„start”: 2.23,
	„end”: 2.90,
	„metadata”: {
		„text”: „KOT”,
		„glossType”: „rzeczownik”
	},
	„new”: false,
	„edited”: true
}
```

Listing 19. JSON z danymi pochodzącymi z pluginu anotującego

Pola `new` i `edited` zawierają informację dla backendu, co zrobić z danym rekordem – utworzyć nowy, czy zaktualizować już istniejący.

Oba pluginy przy ładowaniu ustawiają swoje własne kontrolki, za pośrednictwem których odbywa się sterowanie pluginem. Przykładowo plugin „annotation” przy ładowaniu wywołuje kod tworzący element zawierający jego kontrolki i umieszcza je po elemencie `#player-file`.

```
var controls = $('<div id="annotation-controls"></div>');

controls.html(html);
$('#player-file').after(controls);
```

Listing 20. Utworzenie kontrolek pluginu i umieszczenie ich w aplikacji

Gdy plugin zostaje zdezaktywowany, kontroler pluginów wywołuje jego metodę clean, w której następuje usunięcie dodanych kontrolek.

```
$('#annotation-controls').remove();
```

Listing 21. Usunięcie kontrolek pluginu

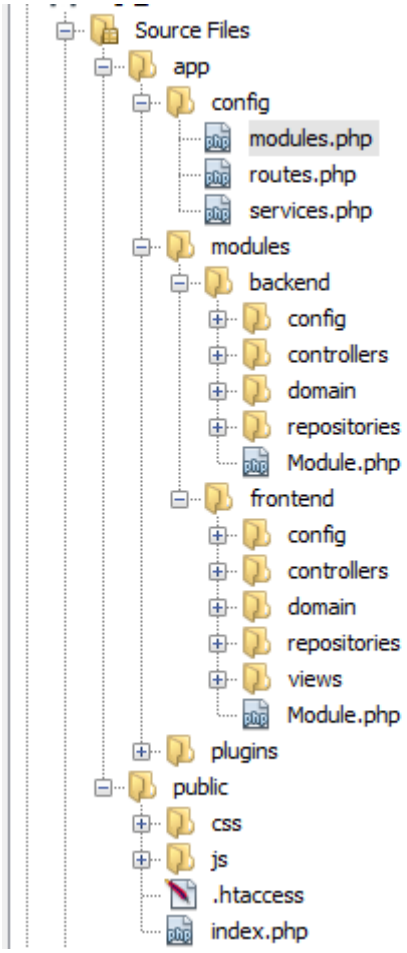
5.3. Backend

W niniejszym podrozdziale Autor prezentuje część backendową. Opisuje strukturę backendu oraz implementację przyjętych założeń. Szczególną uwagę poświęca mechanizmowi wymienialności repozytoriów i kontenerowi Dependency Injection.

5.3.1 Modularyzacja

Zadaniem części backendowej jest wyświetlenie treści dla użytkownika, przetwarzanie danych otrzymanych od użytkownika, ich walidacja i zapisywanie w repozytorium danych. Autor podzielił backend na moduły, które są niezależne od siebie i w razie potrzeby mogą być zainstalowane na innej maszynie. Umożliwia to skalowanie rozwiązania. Uniezależnienie od siebie modułów powoduje, że łatwo można je wymieniać. API REST zostało napisane w PHPie, ale ponieważ jest to odrębny moduł, można go zastąpić API napisanym na bazie Springa bez naruszania modułu generującego panel aplikacji, obsługującego logowanie i pozostałe części serwisu.

Phalcon jest frameworkiem dość minimalistycznym, pozwalającym na swobodę konfiguracji. Rozdzielenie na moduły jest jedną z jego podstawowych funkcjonalności. Rysunek 22 prezentuje strukturę backendu.



Rysunek 22. Struktura aplikacji. Źródło: opracowanie własne

Bootstrap aplikacji mieści się w pliku `public/index.php`. Najważniejsze jego działania to rejestracja serwisów, uruchomienie aplikacji i utworzenie kontenera DI. Aplikacja po przetworzeniu requesta zwraca odpowiedź.

```
require __DIR__ . '/../app/config/services.php';

$application = new Application();

$application->setDI($di);

require __DIR__ . '/../app/config/modules.php';


echo $application->handle()->getContent();
```

Listing 22. Bootstrap backendu

Główna konfiguracja aplikacji zawarta jest w katalogu `config` – są to ścieżki routingów, rejestracja modułów, oraz rejestracja serwisów w kontenerze DI.

Każdy moduł zawarty jest w katalogu `modules` i jest zdefiniowany w pliku `Module.php`, w którym rejestrowane są w kontenerze DI właściwe dla niego serwisy oraz rejestrowane są przestrzenie nazw.

5.3.2 Kontener Dependency Injection i wymienialność repozytoriów

Phalcon udostępnia w swoim frameworku kontener Dependeny Injection. Jest to klasa `Phalcon\Di\FactoryDefault`. Framework umożliwia napisanie własnego kontenera – musi on implementować interfejs `Phalcon\DiInterface`. Autor korzysta w swojej pracy z domyślnego kontenera.

Przykładowo, Listing 23 prezentuje, jak moduł frontendowy rejestruje w kontenerze serwis `userRepository` ze wstrzykniętym połączeniem z bazą danych `db`, który później jest wykorzystywany do pobrania danych o użytkowniku.

```
        $di->set('db', function() use ($config) {
            return new \Phalcon\Db\Adapter\Pdo\Postgresql(array(
                "host" => $config->database->host,
                "username" => $config->database->username,
                "password" => $config->database->password,
                "dbname" => $config->database->dbname
            ));
        });

        $di['userRepository'] = [
            'className' => 'Modules\Frontend\Repository\UserRepository',
            'arguments' => [
                ['type' => 'service', 'name' => 'db']
            ]
        ];
```

Listing 23. Konfiguracja repozytorium danych

Moduł backendowy może korzystać z innego repozytorium, niż baza danych – na przykład z bazy Redis. Przykład konfiguracji prezentuje Listing 24.Listing 24. Konfiguracja bazy Redis

```
$di['redis'] = function() use ($config) {
     $redis = new \Redis();
     $redis->connect($config->redis->host, $config->redis->port);

     return $redis;
};

$di['redisRepository'] = [
     'className' => 'Modules\Backend\Repository\RedisRepository',
     'arguments' => [
          ['type' => 'service', 'name' => 'redis']
     ]
];
```

Listing 24. Konfiguracja bazy Redis

Dane konfiguracyjne z pliku `config/config.php` w module backend definiują wszystkie dane potrzebne do połączenia z bazą danych postgres i redis. Ustawiony jest także parametr `repository` na `redis`.

```
<?php
return new \Phalcon\Config(array(
    'database' => array(
        'adapter'  => 'Postgres',
        'host'     => 'localhost',
        'username' => 'mgr',
        'password' => 'mgr',
        'dbname'     => 'mgr',
    ),
    'redis' => [
        'host' => 'localhost',
        'port' => 6379,
        'persistent' => false
    ],

    'repository' => 'redis',
));
```

Listing 25. Konfiguracja backendu

Kontener Dependency Injection przy ładowaniu modułu ładuje odpowiednie repozytorium:

```
if($di[$repository . 'Repository'] instanceof StoreRepositoryInterface !== false) {
    $di['storeRepository'] = $di[$repository . 'Repository'];
} else {
        throw new \Exception('Repozytorium nie implementuje
StoreRepositoryInterface!');
}
```

Listing 26. Ładowanie odpowiedniego repozytorium

W tym wypadku jest to wcześniej zarejestrowany `redisRepository`. Jak widać, kontener może być dowolnie kontrolowany przez programistę. Wybór repozytorium może być ustawiony w pliku konfiguracyjnym, może być zaszyty w bazie danych, skąd kontener może go pobrać, albo w pliku tekstowym, aby mógł być łatwo edytowany przez użytkownika. Po stronie programisty leży zapewnienie, że repozytorium posiada odpowiednie metody. Jest to realizowane poprzez interfejs `StoreRepositoryInterface`.

Zarejestrowany serwis jest wykorzystywany w kontrolerze `StoreController`:

```
$storeRepository = $this->di->get('storeRepository');
$stores = $storeRepository->getFiltered($auth['id'], $plugin, $movieName,
$movieSize);
```

Listing 27. Użycie zarejestrowanego repozytorium przez kontroler

W ten sposób możliwa jest wymienialność repozytoriów do zapisu i odczytu danych z backendu. Wystarczy ustawić odpowiednie dane konfiguracyjne dla nowego repozytorium, stworzyć odpowiednią klasę implementującą `StoreRepositoryInterface`, zarejestrować w kontenerze pod jakimś kluczem, który można ustawić w pliku `config.php`.

5.3.3 Zabezpieczenie dostępu

W przypadku anotacji tekstów języka migowego zabezpieczenie danych przed niepowołanym dostępem ma istotne znaczenie. Phalcon pozwala w prosty sposób zabezpieczyć wybrane akcje, przekierować na stronę logowania. Wystarczy w konfiguracji serwisów odpowiednio skonfigurować Dispatcher tak, aby ten przed wykonaniem odpowiedniej akcji z routingu sprawdził, czy użytkownik ma do niej dostęp (Listing 28). W przypadku braku uprawnień następuje przekierowanie do strony logowania.

```
$di['dispatcher'] =  function () {
    $eventsManager = new EventsManager();
    $eventsManager->attach('dispatch:beforeExecuteRoute', new SecurityPlugin);
    $dispatcher = new Dispatcher();
    $dispatcher->setEventsManager($eventsManager);

    return $dispatcher;
};
```

Listing 28. Zarejestrowanie dispatchera i zabezpieczenie dostępu

Kontrolą dostępu zajmuje się `SecurityPlugin`, który jest wywoływany przy przechwyceniu zdarzenia `dispatch:beforeExecuteRoute`. Klasa ta za parametr otrzymuje zdarzenie, oraz instancję Dispatchera przechowującą wszystkie informacje o żądaniu użytkownika, tj. o tym, która akcja została wykonana. Tam można zdefiniować poziom dostępu, skorzystać z listy ACL, lub z innej metody przyznania dostępu. Możliwe jest również zdefiniowanie poziomu dostępu w samym kontrolerze, którego dotyczy żądanie. Odbywa się to poprzez zaimplementowanie metody `beforeExecuteRoute`, co prezentuje Listing 29. Prototyp wykorzystuje obie metody, ponieważ moduł backendu może być przeniesiony w zupełnie inne miejsce i powinien mieć wydzieloną konfigurację. W kontrolerze `StoreController` zdefiniowana jest metoda, która przed wykonaniem akcji sprawdza po prostu, czy użytkownik jest zalogowany.

```
    public function beforeExecuteRoute($dispatcher)
    {
        $auth = $this->session->get('auth');

        if(!$auth) {
            $this->dispatcher->forward(
                array(
                    'controller' => 'error',
                    'action'     => 'api',
                    'params'     => ['message' => 'Not allowed']
                )
            );
        }
    }
```

Listing 29. Implementacja zabezpieczenia na poziomie kontrolera

6. Podsumowanie

Rozwijające się badania lingwistyczne nad językami migowymi wskazują na potrzebę istnienia oprogramowania ułatwiającego pracę badaczom. Istniejące rozwiązania wymuszają na powstających jednostkach naukowych wdrażanie dużych systemów typu iLex wraz z infrastrukturą bazodanową i sprzętem firmy Apple, albo nie umożliwiają skorzystanie z zalet nowoczesnych technologii internetowych (ELAN, Sign Stream).

W niniejszej pracy zrealizowano prototyp systemu pozwalającego na zdalną pracę z klipami video. Założeniem autora było stworzenie systemu, który przede wszystkim pozwalałby na wykonywanie pracy badawczej nad językiem migowym w sposób zbliżony do dotychczas istniejących programów. Prototyp jest podzielony na 3 główne części, Frontend, Backend i Repozytorium. Ich architektura umożliwia wdrożenie aplikacji w sposób niskokosztowy i dający możliwość wydzielenia osobnego serwera do przechowywania danych.

Główna część Frontendu jest napisana w Javascript z użyciem RequireJS, ułatwiającym zarządzanie kodem. Ma to duże znaczenie dla przyszłego rozwijania projektu. Backend i Frontend są zaimplementowane w wydajnym frameworku Phalcon wydanym dla języka PHP.

6.1. Wady i zalety rozwiązania

Zaprojektowanie prototypu jako aplikacji internetowej daje możliwości zdalnej pracy badaczy, wspólnego anotowania korpusu i przy dodaniu kolejnych funkcjonalności może ułatwić im pracę badawczą. Korzystanie z takiej aplikacji nie jest ograniczone przez wersję czy rodzaj systemu operacyjnego.

W przypadku internetowych systemów możliwość dzielenia się danymi jest ułatwiona. Może to być duża zaleta w przypadku międzynarodowej pracy zespołów badawczych i chęci przeprowadzenia badań porównawczych.

Wadą rozwiązania jest prosty i skromny interfejs, który może przeszkadzać użytkownikom przyzwyczajonym do dobrze zaprojektowanych aplikacji. Przyjęty w pracy sposób wyświetlania danych jest możliwy do modyfikacji poprzez odpowiednie ostylowanie albo skorzystanie z gotowych frameworków HTML, jak np. Bootstrap, czy PureCSS.

Pracy na klipach video oznacza konieczność posiadania szybkiego łącza internetowego. W obecnych czasach nie jest to problem, jednak w przypadku badań wyjazdowych, np. w celu zbadania języków migowych używanych w słabo ucywilizowanych regionach świata, może to być problem.

6.2. Proponowane kierunki rozwoju aplikacji

Autor skupił się przede wszystkim na implementacji podstawowych funkcjonalności, zdając sobie sprawę z tego, że istniejące programy są znacznie bardziej rozwinięte. Jednak prototyp systemu nie posiada ograniczenia sprzętowego ani aplikacyjnego, a jego funkcjonalności można rozwinąć poprzez zestaw wtyczek.

Proponowane możliwości rozwoju aplikacji i rozszerzania jej funkcjonalności:

1. Rozszerzenie pluginu do anotacji i opracowanie zdefiniowanych warstw z zdefiniowanymi wartościami

Obecnie plugin do adnotacji pozwala na zdefiniowanie typu znaku migowego w wąskim zakresie. Rozszerzenie możliwości definiowania pozwoliłoby na przyszłe skuteczne przeszukiwanie zanotowanych danych i dokładniejsze badania lingwistyczne.

2. Rozszerzenie możliwości ustawiania znaczników czasowych

Przydatna jest możliwość ustawiania dokładnych czasów rozpoczęcia i zakończenia trwania danego rekordu. Rozbudowanie obecnie istniejących prostych funkcjonalności ustawiania czasów do dokładnego wpisywania numeru klatki, czy czasu, ułatwiłoby pracę w anotacji.

3. Opracowanie języka zapytań i implementacja przeszukiwania

Wartym uwagi jest możliwość przeszukiwania danych w sposób wygodny dla badacza. Po opracowaniu dokładnego schematu danych anotowanych przez plugin możliwe jest opracowanie sposobu przeszukiwania.

4. Implementacja modułu do konwersji danych pomiędzy różnymi systemami anotacji

Ze względu na już istniejące zanotowane dane, które są zapisywane w różnych formatach, jak np. XML, czy w postaci rekordów w bazie danych, należałoby rozszerzyć aplikację o możliwość wykorzystywania tych danych poprzez dodanie modułu do importu i konwersji danych, a także eksportu.

5. Moduł do wyświetlania danych statystycznych

W badaniach lingwistycznych duże znaczenie mają dane statystyczne, na podstawie których testuje się hipotezy lingwistyczne. Zdefiniowanie jakiegoś sposobu prezentowania zagregowanych danych oraz raportów z danych językowych byłoby bardzo przydatne dla użytkowników aplikacji.

6. Moduł do automatycznego wyświetlania słownika zanotowanych danych

Jednym z podstawowych celów tworzenia korpusów językowych są nie tylko badania gramatyczne, ale także tworzenie słowników do różnych zastosowań – frekwencyjnych, synonimicznych, itp. Dodanie takiego modułu zwiększyłoby wartość aplikacji.

7. Możliwość anotowania filmów pochodzących z YouTube

Znaczna ilość materiału językowego znajduje się na wielu kanałach YouTube. Zawierają one cenne z punktu widzenia lingwistyki dane, gdyż są to swobodne i niekontrolowane teksty. Dodając do aplikacji możliwość zbierania i anotowania takich materiałów zwiększono by jej przydatność i skuteczność badań lingwistycznych.

Bibliografia

[1] K. Jednoróg, Ł. Bola, P. Mostowski, M. Szwed, P. Boguszewski, A. Marchewka i P. Rutkowski, „Three-dimensional grammar in the brain: Dissociating the neural correlates of natural sign language and manually coded spoken language," *Neuropsychologia,* nr 71, pp. 191-200, 2015.

[2] S. Łozińska, „Języki migowe jako przedmiot badań," w *Lingwistyka przestrzeni i ruchu. Komunikacja migowa a metody korpusowe*, P. Rutkowski i S. Łozińska, Redaktorzy, Warszawa, Wydział Polonistyki Uniwersytetu Warszawskiego, 2014, pp. 37-38, ISBN 978-83-64111-12-9.

[3] J. Filipczak, „Anotacja korpusu PJM," w *Lingwistyka przestrzeni i ruchu. Komunikacja migowa a metody korpusowe*, Warszawa, Wydział Polonistyki Uniwersytetu Warszawskiego, 2014, ISBN 978-83-64111-12-9.

[4] korpusy.net, „Słowniczek terminów korpusowych," [Online]. Available: http://korpusy.net/component/option,com_glossary/id,8/. [Data uzyskania dostępu: 28 sierpień 2016].

[5] P. Wittenburg, H. Brugman, A. Russel, A. Klassmann i H. Sloetjes, „ELAN: a Professional Framework for Multimodality Research" w *Proceedings of LREC 2006*, Fifth International Conference on Language Resources and Evaluation, 2006.

[6] D. V. U. M. H. A. S. M. T. J. G. Birgit Hellwig, „ELAN - Linguistic Annotator Manual," 12 maj 2016. [Online]. Available: http://www.mpi.nl/corpus/html/elan/index.html . [Data uzyskania dostępu: 28 sierpień 2016].

[7] H. Thomas i S. Jakob, iLex – A Database Tool for Integrating Sign Language Corpus Linguistics and Sign Language Lexicography, Hamburg, 2008.

[8] D. MacLaughlin, C. Neidle i D. Greenfield, „Sign Stream's User Guide," Grudzień 2000. [Online]. Available: http://www.bu.edu/asllrp/asllrpr9.pdf.

[9] E. Evans, Domain-Driven Design. Zapanuj nad złożonym systemem informatycznym, Helion, 2004, ISBN 978-83-283-0525-0.

[10] C. Larman, UML i wzorce projektowe. Analiza i projektowanie obiektowe oraz iteracyjny model wytwarzania aplikacji, Helion, 2011, ISBN 978-83-246-2874-2.

[11] M. Fowler, „Inversion of Control Containers and the Dependency Injection," 2004. [Online]. Available: http://martinfowler.com/articles/injection.html.

[12] Z. Kessin, HTML5. Programowanie aplikacji, Helion, 2012, ISBN 978-83-246-4897-9.

[13] A. MacCaw, Javascript. Aplikacje WWW, Helion, 2012, ISBN 978-83-246-3887-1.

[14] D. Flanagan, jQuery. Leksykon kieszonkowy, Helion, 2012.

[15] J. Burke, „Why AMD?," [Online]. Available: http://requirejs.org/docs/whyamd.html. [Data uzyskania dostępu: 23 sierpień 2016].

[16] K. Seguin, „The Little Redis Book," [Online]. Available: http://redis.io/documentation. [Data uzyskania dostępu: 28 sierpień 2016].

[17] „Phalcon 3.0.0 Documentation," 2016. [Online]. Available: https://docs.phalconphp.com/en/latest/index.html.

[18] S. Stefanov, Javascript. Wzorce, Helion, 2012, ISBN 978-83-246-3821-5.

[19] T. Hanke, iLex - A tool for sign language lexicography and corpus analysis, LREC 2002, 2002.

[20] S. Newman, Budowanie mikrousług, Helion, 2015, ISBN 978-83-283-1381-1.

[21] W. Sanders, PHP. Wzorce projektowe, Helion, 2013, ISBN 978-83-246-7455-8.

Rysunki

Listingi

Wykresy

Tabele