AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Fizyki i Informatyki Stosowanej

Praca magisterska

Karol Stasiak

kierunek studiów: **Informatyka stosowana**

specjalność: **Modelowanie i analiza danych**

Konwerter prezentacji PowerPoint2TeX

Opiekun: **dr hab. Małgorzata Krawczyk, prof. AGH**

Kraków, wrzesień 2020

Oświadczenie studenta

Uprzedzony(-a) o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. z 2018 r. poz. 1191 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony(-a) o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. — Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

.....

(czytelny podpis)

Kraków, ?? września 2020

Tematyka pracy magisterskiej i praktyki dyplomowej Karola Stasiak, studenta drugiego roku studiów drugiego stopnia na kierunku informatyka stosowana, specjalności analiza i modelowanie danych

Temat pracy magisterskiej: **Konwerter prezentacji PowerPoint2TeX**

Opiekun pracy:	dr hab. Małgorzata Krawczyk, prof. AGH
Recenzenci pracy:	
Miejsce praktyki dyplomowej:	WFiIS AGH, Kraków

Program pracy magisterskiej i praktyki dyplomowej

1. Omówienie realizacji pracy magisterskiej z opiekunem.
2. Zebranie i opracowanie literatury dotyczącej tematu pracy.
3. Praktyka dyplomowa:
   - zapoznanie się z ideą formatu PPTX oraz TEX,
   - wybór technologii i tworzonego oprogramowania,
   - opracowanie architektury programu,
   - sporządzenie sprawozdania z praktyki.
4. Kontynuacja implementacji konwersji z formatu PPTX na TEX:
   - rozwój serwera aplikacyjnego,
   - stworzenie kodu klienta aplikacji,
   - dopracowanie konwersji.
5. Opracowanie redakcyjne pracy.

Termin oddania w dziekanacie: ?? września 2020

....

(podpis kierownika katedry) (podpis opiekuna)

Merytoryczna ocena pracy przez opiekuna:

Merytoryczna ocena pracy przez recenzenta:

Spis treści

1 Wstęp

1.1 Wprowadzenie

Jednym z najpopularniejszych sposobów obrazowania treści są prezentacje multimedialne. Obecnie wykorzystywane są one w coraz większej ilości dziedzin życia: do prezentowania treści nauczania podczas wykładów na uczelni, do przedstawiania wyników badań podczas konferencji naukowej czy podczas spotkań biznesowych w firmie. Dzięki specjalnym programom jak PowerPoint, tworzenie prezentacji stało się bardzo proste. Same w sobie nie przekazują dużo treści, lecz najczęściej używane są jako podstawa, na której opiera się wystąpienie. W rzeczywistości, prezentacja powinna zawierać jak najmniej tekstu. Jak pokazują badania, ludzki umysł nie jest w stanie jednocześnie czytać i słuchać, gdyż za te dwie czynności odpowiada ten sam obszar mózgu [1]. Jak zatem powinna wyglądać dobra prezentacja? By odpowiedzieć na to pytanie warto sięgnąć do badań naukowych z dziedziny neurobiologii, a dokładniej tego, w jaki sposób działa ludzka percepcja. Człowiek zapamiętuje dużo więcej informacji, jeśli pokazywane są one na różne sposoby, np. poprzez prezentację wraz z przemową i komunikację niewerbalną [2]. Ważne by w prezentacji nie przesadzać z tekstem. Warto wspomóc się innymi sposobami wizualizacji informacji, np. poprzez tabele, obrazki czy wykresy. Znaczące jest także ułożenie elementów na slajdzie oraz użyta kolorystyka.

Gdy już wiemy jak powinna wyglądać nasza perfekcyjna prezentacja, możemy przystąpić do jej wykonania. Pytanie, jak ją stworzyć? Można użyć programu typu WYSIWYG[1] np. PowerPoint. Tworzenie prezentacji w ten sposób jest bardzo proste i przyjemne. Czasami jednak potrzebujemy większej precyzji i dokładności przy umieszczaniu elementów na slajdzie. Możemy do tego wykorzystać inne sposoby tworzenia prezentacji, np. stworzyć prezentację z użyciem kodu TEX. Taka prezentacja odznacza się dużo bardziej czytelną i przewidywalną strukturą.

Jak zatem wybrać? Dzięki nowoczesnym rozwiązaniom nie musimy być przywiązani do jednego formatu. W ramach tej pracy stworzono konwerter prezentacji w formacie PPTX na kod TEX. Dzięki temu możliwe jest połączenie prostoty tworzenia slajdów z ich estetycznym wyglądem.

1.2 Cel i zakres pracy

Celem pracy było stworzenie aplikacji umożliwiającej konwersję prezentacji w formacie PPTX na kod TEX. Aplikacja taka powinna wspierać konwersję dostępnych w formacie PPTX treści, jednocześnie brać pod uwagę specyfikę prezentacji tworzonych z użyciem kodu TEX. Zdecydowano się na realizację tego zadania, poprzez aplikację internetową w architekturze klient-serwer.

[1] WYSIWYG (ang. what you see is what you get) – akronim stosowany w informatyce dla określenia metod, które pozwalają uzyskać wynik w publikacji identyczny lub bardzo zbliżony do obrazu na ekranie

2 Wykorzystane technologie

2.1 Backend

2.1.1 Java

Java to język programowania, który powstał w 1995 [3] i nadal cieszy się on dużą popularnością (drugie miejsce według wskaźnika Tiobe [4]). W ostatnich latach zaszło w nim wiele zmian, dzięki przejściu na półroczny cykl wydawniczy. Co pół roku udostępniana jest nowa wersja z nowymi funkcjonalnościami. Aktualnie wersją z najdłuższym wsparciem (LTS - *long-term support*) jest wersja 11, w której pojawiły się m.in.:

- zmienne lokalne,
- rozszerzone API[2] wielu klas, w tym String oraz Optional,
- uruchamianie programu jedną komendą,
- metody prywatne w interfejsach.

2.1.2 Spring

Spring jest szkieletem tworzenia aplikacji, który obecnie jest już standardem w rozwiązaniach biznesowych [5]. Jego architektura opiera się o kontener IoC[3] oraz o wstrzykiwanie zależności. Programista definiuje specjalne klasy-komponenty (ang. `beans`), a Spring odpowiada za ich cykl życia. By skorzystać z tak zdefiniowanych klas, należy użyć specjalnej adnotacji `@Autowired`. Spring "wstrzyknie" tam odpowiednią instację danej klasy. Ułatwia to w znacznym stopniu pisanie skomplikowanych aplikacji o wielu zależnościach. Dodatkowo, kod napisany w ten sposób jest łatwy w testowaniu. Spring dostarcza wiele innych, niemniej użytecznych, projektów, takich jak: Spring Boot [6], Spring Data [7] czy Spring MVC [8].

2.1.3 Biblioteka Apache POI

Biblioteka Apache POI jest biblioteką dostarczającą API w języku Java do manipulacji wieloma formatami dokumentów opartych na standardzie Office Open XML (OOXML) [9]. Dzięki niej w Javie możliwe jest odczytywanie i tworzenie dokumentów MS Excel, MS Word czy MS PowerPoint.

[2] API (ang. Application Programming Interface) - interfejs programistyczny aplikacji

[3] IoC (ang. Inversion of Control) - paradygmat polegający na przeniesieniu funkcji sterowania wykonywaniem programu do używanego frameworku

2.2 Frontend

2.2.1 React

React jest jednym z trzech obecnie najpopularniejszych projektów do tworzenia aplikacji klienckich [10]. Został on stworzony w 2013 przez Facebook i nadal jest bardzo prężnie rozwijany [11]. Opiera się na komponentach i ich stanach. Jako biblioteka udostępnia bardzo bogate API do modyfikacji DOMu[4]. Ostatnią duża zmianą w wersji 16.8 było dodanie tzw. hooków, które umożliwiają korzystanie ze stanów obiektów oraz innych funkcjonalności bez konieczności tworzenia tradycyjnych klas.

2.2.2 Redux

Redux to biblioteka, która w znaczny sposób ułatwia zarządzaniem stanem aplikacji [12]. Oferuje ona główny "magazyn" stanu oraz definiuje metody jego modyfikacji. Główne założenia biblioteki to:

- jedno źródło prawdy - cały stan przechowywany jest w pojedynczym obiekcie,
- stan jest tylko do odczytu - nie ma możliwości jego zmiany, gdy zajdzie potrzeba tworzona jest jego nowa wersja,
- zmiany stanu mogą być wykonywane jedynie przez tzw. czyste funkcje[5].

Gdy zajdzie potrzeba zmiany stanu, w kodzie wywoływane są specjalne `akcje`. Wyzwalają one funkcje nazywane `reducer-ami`. `Reducery` to czyste funkcje, które otrzymują aktualny stan aplikacji oraz informację, jaka akcja została wywołana. Na tej podstawie zwracają one nowy stan. Taka architektura umożliwia pisanie przejrzystego i łatwego w utrzymywaniu kodu.

3 Format prezentacji

3.1 Office Open XML

PPTX jest formatem wprowadzonym przez firmę Microsoft jako następca formatu PPT. Jest on oparty na standardzie *Office Open XML*, znanym również jako OpenXML lub OOXML, który jest formatem definiowania dokumentów z użyciem języka XML[6].

[4]DOM (ang. Document Object Model) – sposób reprezentacji złożonych dokumentów XML i HTML w postaci modelu obiektowego

[5]czyste funkcje (ang. pure functions) - funkcje, które akceptują wejście i zwracają wynik bez modyfikacji danych, które są poza ich zasięgiem

[6]XML (ang. Extensible Markup Language) – uniwersalny język znaczników przeznaczony do reprezentowania różnych danych w strukturalizowany sposób.

Specyfikacja formatu została wydana we współpracy z firmą Microsoft w 2006 i opisana jest w standardzie ECMA-376 [13]. Druga wersja specyfikacji ukazała się w 2008 roku, trzecia w 2011, czwarta w 2012, natomiast najnowsza piąta w roku 2016.

Poprzednie formaty dokumentów używane przez firmę Microsoft (doc, ppt, xsl) są tzw. formatami zamkniętymi. Są to pliki binarne, możliwe do pełnej modyfikacji jedynie przez konkretne programy. Nowe formaty są natomiast otwarte, przez co każdy ma do nich wgląd, może je modyfikować i ich używać.

Nowy format od początku tworzony był, by w pełni wspierać funkcjonalności poprzednich formatów, a przy tym dodawać nowe funkcjonalności. Twórcy starali się zawrzeć w nim [9]:

- Interoperacyjność - format jest niezależny od zastrzeżonych formatów czy środowisk,
- Umiędzynarodowienie - wspieranie większości języków,
- Niska bariera adaptacji - łatwy sposób adaptacji przez inne systemy, niska krzywa uczenia,
- Wysoka wierność migracji - pełna migracja funkcjonalności poprzednich formatów,
- Przestrzeń dla innowacji - budowa łatwo rozszerzalnych w przyszłości modułów.

Dokumentacja formatu zawiera 3 specyfikacje:

- WordprocessingML[7] - procesowanie dokumentów tekstowych,
- SpreadsheetML - procesowanie arkuszy kalkulacyjnych,
- PresentationML - procesowanie prezentacji.

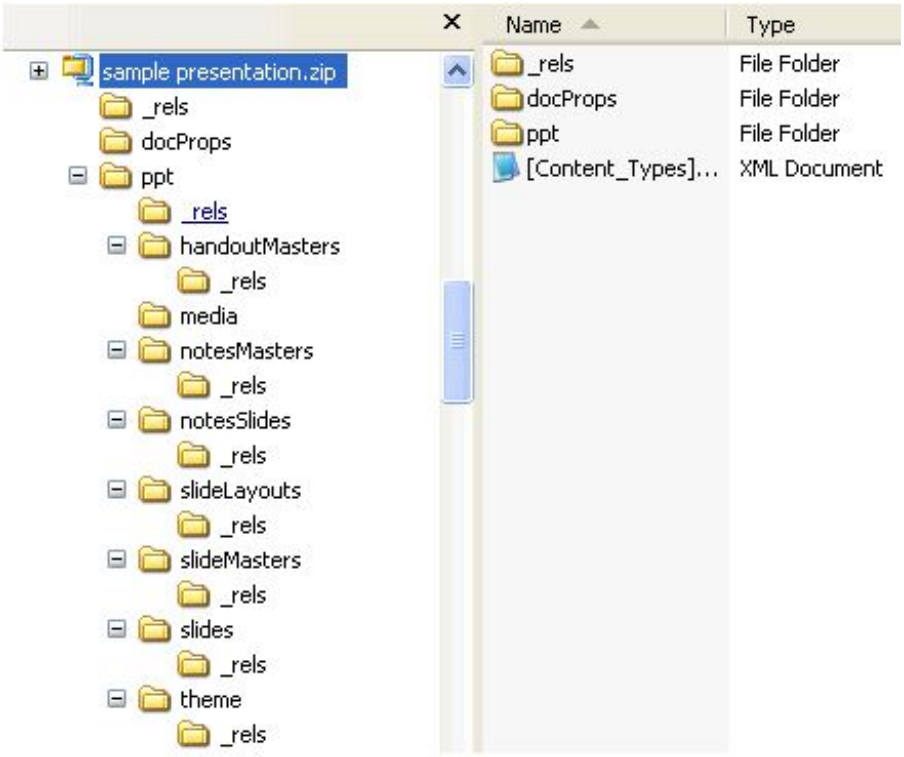
oraz opisy dodatkowych formatów, takich jak:

- DrawingML - rysunki, wykresy, tabele,
- Math - formuły matematyczne,
- Bibliography - bibliografia.

3.2 PresentationML

Plik PPTX jest plikiem zip, składającym się z wielu plików XML wzajemnie ze sobą powiązanych oraz z plików multimedialnych (np. obrazków). Struktura pliku zip została opisana w części drugiej standardu ECMA-376. Przykładowa struktura archiwum zip przedstawiona została na Rys. 1.

[7]ML (ang. Markup Language) - język znaczników



Rysunek 1: Przykładowa struktura archiwum zip.
Źródło: officeopenxml.com

Głównym elementem archiwum jest plik [Content_Types].xml, znajdujący się w katalogu głównym. Każdy element występujący w prezentacji (*part* - część) musi być zdefiniowany w tym pliku. Poniżej przedstawiono jego przykładową strukturę:

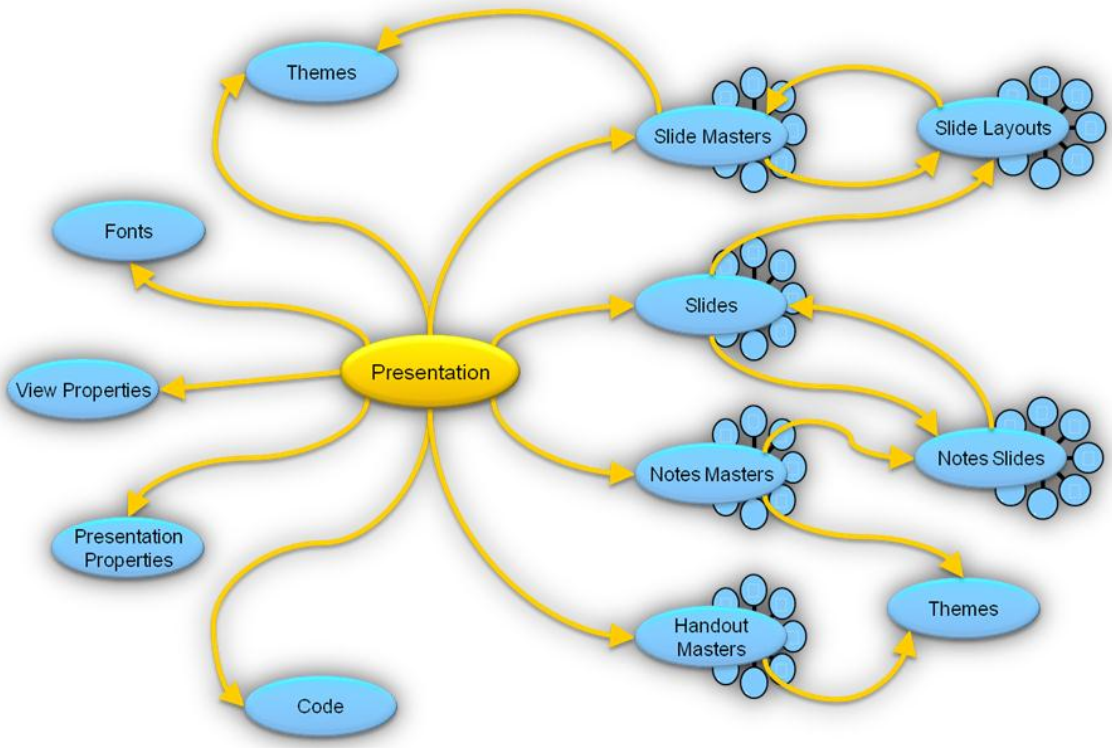
```
<Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
    <Default Extension="rels"
        ContentType="application/vnd.openxmlformats-package.relationships+xml"/>
    <Override PartName="/presentation.xml"
        ContentType="application/vnd.openxmlformats-officedocument.
    presentationml.presentation.main+xml"/>
    <Override PartName="/ppt/slides/slide1.xml"
        ContentType="application/vnd.openxmlformats-officedocument.
    presentationml.slide+xml"/>
</Types>
```

W powyższym pliku zdefiniowano część z relacjami, główną część prezentacji oraz jeden slajd.

Każde archiwum zawiera plik, w którym zdefiniowane są relacje zachodzące pomiędzy poszczególnymi częściami archiwum, przy czym każda z nich (jak na przykład każdy slajd) ma również swój własny plik z relacjami. Pliki te, znajdują się w katalogu "_rels" i mają rozszerzenie ".rels".

```
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Id="rId1"
        Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/slideLayout"
                Target="../slideLayouts/slideLayout1.xml"/>
</Relationships>
```

Powyżej przedstawiono plik relacji dla pojedynczego slajdu. Określa on układ slajdu (slide layout). Dzięki takim relacjom możliwa jest zmiana podanego układu slajdu, bez konieczności modyfikacji samego slajdu. Jeśli użytkownik będzie chciał zmienić układ na inny, wystarczy zmienić powyższą relację na inną.



Rysunek 2: Przykładowa struktura zawartości archiwum.
Źródło: docs.microsoft.com

Elementy, które definiuje PresentationML, i które mogą znaleźć się w archiwum, przedstawiono w tabeli 1. Na rysunku 2 przedstawiono natomiast graficznie relacje pomiędzy nimi.

Część	Opis
Presentation	Główny element prezentacji. Zawiera definicję prezentacji oraz listę slajdów.
Themes	Motyw prezentacji, określa np. czcionkę i schemat kolorów.
View Properties	Określa właściwości wyświetlania prezentacji.
Presentation Properties	Zawiera wszystkie właściwości prezentacji.
Slide Master	Zawiera definicję formatowania tekstu i obiektów, które pojawiają się na każdym slajdzie. Jest on nadrzędnym wzorcem i jeśli slajd nie definiuje własnego formatowania jest on wykorzystywany.
Slide Layouts	Szablon, z którego zbudowane są slajdy. Definiuje on domyślny wygląd i położenie np. obrazków na slajdzie.
Slide	Definiuje zawartość pojedynczego slajdu.
Notes Masters	Określa styl formatowania tekstu dla wszystkich notatek.
Notes Slide	Zawiera notatki dla pojedynczego slajdu.
Handout Masters	Zawiera wygląd, położenie i rozmiar slajdów, tekst nagłówka i stopki, datę oraz numer slajdu.
Comments	Zawiera komentarze do pojedynczego slajdu.
Comments Authors	Zawiera informacje o autorze komentarzy.
User-Defined Tags	Zawiera zdefiniowane przez użytkownika właściwości dla obiektu na slajdzie.
Audio	Zawiera plik dźwiękowy.
Image	Zawiera obrazek.
Video	Zawiera plik wideo.
Embedded package	Zawiera kompletny pakiet zewnętrzny, np. arkusz kalkulacyjny.

Tabela 1: Opis części, które mogą znaleźć się w archiwum.

W plikach XML znajdujących się w archiwum, używane są różne schematy XML. Najważniejsze z nich to:

- PresentationML (przedrostek p),
- Relationships (przedrostek r),
- DrawingML (przedrostek a).

3.2.1 Prezentacja

Głównym elementem archiwum jest plik presentation.xml. Zawiera on listę slajdów, a dokładniej referencje do nich, gdyż każdy slajd jest oddzielną częścią umieszczoną w katalogu "/ppt/slides".

```
<p:presentation>
    <p:sldMasterIdLst>
        <p:sldMasterId id="2147483648" r:id="rId1"/>
    </p:sldMasterIdLst>
    <p:sldIdLst>
        <p:sldId id="256" r:id="rId2"/>
        <p:sldId id="257" r:id="rId3"/>
    </p:sldIdLst>
    <p:sldSz cx="12192000" cy="6858000"/>
    <p:notesSz cx="6858000" cy="9144000"/>
    <p:defaultTextStyle>
        ...
    </p:defaultTextStyle>
</p:presentation>
```

Powyżej przedstawiono przykładową strukturę pliku presentation.xml. Zawiera on odwołanie do dwóch slajdów.

Element:

- <p:sldMasterIdLst> (List of Slide Master IDs) - określa relację do slajdu głównego, który zawiera definicję formatowania slajdu. Atrybut "r:id" określa konkretne id relacji w pliku z relacjami (taki plik zawsze znajduje się w bieżącej ścieżce w katalogu pod nazwą "_rels"),
- <p:sldIdLst> (List of Slide IDs) - zawiera listę slajdów (również poprzez id relacji). W tym przypadku są to dwa slajdy,
- <p:sldSz> (Presentation Slide Size) - określa powierzchnię slajdu, na której mogą znajdować się tekst oraz inne elementy,
- <p:noteSz> (Notes Slide Size) - określa powierzchnię używaną dla notatek,
- <p:defaultTextStyle> (Presentation Default Text Style) - określa domyślny styl tekstu dla prezentacji.

3.2.2 Slajd główny

Slajd główny (ang. Slide Master) zawiera definicję formatowania tekstu i obiektów, które pojawiają się na każdym slajdzie w prezentacji, a także listę dostępnych układów (ang. Slide

Layouts) będących szablonami, z których budowane są slajdy. Pojedyncze slajdy definiują relacje do slajdu głównego i dziedziczą wszystkie jego właściwości. W archiwum musi znajdować się przynajmniej jeden slajd główny.

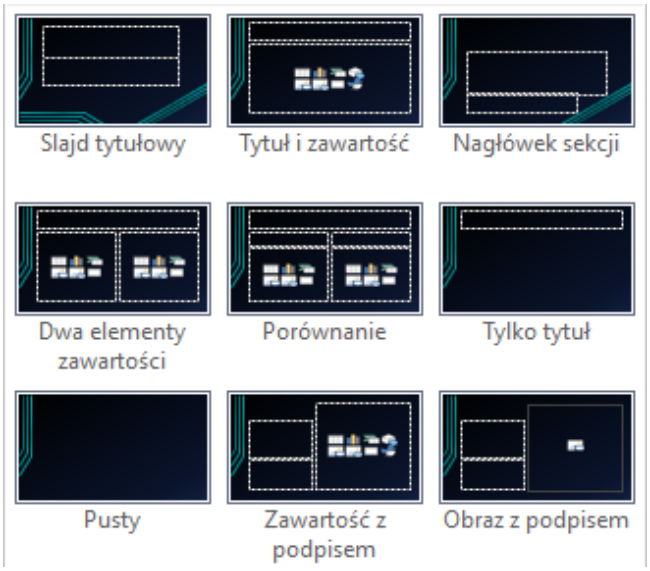
```
<p:sldMaster>
    <p:cSld>
        ...
    </p:cSld>
    <p:clrMap bg1="lt1" tx1="dk1" bg2="lt2" tx2="dk2" accent1="accent1"
        accent2="accent2" />
    <p:sldLayoutIdLst>
        <p:sldLayoutId id="2147483649" r:id="rId1"/>
        <p:sldLayoutId id="2147483650" r:id="rId2"/>
    </p:sldLayoutIdLst>
    <p:txStyles>
        <p:titleStyle>
            ...
        </p:titleStyle>
        <p:bodyStyle>
            ...
        </p:bodyStyle>
    </p:txStyles>
</p:sldMaster>
```

Powyżej przedstawiono przykładowy wygląd pliku XML slajdu głównego.
Element:

- <p:cSld> (Common Slide Data) - określa zawartość slajdu,
- <p:clrMap> (Color Scheme Map) - definiuje schemat kolorów; atrybuty "bg1" oraz "tx1" mówią odpowiednio o kolorze tła i kolorze teksu, a atrybuty "accent" określają akcenty kolorystyczne na slajdzie,
- <p:sldLayoutIdLst> (List of Slide Layouts) - zawiera listę definicji układów w slajdzie głównym,
- <p:txStyles> (Slide Master Text Styles) - opisuje styl tekstu dla tytułu oraz głównej części slajdu.

3.2.3 Szablon slajdu

Szablon slajdu (ang. Slide Layout) definiuje styl tekstu oraz położenie elementów na slajdzie. Jego graficzne przedstawienie w programie PowerPoint pokazano na rysunku 3.



Rysunek 3: Szablony w programie PowerPoint.

Dostępnych jest 9 szablonów, które użytkownik może wykorzystać przy tworzeniu poszczególnych slajdów prezentacji.

Przykładowy wygląd pliku XML, w którym zawarta jest definicja szablonu, przedstawiono poniżej.

```
<p:sldLayout>
    <p:cSld name="Slajd tytulowy">
        <p:spTree>
            ...
            <p:sp>
                <p:nvSpPr>
                    <p:cNvPr id="2" name="Tytul 1">
                        ...
                    </p:cNvPr>
                    ...
                </p:nvSpPr>
                <p:txBody>
                    ...
                    <a:p>
                        <a:r>
                            ...
                            <a:t>Kliknij, aby edytowac</a:t>
                        </a:r>
                    </a:p>
                </p:txBody>
            </p:sp>
        </p:spTree>
```

```
    </p:cSld>
    <p:clrMapOvr>
        <a:masterClrMapping/>
    </p:clrMapOvr>
</p:sldLayout>
```

Element z pliku XML:

- <p:cSld> (Common Slide Data) - określa zawartość slajdu; jego atrybut "name" określa nazwę szablonu. Widoczna jest ona na rysunku 3,
- <p:spTree> (Shape Tree) - opisuje wszystkie obiekty, które mają kształt; może zawierać wiele zgrupowanych elementów,
- <p:sp> (Shape) - określa pojedynczy obiekt (kształt), może to być np. pole tekstowe lub obrazek,
- <p:nvSpPr> (Non-Visual Properties for a Shape) - określają właściwości kształtu, które nie mają wpływu na jego wyświetlanie, w przykładzie użyty został on do nadania nazwy kształtowi,
- <p:txBody> (Shape Text Body) - określa definicję ciała tekstu, który ma znaleźć się na slajdzie,
- <a:p> (Text Paragraphs) - określa paragraf tekstu,
- <a:r> (Text Run) - określa pojedynczą część tekstu,
- <a:t> (Text) - określa tekst (tutaj jest to tekst z szablonu),
- <p:clrMapOvr> (Color Scheme Map Override) - określa jaki schemat kolorów ma zostać użyty (tutaj kolejny element <a:masterClrMappig/> oznacza, że powinien zostać użyty schemat ze slajdu głównego).

3.2.4 Slajd

Slajd (ang. Slide) jest podstawowym elementem prezentacji. Zawiera on komponenty i definicję pojedynczego slajdu. Poniżej przedstawiono przykładowy plik XML dla pojedynczego slajdu.

```
<p:sld>
    <p:cSld>
        <p:spTree>
            ...
        </p:spTree>
```

```
    </p:cSld>
    <p:clrMapOvr>
        ...
    </p:clrMapOvr>
    <p:transition>
        ...
    </p:transition>
    <p:timing>
        ...
    </p:timing>
</p:sld>
```

Elementy nieopisane w poprzednich rozdziałach:

- <p:sld> (Presentation Slide) - określa instancję slajdu. Zawarte są w nim wszystkie obiekty, które stanowią ciało slajdu,
- <p:transition> (Slide Transition for a Slide Layout) - określa sposób przejścia pomiędzy kolejnymi slajdami,
- <p:timing> (Slide Timing Information for a Slide Layout) - zawiera informację o czasach animacji wszystkich elementów w danych slajdzie.

3.2.5 Zawartość slajdu

Jak opisano w poprzednich rozdziałach, pojedynczy slajd składa się z wielu obiektów, które zgrupowane są w elementach <p:cSld> (Common Slide Data) oraz <p:spTree> (Shape Tree). Najważniejsze ich składowe to:

1. Tekst (ang. Text Run)

   Tekst reprezentowany jest przez element <a:r> i jest on najmniejszym elementem separacji tekstu. Jeśli chcemy, by część tekstu była pogrubiona, wtedy jest on dzielony na dwa elementy (Text Run) i za pomocą elementu <a:rPr> (Text Run Properties) ustawiane są odpowiednie atrybuty.

```
<p:txBody>
    <a:p>
        <a:r>
            <a:rPr lang="pl-PL"/>
            <a:t>normalny</a:t>
        </a:r>
        <a:r>
            <a:rPr lang="pl-PL" b="1"/>
```

```
            <a:t>pogrubiony</a:t>
        </a:r>
    </a:p>
</p:txBody>
```

Atrybut "lang" oznacza język tekstu, a "b" - tekst pogrubiony. Dostępne są też inne atrybuty, takie jak np. "i" - italic, "strike" - przekreślenie tekstu czy "sz" - rozmiar czcionki. W elemencie <a:t> znajduje się właściwy tekst.

2. Paragraf (ang. Text Paragraph)

   Paragraf reprezentowany jest przez element <a:p> i składa się z elementów Text Run. Elementem <a:pPr> można określić w jaki sposób tekst ma być formatowany np. poprzez wyrównanie czy ustawienie marginesu.

```
<p:txBody>
    <a:p>
        <a:pPr algn="r"/>
        <a:r>
            <a:rPr lang="pl-PL"/>
            <a:t>Tekst wyrownany do prawej strony</a:t>
        </a:r>
    </a:p>
    <a:p>
        <a:r>
            <a:rPr lang="pl-PL"/>
            <a:t>Tekst wyrownany do lewej strony (wartosc domyslna)</a:t>
        </a:r>
    </a:p>
</p:txBody>
```

W przedstawionym fragmencie atrybut "algn" ustawiony na "r" oznacza wyrównanie tekstu do prawej strony. Domyślnie tekst wyrównywany jest do lewej, w takim przypadku odpowiedni atrybut jest pomijany (drugi paragraf w przykładzie).

3. Obrazki

   Obrazek w prezentacji oznacza się elementem <p:pic> (Picture), natomiast element <p:blipFill> (Picture Fill) definiuje jego zawartość. W przykładzie poniżej, element <a:blip> (Blip - binary large image or picture) poprzez referencję określa konkretny obrazek. <a:stretch> (Stretch) informuje, że obrazek powinien zostać rozciągnięty, tak by wypełniał cały przypisany mu obszar. <p:spPr> (Shape Properties) określają właściwości obrazka, w tym przypadku jego przesunięcie względem początku układu współrzędnych

slajdu (czyli górnego lewego rogu) - <a:off> (Offset) oraz jego długość i szerokość - <a:ext> (Extend). Obie właściwości znajdują się w elemencie <a:xfrm> (Graphic Frame Transform), który określa transformację kształtu.

```
<p:spTree>
    <p:pic>
        ...
        <p:blipFill>
            <a:blip r:embed="rId2"/>
            <a:stretch>
                <a:fillRect/>
            </a:stretch>
        </p:blipFill>
        <p:spPr>
            <a:xfrm>
                <a:off x="1686757" y="3427261"/>
                <a:ext cx="2789808" cy="2789808"/>
            </a:xfrm>
            ...
        </p:spPr>
    </p:pic>
</p:spTree>
```

4. Tabele

   Głównym elementem tabeli jest <a:tbl> (Table). Element <a:tblGrid> (Table Grid) definiuje m.in. rozmiar kolumn. Tabela opisana jest poprzez wiersze - <a:tr> (Table Row), które posiadają atrybut "h" mówiący o ich wysokościach oraz z kolumn <a:tc> (Table Column). Wewnątrz kolumn zawarta jest właściwa część tabeli.

   Przykład poniżej prezentuje kod XML tabeli oraz jego graficzną reprezentację w programie PowerPoint.

```
<a:tbl>
    <a:tblGrid>
        <a:gridCol w="4064000"/>
        <a:gridCol w="4064000"/>
    </a:tblGrid>
    <a:tr h="370840">
        <a:tc>
            ...
        </a:tc>
        <a:tc>
```

```
            ...
        </a:tc>
    </a:tr>
    <a:tr h="370840">
        <a:tc>
            ...
        </a:tc>
        <a:tc>
            ...
        </a:tc>
    </a:tr>
</a:tbl>
```

Wiersz 1	Wiersz 1
Wiersz 2	Wiersz 2

Rysunek 4: Wygląd tabeli w programie PowerPoint.

Jeśli zawartością kolumny jest tekst, to zawiera ona element <a:txBody> (Text Body), a w nim opisane w poprzednich punktach elementy Text Paragraph.

5. Formuły matematyczne

   Wszystkie formuły matematyczne umieszczone w prezentacji pochodzą ze wspólnego języka znaczników - Office Math Markup Language (OMML). Jest on używany również przez dokumenty tekstowe oraz arkusze kalkulacyjne. Można nim wyrażać m.in.: wzory i wyrażenia matematyczne, macierze czy symbole matematyczne.

   Formuły umieszczane są w elemencie Text Paragraph. Element <m:oMath> (Office Math) oznacza pojedynczą instancję matematyczną i może się on znajdować w elemencie <m:oMathPara> (Office Math Paragraph). Oznacza to, że wyrażenie matematyczne nie powinno być umieszczane bezpośrednio w tekście, a w osobnej linii.

   Elementy Office Math to kombinacje elementów <m:r> (Run) oraz obiektów i funkcji. Poniżej przedstawiono najprostszy przykład wzoru. Z uwagi na umieszczenie go w elemencie Office Math Paragraph, będzie on wyświetlony w nowej linii:

$$j + b$$

   W elemencie <m:oMathParaPr> (Office Math Paragraph Properties) ustawione jest wyrównanie tekstu <m:jc> (Justification) oznaczające jego wyśrodkowanie.

```
<m:oMathPara>
    <m:oMathParaPr>
        <m:jc m:val="centerGroup"/>
    </m:oMathParaPr>
    <m:oMath>
        <m:r>
            ...
            <m:t>j+b</m:t>
        </m:r>
    </m:oMath>
</m:oMathPara>
```

Specyfikacja wspiera wszystkie możliwe funkcje matematyczne. Przykładem może być potęga, reprezentowana poprzez element <m:sSup> (Superscript Object), który składa się z elementu głównego <m:e> (Element) - podstawy oraz z <m:sup> (Superscript) - wykładnika potęgi.

Poniżej przedstawiono przykład dla wyrażenia: r^2.

```
<m:sSup>
    ...
    <m:e>
        <m:r>
            ...
            <m:t>r</m:t>
        </m:r>
    </m:e>
    <m:sup>
        <m:r>
            ...
            <m:t>2</m:t>
        </m:r>
    </m:sup>
</m:sSup>
```

3.3 TEX

TEX jest systemem profesjonalnego składu drukarskiego. Powstał on w roku 1985 za sprawą Donalda E. Knutha, który pisząc swoją książkę był niezadowolony z efektów jakie uzyskiwał i postanowił napisać własny program do składu tekstu [14].

Obecnie TEX używany jest w wielu profesjonalnych zastosowaniach, jak np. w publikacjach naukowych. Odznacza się przejrzystością oraz wysoką jakością prezentowania treści. Umożliwia pisanie skomplikowanych formuł matematycznych, ma wsparcie do tworzenia skorowidzów, spisów ilustracji, tworzenia bibliografii i wielu innych.

TEX nie jest systemem typu WYSIWYG, co oznacza, że użytkownik musi napisać "kod", który "kompilowany" jest do wersji ostatecznej. Kod ten, składa się z wielu znaczników i komend sterujących. Może on być modyfikowany w dowolnym edytorze tekstowym, przez co jest łatwo przenośny na inne platformy.

TEX jest oprogramowaniem otwarto-źródłowym. Dzięki makrom, każdy użytkownik może do niego dodać własny pakiet, z którego mogą korzystać inni.

3.3.1 LATEX

LATEX jest rozwinięciem systemu TEX o zestaw makr, które w znaczący sposób ułatwiają jego używanie. Jest tak popularny, że często słowa LATEX i TEX używane są zamiennie. Jego stworzenie zawdzięczamy amerykańskiemu informatykowi Leslie Lamportowi [15].

Poniżej przedstawiono przykładowy kod źródłowy w języku LATEX oraz jego graficzną reprezentację po kompilacji do formatu PDF (rysunek 5).

```
\documentclass[11pt,a4paper]{article}
\usepackage{polski}
\usepackage[utf8]{inputenc}
\usepackage{hyperref}
\usepackage{graphicx}
\title{Dżuma}
\author{Karol Stasiak}
\date{Luty 2020}
\begin{document}
\maketitle
\section{Wstęp}
Dżuma jest ostrą chorobą zakaźną wywoływaną przez bakterie Yersinia pestis, zwane
    prościej pałeczkami dżumy. Jedną z dróg infekcji są ugryzienia zarażonych pcheł
    lub samych zwierząt \cite{dzuma}.
\begin{figure}[h!]
\centering
\includegraphics[scale=0.5]{dzuma.jpg}
\caption{Ubiór ochronny lekarza podczas epidemii dżumy.}
\label{fig:outfit}
\end{figure}
\section{Czy dżuma nadal występuje?}
Tak. Mongolska służba zdrowia donosi o śmierci 15-letniego chłopca, który zmarł z
```

```
    powodu zakażenia bakteriami dżumy \cite{mongolia}.
\begin{thebibliography}{9}
\bibitem{dzuma}
\href{https://www.medonet.pl/zdrowie,dzuma-wciaz-grozna--oto--co-powinnismy-wiedziec-o
-tej-chorobie,artykul,1724417.html}{Czym jest dzuma?}
\bibitem{mongolia}
\href{https://www.o2.pl/informacje/czarna-smierc-wrocila-w-azji-kolejna-ofiara-wladze-
boja-sie-powrotu-epidemii-6531922690734816a}{"Czarna śmierć" wróciła w Azji. Kolejna
    ofiara.}
\end{thebibliography}
\end{document}
```

Dżuma

Karol Stasiak

Luty 2020

1 Wstęp

Dżuma jest ostrą chorobą zakaźną wywoływaną przez bakterie Yersinia pestis, zwane prościej pałeczkami dżumy. Jedną z dróg infekcji są ugryzienia zarażonych pcheł lub samych zwierząt [1].

Rysunek 1: Ubiór ochronny lekarza podczas epidemii dżumy.

2 Czy dżuma nadal występuje?

Tak. Mongolska służba zdrowia donosi o śmierci 15-letniego chłopca, który zmarł z powodu zakażenia bakteriami dżumy [2].

Literatura

[1] Czym jest dżuma?

[2] "Czarna śmierć" wróciła w Azji. Kolejna ofiara.

Rysunek 5: Wygląd dokumentu po kompilacji.

Powyższy przykład pokazuje, że kod składa się z wielu znaczników. Budowane są one wg schematu:

```
\nazwa[opcje]{argument}
```

Komenda `\documentclass` określa klasę dla dokumentu, która wpływa na formatowanie całego tekstu. W tym przypadku jest to artykuł. Inne dostępne opcje to np.: `report`, `book` czy `beamer`.

Komenda `\usepackage` używana jest do definiowania dodatkowych pakietów, jakie użytkownik chciałby użyć, np. obsługa polskich znaków diakrytycznych, dodawanie obrazków czy używanie linków.

`\title`, `\author`, `\date` używane są do definicji podstawowych informacji, które będą umieszczone jako tytuł artykułu za sprawą komendy `\maketitle`.

Właściwą część dokumentu wyznaczają komendy `\begin` oraz `\end` z argumentem `document`, określając w ten sposób zakres tego środowiska. W środku możemy znaleźć znaczniki `\section`, które wyznaczają kolejne sekcje artykułu. Na ich podstawie możliwe jest automatyczne stworzenie spisu treści.

Wszelkie cytowania stworzone zostały z pomocą komendy `\cite`, która odnosi się przez nazwę do definicji cytowanego fragmentu zawartą w środowisku `thebibliography`.

Środowisko `figure` zostało użyte do dodania obrazka do artykułu. Zawarte są w nim różne definicje jak podpis obrazka (`\caption`) czy określenie jaki konkretnie plik graficzny powinien zostać umieszczony (`\includegraphics`).

3.3.2 BEAMER

BEAMER jest klasą dokumentów używaną do tworzenia prezentacji multimedialnych z użyciem systemu LaTeX. Powstał on w 2003 roku za sprawą Tilla Tantau [16], który użył go do prezentacji swojej pracy doktorskiej. Następnie umieścił go jako pakiet w repozytorium CTAN[8] i od tej pory jest dostępny dla każdego [17]. BEAMER jest chętnie używany przez naukowców do tworzenia prezentacji (np. na potrzeby konferencji), ale również przez innych użytkowników, którzy pragną, by ich prezentacje wyglądały ładnie i były łatwo utrzymywalne.

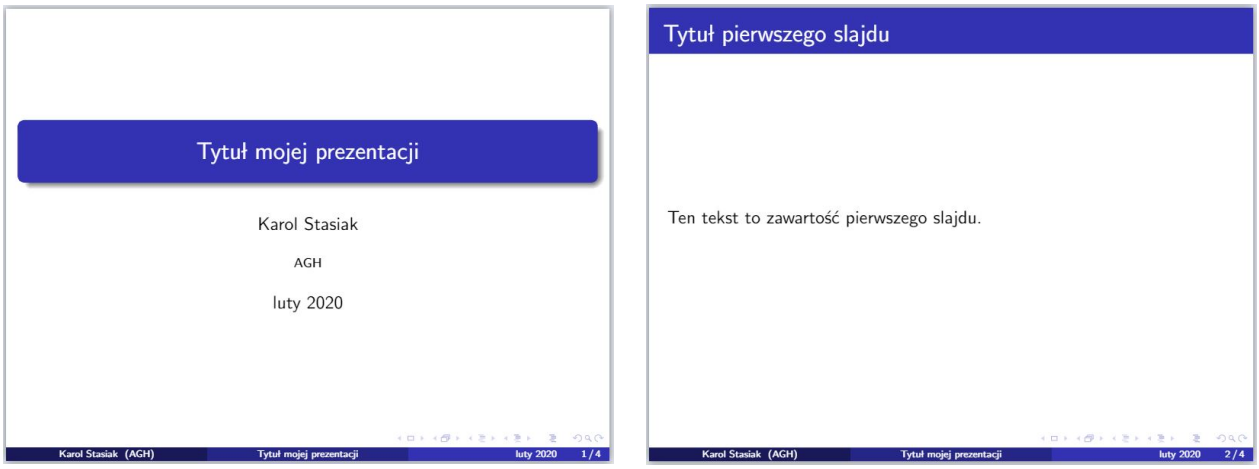
Schemat tworzenia prezentacji z użyciem BEAMERA jest bardzo podobny do tworzenia zwykłych dokumentów w systemie LaTeX. Wszystkie występujące tam elementy, dostępne są również tutaj. Same slajdy tworzy się z użyciem specjalnych środowisk.

Najważniejszym środowiskiem jest `frame`. Definiuje się w nim zawartość pojedynczego slajdu, co prezentuje przykład poniżej. Rysunek 6 przedstawia wynik kompilacji.

```
\documentclass{beamer}
\usepackage[utf8]{inputenc}
\usetheme{Madrid}
```

[8] CTAN (ang. The Comprehensive TeX Archive Network) - m.in. repozytorium pakietów TeX

```
\title{Tytuł mojej prezentacji}
\author{Karol Stasiak}
\institute{AGH}
\date{luty 2020}

\begin{document}
\frame{\titlepage}
\begin{frame}
    \frametitle{Tytuł pierwszego slajdu}
    Ten tekst to zawartość pierwszego slajdu.
\end{frame}
\end{document}
```



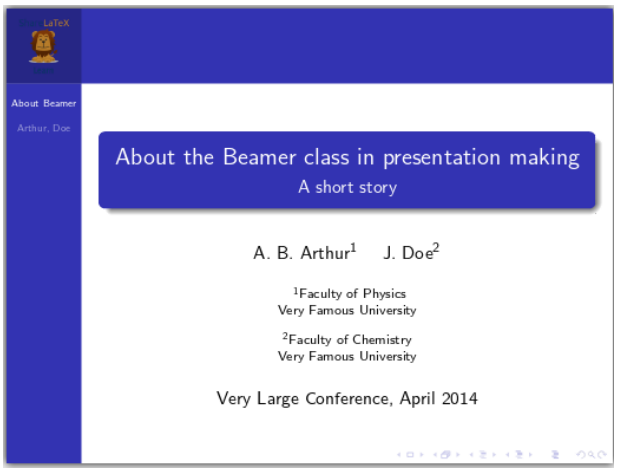
(a) Slajd 1.

(b) Slajd 2.

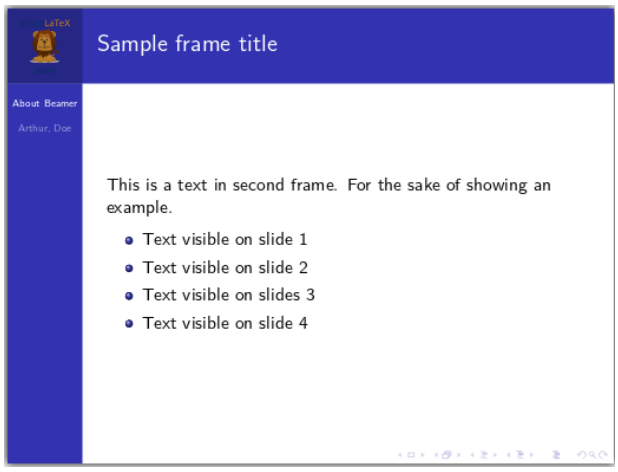
Rysunek 6: Slajdy po konwersji do formatu PDF.

Tak jak w LaTeXu, w BEAMERZE możliwe jest zdefiniowanie informacji dla slajdu tytułowego (rysunek 6a).

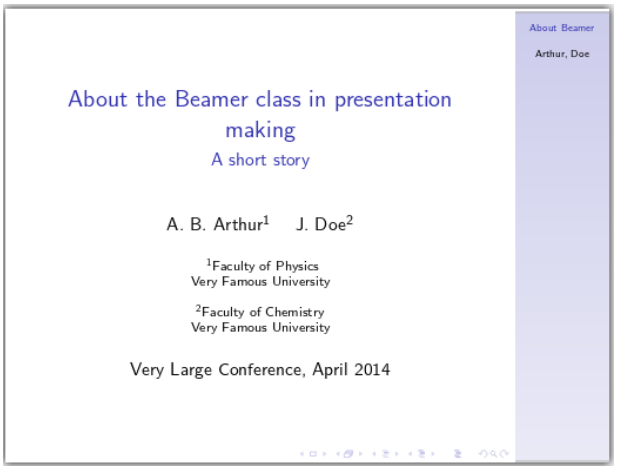
Za pomocą komendy `\usetheme` można zdefiniować formatowanie zawartości slajdu. Schemat kolorów można natomiast zmienić za pomocą komendy `\usecolortheme`. Oprócz użytego w powyższym przykładzie schematu `Madrid`, dostępnych jest kilka predefiniowanych formatów, część z nich pokazano na rysunku 7.



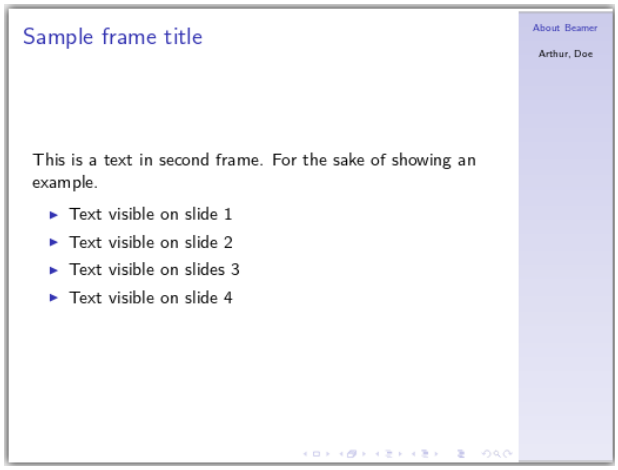
(a) Slajd tytułowy - PaloAlto.



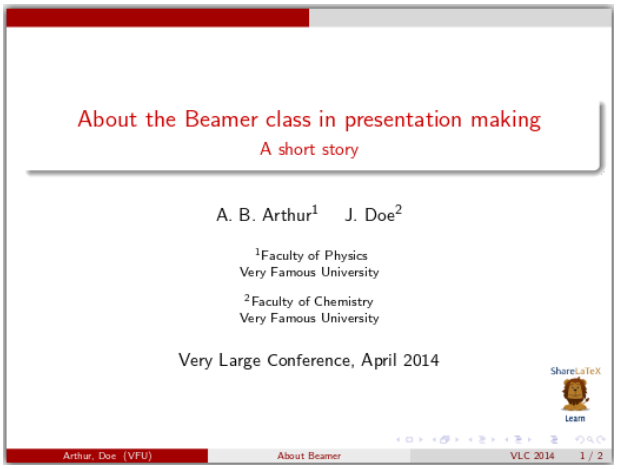
(b) Slajd przykładowy - PaloAlto.



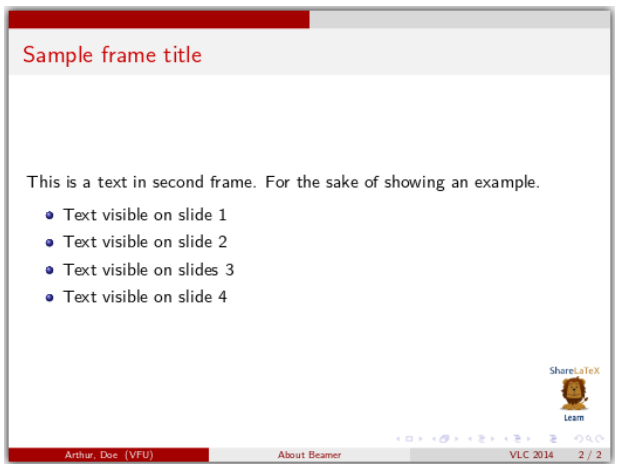
(c) Slajd tytułowy - Goettingen.



(d) Slajd przykładowy - Goettingen.



(e) Slajd tytułowy - CambridgeUS.

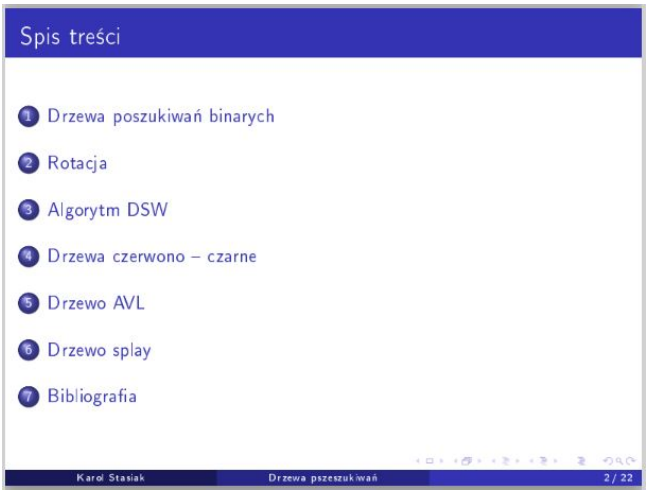


(f) Slajd przykładowy - CambridgeUS.

Rysunek 7: Przykłady dostępnych motywów.
Źródło: overleaf.com

Dla prezentacji tworzonych z wykorzystaniem BEAMERA możliwe jest automatyczne stworzenie spisu treści slajdów. Aby to zrobić należy utworzyć slajd zawierający znacznik `\tableofcontents`. Wynikiem będzie slajd widoczny na rys. 8. Aby odnośnik do zawartości znalazł się w spisie treści, musi być zawarty w znaczniku `\section`.

```
\begin{frame}{Spis treści}
\tableofcontents
\end{frame}
```

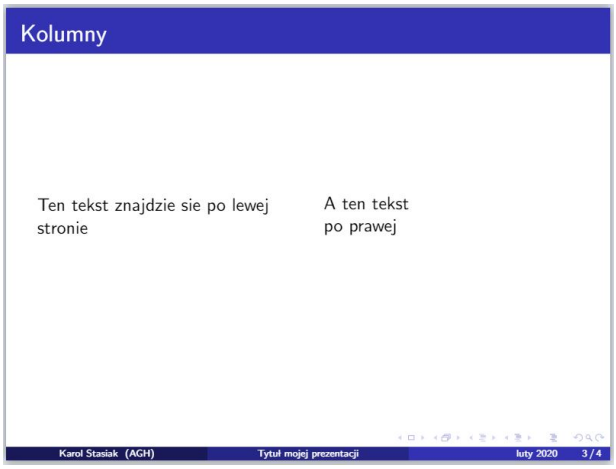


Rysunek 8: Przykładowy spis treści.

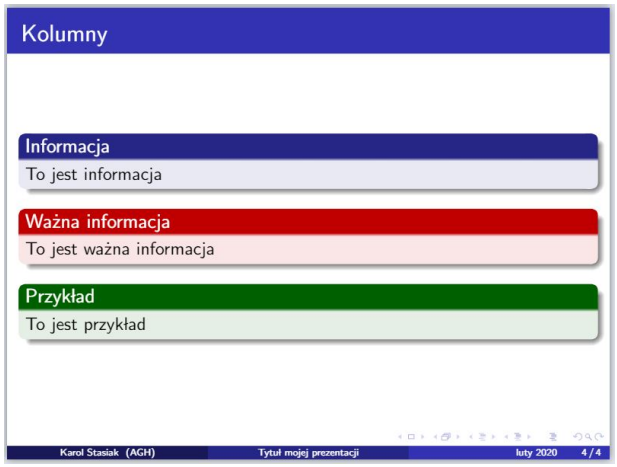
Czcionka dla całej prezentacji może być zmieniona poprzez użycie komendy `\usefonttheme` lub `\usepackage` (zmiana całej rodziny). Modyfikowalny jest również rozmiar czcionki, do czego służą opcje komendy `\documentclass`: `\documentclass[15pt]{beamer}` zmieni rozmiar czcionki na 15 (z domyślnego 11).

W pakiecie BEAMER zawarte są użyteczne środowiska służące do dzielenia zawartości slajdu wertykalnie (`columns`) oraz horyzontalnie (`block`). Poniżej przedstawiono w jaki sposób działa środowisko `columns`, gdzie argument `c` oznacza wyśrodkowanie zawartości. Znacznik `column` określa na ile kolumn ma zostać podzielony slajd a jego argument określa ile miejsca ma zajmować jedna kolumna. Wynik uzyskany dla poniższego kodu przedstawiono na rysunku 9a.

```
\begin{frame}

    \frametitle{Kolumny}
    \begin{columns}[c]
        \column{.45\textwidth}
            Ten tekst znajdzie się po lewej stronie
        \column{.45\textwidth}
            A ten tekst \newline po prawej
    \end{columns}

\end{frame}
```



(a) Podział wertykalny.



(b) Podział horyzontalny.

Rysunek 9: Podział slajdu na części.

Podział slajdu na horyzontalne bloki przedstawiono na rysunku 9b. Dzięki środowiskom `block`, `alertblock` oraz `exampleblock` możliwe jest wyróżnienie pewnych ważnych informacji. Poniżej znajduje się kod źródłowy przytoczonego przykładu.

```
\begin{frame}
    \frametitle{Kolumny}
    \begin{block}{Informacja}
        To jest informacja
    \end{block}

    \begin{alertblock}{Ważna informacja}
         To jest ważna informacja
    \end{alertblock}

    \begin{exampleblock}{Przykład}
        To jest przykład
    \end{exampleblock}
\end{frame}
```

3.3.3 Pakiety

We wspomnianym już repozytorium CTAN znajduje się prawie 6 tysięcy ogólnodostępnych pakietów[18], dających mnóstwo dodatkowych możliwości określania stylu tekstu (i nie tylko) ponad to, co oferuje sam LaTeX. Najważniejsze spośród pakietów używanych w utworzonej aplikacji to:

1. `inputenc`
   Konwersja kodu TeX wymaga, by ten zakodowany był w ASCII[9]. Pakiet `inputenc` umożliwia konwersję znaków (w tym polskich), tak by były zrozumiałe dla silnika konwersji. W jego opcjach definiuje się w jakim standardzie został napisany kod źródłowy (najczęściej jest to UTF-8[10]).

2. `fontenc`
   Pakiet ten umożliwia wybór czcionki, jaką będą wyświetlane litery w końcowym dokumencie. Dzięki opcji `T1` prawidłowo będą wyświetlane znaki z akcentem np. `ó`.

3. `ragged2e`
   Dzięki niemu możliwe jest wyrównywanie tekstu. Możliwe jest to dzięki nowym środowiskom: `Center`, `FlushLeft`, `FlushRight`. Dostępne są również równoważne komendy.

4. `underscore`
   Dzięki temu pakietowi możliwe jest pisanie znaku podkreślenia `'_'` w tekście bez konieczności poprzedzania go znakiem `'\'`.

5. `calc`
   Dodaje on możliwość prostych obliczeń w kodzie LaTeX. Przykładem jest użycie komendy `\setlength` do ustawienia szerokości marginesu.

6. `listings`
   Pakiet ten dodaje możliwość formatowania kodu źródłowego. Wspiera on wiele języków programowania a sam sposób kolorowania składni jest łatwo konfigurowalny. Kod można dodawać bezpośrednio w TeX lub importować z innych plików.

7. `anyfontsize`
   Dzięki niemu możliwa jest zmiana rozmiaru tekstu. Dodaje on komendę `\fontsize`, w której podaje się rozmiar czcionki, a komendą `\selectfont` oznacza początek tekstu.

8. `multirow`
   Daje możliwość utworzenia w tabeli komórki, która będzie scalać kilka wierszy.

[9] ASCII - (ang. American Standard Code for Information Interchange) - siedmiobitowy system kodowania znaków, szeroko używany w systemach informatycznych

[10] UTF-8 - (ang. 8-bit Unicode Transformation Format) - system kodowania Unicode, wykorzystujący od 1 do 4 bajtów do zakodowania pojedynczego znaku, w pełni kompatybilny z ASCII

9. `graphicx`

   Rozszerza on możliwości dodawania grafik do prezentacji, m.in. umożliwiając wpisywanie opcji w konwencji klucz-wartość.

10. `amsmath`

    Dostarcza kilka pakietów, które umożliwiają różne sposoby wyświetlania równań matematycznych. Dodaje m.in. środowiska: `align`, `multiline` czy `gather`.

11. `mathtools`

    Jest to zestaw pakietów, które rozszerzają `amsmath` o jeszcze więcej opcji wyświetlania równań np. dodając środowisko `makebox` oraz wiele użytecznych symboli.

12. `xfrac`

    Umożliwia wyświetlanie ułamków w taki ½ oraz taki $^1/_2$ sposób.

13. `enumitem`

    Dzięki niemu możliwe jest formatowanie środowisk `enumerate` oraz `itemize` np. poprzez zmianę znaku wypunktowania.

14. `ulem`

    Umożliwia dekorowanie tekstu, np. w ten sposób: ~~dżuma~~, różyczka, ~~ośpa~~ czy pokrzywka.

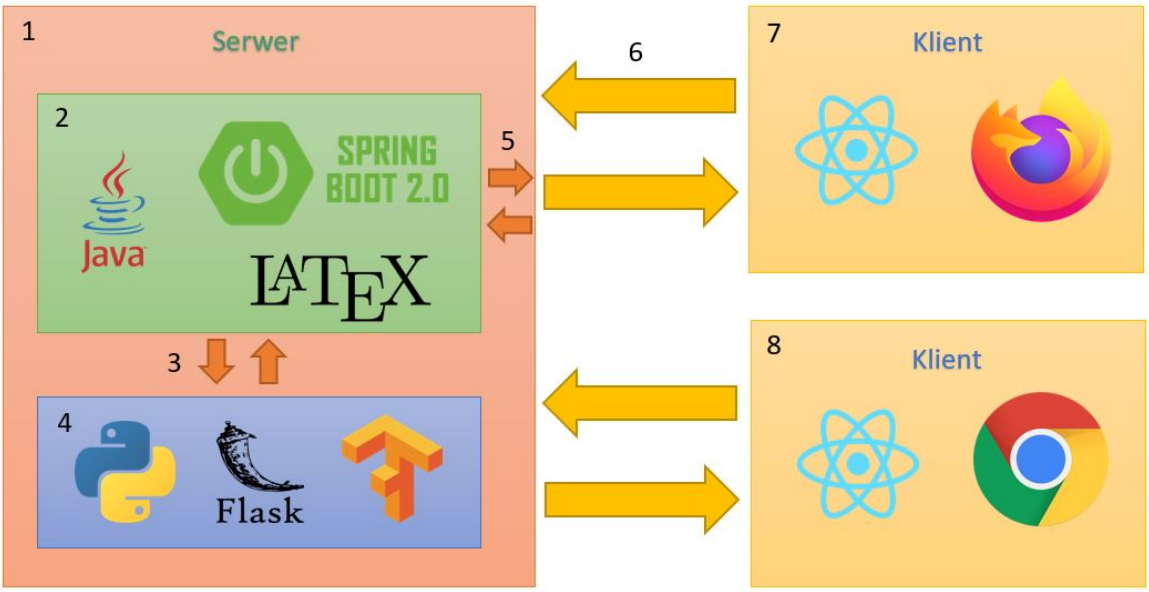
15. `textpos`

    Jest to najważniejszy pakiet używany podczas konwersji. Umożliwia on ustawienie tekstu w dowolnym miejscu na slajdzie. Jest to bardzo ważne, ponieważ w formacie PPTX oraz programie PowerPoint możliwe jest dodawanie pól tekstowych w dowolnym miejscu.

4 Implementacja

4.1 Ogólna architektura

Aplikacja została wykonana w architekturze klient-serwer. Jej graficzna reprezentacja widoczna jest na rysunku 10. Serwer składa się z dwóch składowych: główny element (oznaczony na rysunku numerem 2) to aplikacja dokonująca konwersji. Drugim elementem jest aplikacja odpowiedzialna za rozpoznawanie kodu źródłowego (numer 4 na rysunku). Do serwera zapytania kierują klienci oznaczeni numerami 7 i 8.



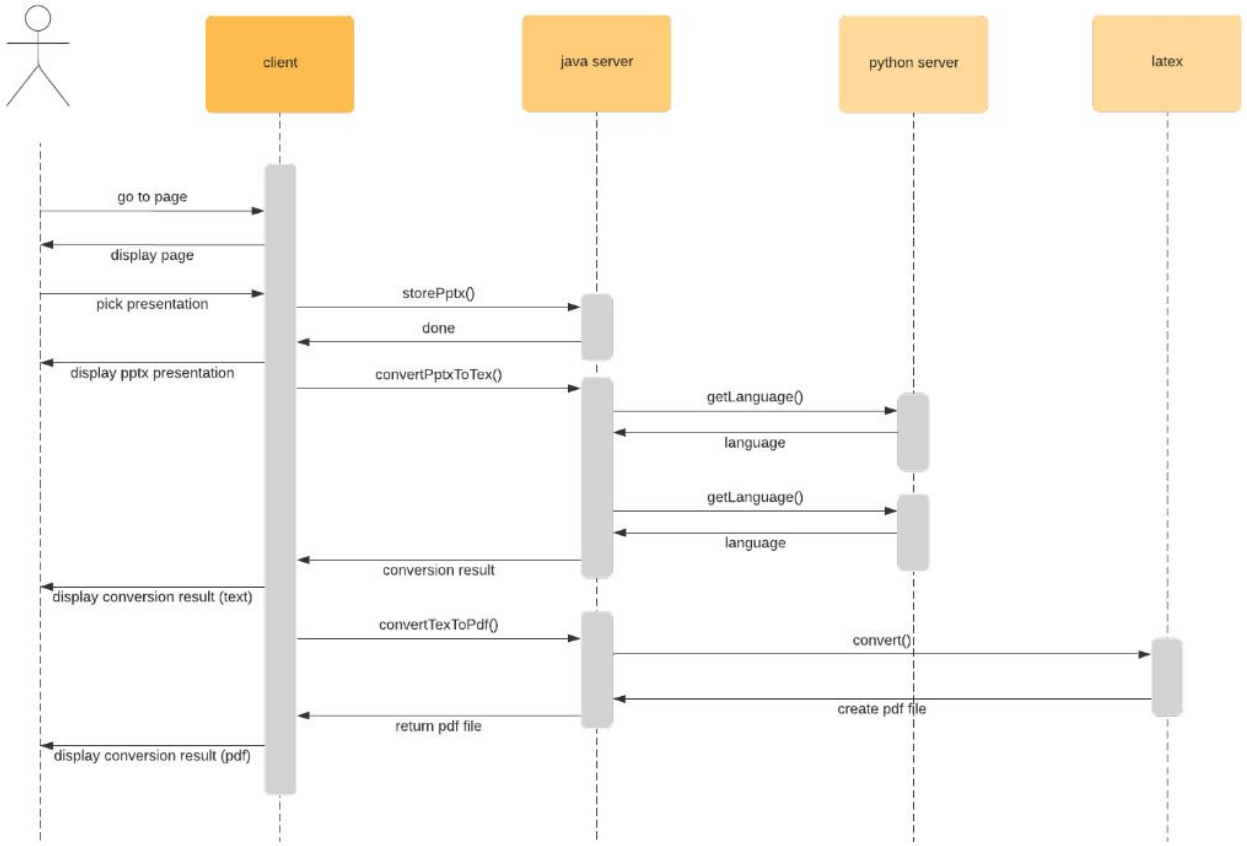
Rysunek 10: Architektura aplikacji.

4.2 Serwer

4.2.1 Spring Boot

Za główną część pracy, czyli mechanizm konwersji, odpowiada serwer aplikacyjny napisany w języku Java z użyciem szkieletu tworzenia aplikacji jakim jest Spring Boot [6]. Serwer ten:

- obsługuje wszystkie żądania od klientów,
- przeprowadza konwersje pliku PPTX na kod TEX,
- używa systemu plików do zapisywania i dalszego udostępniania wyników konwersji,
- komunikuje się z serwerem odpowiedzialnym za rozpoznawanie kodu źródłowego,
- zleca i nadzoruje konwersję pliku TEX na PDF.



Rysunek 11: Diagram sekwencji użycia.

Na rysunku 11 przedstawiono przykładowy diagram użycia aplikacji. Użytkownik wchodząc na stronę musi wybrać plik PPTX, który ma zostać poddany konwersji. Klient wysyła żądanie do serwera na adres `/storePptx`, załączając prezentację. Żądanie jest obsługiwane przez kontroler `ConverterController` w sposób zaprezentowany poniżej.

```
@PostMapping("/storePptx")
public Map<String, String> storePptx(@RequestParam("file") MultipartFile file)
    throws IOException {
    return pptxService.storePptx(file);
}
```

Obsługą zadań związanych z plikami PPTX zajmuje się klasa `PptxService`. Za zapisywanie tych plików na serwerze, odpowiada metoda `storePptx`. Jej ciało przedstawiono poniżej.

```
public Map<String, String> storePptx(MultipartFile file) throws IOException {
    String id = RequestContextHolder.currentRequestAttributes().getSessionId() +
        randomAlphanumeric(5);

    idHolder.setId(id);
    fileSystemService.createDirectory();
    fileSystemService.storeMultipartFile(file, PRESENTATION);
```

```
    logger.info(String.format("Saved presentation with name ---> '%s', id = %s",
        file.getOriginalFilename(), id));

    return ImmutableMap.of("id", id);
}
```

W pierwszej linijce generowany jest unikatowy identyfikator prezentacji, który jednoznacznie będzie ją reprezentować. Następnie jest on wstawiany do klasy `IdHolder`, odpowiedzialnej za jego przechowywanie. Kolejno, metoda `fileSystemService` tworzy katalog o nazwie będącej identyfikatorem prezentacji, w którym będą przechowywane wszystkie pliki związane z daną prezentacją oraz oryginalna prezentacja. Metoda zwraca identyfikator, który jest przechowywany po stronie klienta i wysyłany z każdym kolejnym żądaniem. Aby zoptymalizować ten proces stworzona została specjalna adnotacja `@IdSetter` (oraz aspekt, który ją obsługuje), dzięki której za każdym razem z parametrów żądania wyciągany jest identyfikator i ustawiany w klasie `IdHolder`:

```
@Data
@Component
@Scope(scopeName = SCOPE_REQUEST, proxyMode = ScopedProxyMode.TARGET_CLASS)
public class IdHolder {
    private String id;
}
```

Klasa zawiera jedynie jedno pole - `id`. Adnotacja `@Data` z biblioteki `lombok` zapewnia metody typu `getter`, `setter`, `toString`, `hashCode`, `equals` oraz konstruktor, znacznie zmniejszając ilość potrzebnego kodu.

Klasa ta jest komponentem, co oznacza m.in., że za jej cały cykl życia odpowiada Spring. Adnotacją `@Scope` oznaczono, że z każdym żądaniem na serwer powinien zostać stworzony nowy obiekt tej klasy. Parametr `proxyMode` określa w jaki sposób ma to być osiągnięte (Spring używa do tego proxy).

Klient po otrzymaniu identyfikatora prezentacji wysyła kolejne żądanie, którym jest konwersja. Odpowiada za nią adres `/convertPptxToTex`. Jako parametry przesyłane są opcje konwersji (zostaną one omówione w późniejszych rozdziałach) oraz identyfikator prezentacji. Obsługą zapytań związanych z TEX zajmuje się `TexService`. Poniżej zaprezentowano kod odpowiedzialny za konwersję.

```
public ConversionResult convertPptxToTex() {

    Resource resource = fileSystemService.readFile(PRESENTATION);

    try {
```

```
        logger.info(String.format("Starting conversion with parameters ---> %s",
            options));
        String convertedContent = convertToTex(resource.getInputStream());
        fileSystemService.storeFileWithContent(TEX_FILE, convertedContent);
    } catch (Exception e) {
        logger.warn(resource.getFilename() + " ---> Conversion failed", e);
        conversionResult.addError(e.getMessage());
    }

    return getConversionResult();
}
```

Najpierw, z systemu plików, odczytywana jest zapisana prezentacja, a następnie uruchamiany jest mechanizm konwersji, poprzez wywołanie funkcji `convertToTex`. Zwraca ona kod TEX, który zapisywany jest w odpowiednim pliku i katalogu. Gdyby konwersja się nie udała, błędy zapisywane są w specjalnej klasie `ConversionResult`, która zbiera informacje o błędach i ewentualne ostrzeżenia dotyczące ograniczeń podczas procesu konwersji oraz kod źródłowy. Dodatkowo, logowana jest informacja o rozpoczęciu wykonywania funkcji wraz z parametrami konwersji, jak i wystąpienie ewentualnych błędów.

Klient po otrzymaniu odpowiedzi sprawdza, czy pojawiły się błędy. Jeśli tak, to wyświetlane są one w formie `Toast messages`[11], a sam wynik konwersji kończy się niepowodzeniem. W przeciwnym wypadku klient wysyła kolejne żądanie na serwer, w celu otwarcia pliku TEX, za co odpowiada adres `/getTex`.

```
public ResponseEntity<Resource> getTex() {
    Resource resource = fileSystemService.readFile(TEX_FILE);

    InputStream inputStream;
    try {
        inputStream = resource.getInputStream();
    } catch (IOException e) {
        return ResponseEntity.notFound().build();
    }

    return ResponseEntity.ok()
            .contentType(MediaType.APPLICATION_OCTET_STREAM)
            .header(HttpHeaders.CONTENT_DISPOSITION, String.format("attachment;
                filename=%s", TEX_FILE))
            .body(new InputStreamResource(inputStream));
```

[11] Toast messages - wiadomości typu `push`, które pojawiają się (zazwyczaj na określony czas) na ekranie

```
}
```

Plik odczytywany jest z katalogu określonego przed identyfikator prezentacji i zwracany jest w odpowiedniej postaci.

Kolejnym żądaniem jakie klient wysyła na serwer (w najbardziej podstawowym scenariuszu) jest wygenerowanie pliku PDF z nowo utworzonego kodu TEX. Służy do tego adres `/convertTexToPdf`, a za wykonanie polecenia odpowiedzialna jest metoda `convertTexToPdf` zdefiniowana w klasie `PdfService`, zaprezentowana poniżej:

```
public Map<String, Object> convertTexToPdf() {
    fileSystemService.deleteFile(PDF_FILE);
    String producedFile = PDF_FILE;

    try {
        convertTexToPdf(TEX_FILE);
        boolean convertedWithNoError = convertTexToPdf(TEX_FILE);

        if (!convertedWithNoError) {
            logger.info("TeX File converted with errors");
            boolean contains = fileSystemService.fileContains(LOG_FILE, "no output
                PDF file produced");
            if (contains) {
                logger.warn(format("Converting to pdf %s as %s could not be created",
                    LOG_FILE, PDF_FILE));
                producedFile = LOG_FILE;
                convertLogToPdf(LOG_FILE);
            }
        }
    } catch (Exception e) {
        logger.warn("Unknown exception occurred during conversion TeX -> Pdf", e);
    }
    return ImmutableMap.of("produced", producedFile);
}
```

Metoda usuwa obecny plik PDF (np. wynik poprzedniej konwersji) oraz wywołuje metodę `convertTexToPdf(TEX_FILE)`, która buduje nowy proces odpowiedzialny za komunikację z systemem operacyjnym oraz wywołuje polecenie konwersji `pdflatex` z odpowiednimi parametrami. Dla zapewnienia prawidłowej konwersji metoda `convertTexToPdf(TEX_FILE)` wywoływana jest dwukrotnie. Przekierowuje ona strumień wyjściowy oraz strumień błędów, tak aby możliwy był ich odczyt.

Jeśli podczas konwersji wystąpiły błędy, plik z logami konwertowany jest do formatu PDF.

Wynikiem zapytania jest informacja, czy proces konwersji zakończył się powodzeniem. Dzięki temu, klient może zapytać serwer o ostateczny wynik konwersji, a także wyświetlić odpowiedni plik oraz ewentualne błędy.

Zaprezentowany scenariusz jest jednym z przykładów użycia aplikacji. Inne przypadki użycia i funkcje zostaną omówione w kolejnych rozdziałach pracy.

Ważnym punktem w działaniu serwera jest także jego szybkość. W celu kontroli wydajności został utworzony specjalny aspekt `EventLoggerAspect` oraz adnotacja `@EventLogging`. Każda klasa oraz metoda opatrzona tą adnotacją jest monitorowana pod względem tego, jakie parametry otrzymuje oraz jak długo się wykonuje. Dzięki temu udało się rozpoznać i zoptymalizować wolno działające części systemu.

4.2.2 Flask

Jedną z opcji konwersji jest możliwość automatycznej detekcji kodu źródłowego na slajdach prezentacji PPTX, rozpoznanie jaki jest to język programowania oraz odpowiednie sformatowanie jego składni. Za część dotyczącą detekcji kodu odpowiada aplikacja napisana w języku Python z użyciem szkieletu tworzenia aplikacji internetowych Flask [19]. Korzysta ona z biblioteki `Guesslang`[20] oraz wykorzystuje sztuczną inteligencję. Biblioteka ta za pomocą biblioteki `Tensorflow`[12] została wytrenowana na tysiącach przykładów kodów źródłowych, dzięki czemu wspiera 20 języków programowania [21].

Aplikacja definiuje jeden adres `/code`, który obsługuje żądania. Jako parametr akceptuje tekst, który jest przepuszczany przez wyszkolony model. W odpowiedzi model zwraca posortowaną względem prawdopodobieństwa listę języków programowania.

4.3 Konwersja

Mechanizm konwersji prezentacji w formacie PPTX na kod źródłowy w TEX został zaprojektowany w taki sposób, by był łatwo rozszerzalny. Główną klasą odpowiedzialną za konwersję jest klasa `SlideShowConverter`. Korzysta ona z różnych klas konwerterów, które rozszerzają interfejs `Converter`.

```
public interface Converter {
    List<Shape> convert(XSLFSlide slide, int slideNumber, List<XSLFShape>
        allSlideShapes);

    boolean isEnabled();
}
```

[12] Tensorflow - biblioteka programistyczna wykorzystywana w uczeniu maszynowym i głębokich sieciach neuronowych

Klasa `SlideShowConverter` zawiera metodę `convert` oraz metodę `isEnabled`. Druga z nich odpowiada za to, czy dany konwerter powinien zostać użyty (na podstawie wybranych przez użytkownika opcji). Metoda `convert` otrzymuje obiekt `slide`, reprezentujący dany slajd, jego numer oraz wszystkie kształty, takie jak pola tekstowe czy obrazki, znajdujące się na slajdzie. Zwracany natomiast jest obiekt klasy `Shape`, a dokładniej jej konkretnej implementacji, np. `PictureShape` czy `TextShape`. Klasa definiuje metodę `toTex`, która opakowuje wynik konwersji w odpowiedni blok. Blok ten, odpowiada za umiejscowienie wyniku na slajdzie. Poniżej znajduje się definicja wspomnianej klasy.

By nowy konwerter mógł zostać użyty, wystarczy że znajdzie się w ścieżce klas (`classpath`), a Spring sam znajdzie wszystkie implementacje i dostarczy je głównej klasie konwertującej.

```
@AllArgsConstructor
public abstract class Shape {
    private double x, y, width;

    public String toTex() {
        return
            format("\\begin{textblock*}{%.1fmm}(%.1fmm,%.1fmm)\n%s\n\\end{textblock*}",
            width, x, y, getContent());
    }

    public abstract String getContent();
}
```

Mechanizm konwersji na początku odczytuje rozmiary prezentacji (wszystkie dane będą odpowiednio skalowane) oraz na podstawie wybranych opcji włącza odpowiednie konwertery. Tworzy on obiekt klasy `XMLSlideShow`, z której będą pobierane wszystkie informacje o prezentacji.

Program przechodzi w pętli po wszystkich slajdach, za każdym razem uruchamiając konwertery, zbierając w ten sposób wyniki konwersji (czyli obiekty klasy `Shape`) i wpisuje je do obiektu klasy `Slide`. Na koniec, uruchamiana jest metoda `toTex` z klasy `SlideShowConverter`, która na podstawie wybranych przez użytkownika opcji oraz wyników konwersji tworzy ostateczny kod TEX. Metoda, po dodaniu nagłówka i definicji pakietów, zwraca wynik konwersji.

Podczas konwersji użytkownik ma do wyboru opcje:

1. Wybór stylu prezentacji

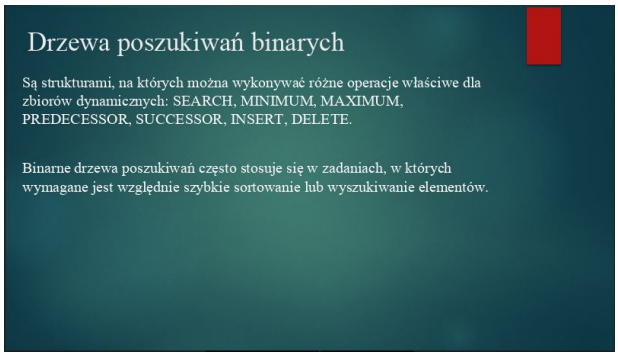
   Wybór dotyczy motywów opisanych w rozdziale 3.3.2 oraz zaprezentowanych na rysunku 7. Dostępne są opcje: default, Madrid, AnnArbor, Antibes, Bergen, Berkeley, Boadilla, Copenhagen, Darmstadt, Goettingen, PaloAlto, Szeged, Warsaw. W zależności od stylu, ustawiany jest inny początek układu współrzędnych slajdu.

2. Wybór koloru stylu

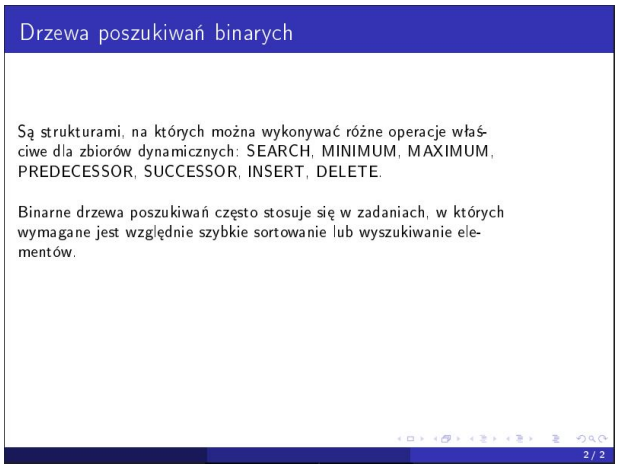
   Dotyczy motywu kolorystycznego użytego razem z głównym stylem prezentacji. Dostępne opcje to: default, beaver, beetle, seahorse i wolverine.

3. Rozpoznawanie tytułów slajdów

   By pole tekstowe zostało rozpoznane jako tytuł slajdu, musi być ono typu `TITLE` oraz zawierać tylko jeden paragraf tekstu. Jeśli tak jest, zostanie ono oznaczone znacznikiem `frametitle`. Na rysunku 12 przestawiono przykład takiej konwersji.



(a) Wersja PPTX.

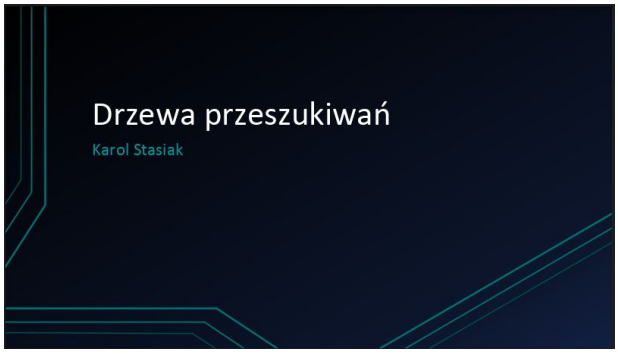


(b) Wersja TEX po konwersji.

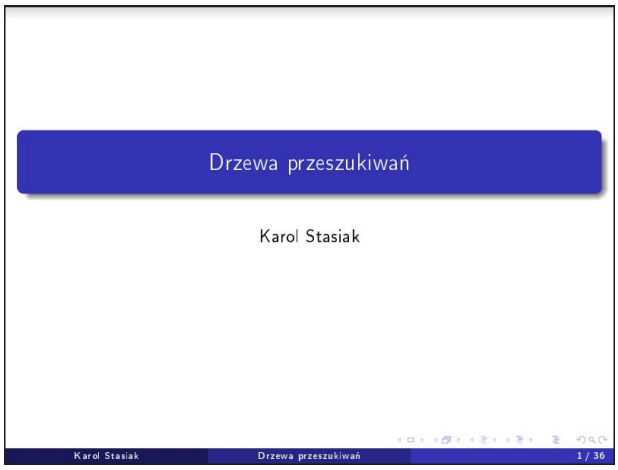
Rysunek 12: Przykład rozpoznawania tytułu slajdu.

4. Dodanie slajdu tytułowego

   By slajd w formacie PPTX został uznany za slajd tytułowy musi zawierać dokładnie dwa pola tekstowe, układ slajdu musi mieć typ `TITLE`, długość tekstu w pierwszym polu tekstowym (tytuł) nie może być dłuższa niż 80 znaków a długość tekstu w drugim polu tekstowym (autor) nie może być dłuższa niż 90 znaków. Jeżeli zajdą wszystkie wymienione warunki, tworzony jest specjalny, dodatkowy slajd tytułowy, jak zaprezentowano na rysunku 13.



(a) Wersja PPTX.



(b) Wersja TEX po konwersji.

Rysunek 13: Przykład dodania slajdu tytułowego.

5. Dodanie spisu treści
   Gdy opcja zostanie wybrana, każdy slajd otrzymuje nowy znacznik `section`, którego argumentem jest tytuł slajdu, a który określa nazwę paragrafu. Nowy slajd ze spisem treści dodawany jest jako drugi slajd prezentacji, a jego przykład przedstawiono na rysunku 14.

Rysunek 14: Przykład wygenerowanego spisu treści.

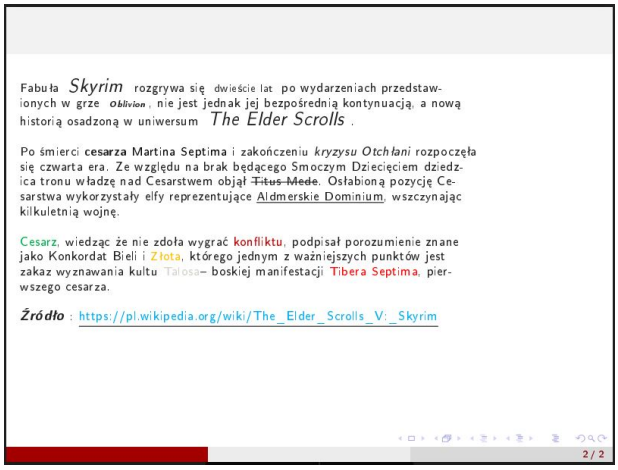
6. Zachowanie rozmiaru czcionek
   Określa, czy rozmiar czcionek ustawiony w oryginalnej prezentacji powinien zostać odwzorowany podczas konwersji. Przykład na rysunku 15 (pierwszy paragraf).

7. Zachowanie stylów czcionek
   Definiuje, czy powinny zostać odtworzone formatowania: pogrubienie, kursywa, przekreślenie, podkreślenie. Przykład na rysunku 15 (drugi paragraf).

8. Zachowanie kolorów czcionek

   Określa, czy kolorowanie czcionek będzie dostępne w nowo utworzonej prezentacji. Podczas procesu konwersji tworzona jest specjalna mapa kolorów, a same kolory definiowane są na początku kodu TEX, by można je było w łatwy sposób zmieniać. Przykład na rysunku 15 (trzeci paragraf).

(a) Wersja PPTX.



(b) Wersja TEX po konwersji.

Rysunek 15: Przykład różnego formatowania tekstu.

9. Wykrywanie i formatowanie kodu źródłowego

   Wybór tej opcji spowoduje uruchomienie silnika konwersji odpowiedzialnego za rozpoznawanie kodu źródłowego.

4.3.1 Tekst

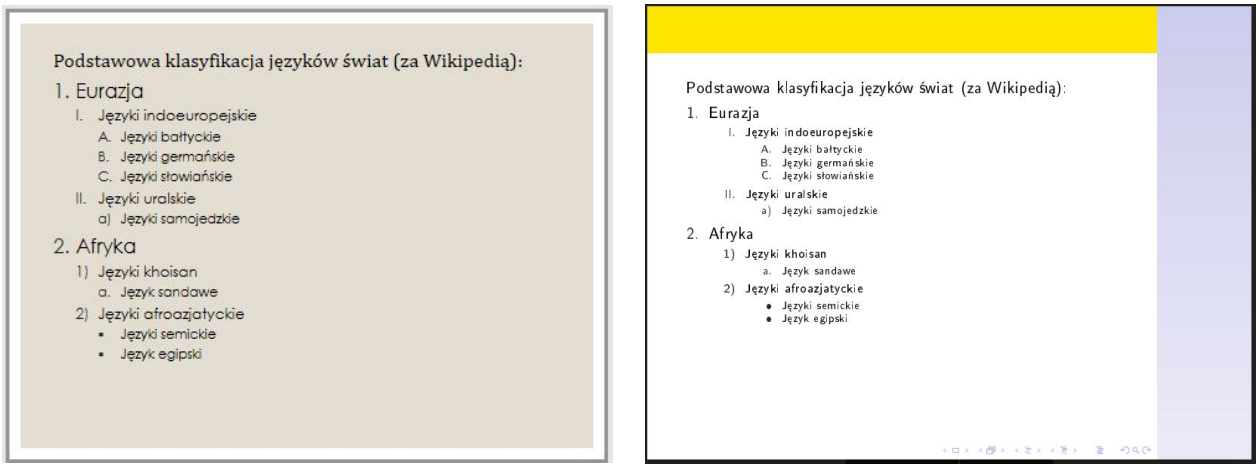
Za konwersję tekstu odpowiada klasa `TextConverter`. Bierze ona pod uwagę kształty typu `'pole tekstowe'` umieszczone na slajdzie. Dodatkowo, mechanizm konwersji tekstu używany jest np. w tabelach.

Każde pole tekstowe może składać się z wielu paragrafów tekstu, a te z wielu pojedynczych części tekstu (Text Run - opisanych w rozdziale 3.2.5). Wykrywane są wszystkie takie elementy, a następnie są one łączone, gdy nie różnią się stylami czy rozmiarami czcionek. Neutralizowane są znaki specjalne (np. poprzez zmianę znaku '$' na '\$') oraz odpowiednio zmieniane znaki nowej linii oraz tabulacji. Na podstawie wybranych opcji dodawane są style.

4.3.2 Listy i wypunktowania

Wspierane są oba środowiska `enumerate` oraz `itemize`. Dla drugiego z nich wszystkie znaki w PPTX będą przekonwertowane na '•', gdyż inne nie są łatwo dostępne w TEX, natomiast dla pierwszego, wspierana jest zarówno konwersja cyfr rzymskich, arabskich, jak i liter. Zacho-

wywane są rozmiary wcięć, style tekstu oraz kolor punktów. Wspierane są także wielokrotne zagnieżdżenia. Przykład przedstawiono na rysunku 16.

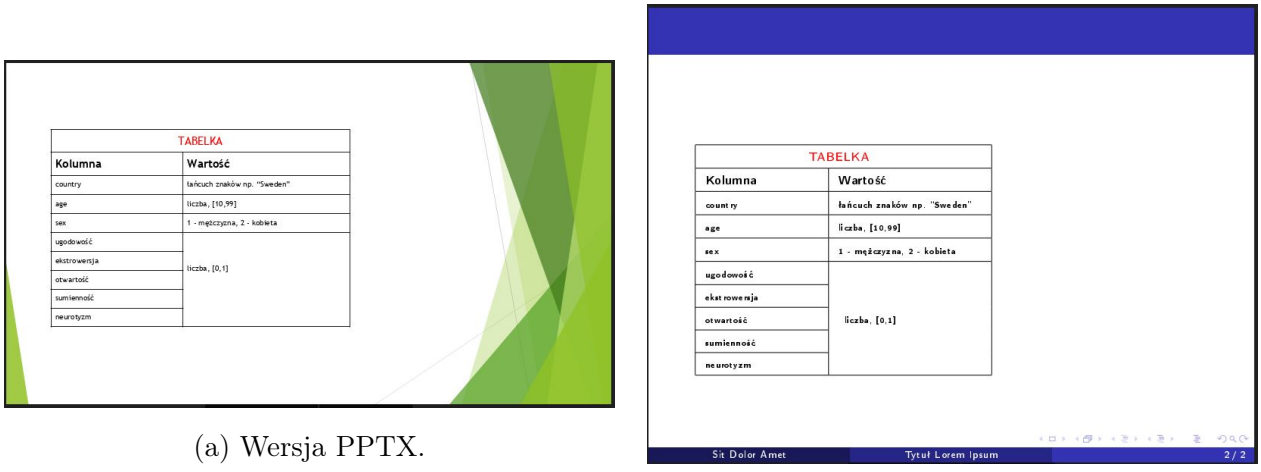


(a) Wersja PPTX. (b) Wersja TEX po konwersji.

Rysunek 16: Przykład konwersji list i wypunktowań.

4.3.3 Tabele

Za konwersję tabel odpowiada klasa `TableConverter`. Wpiera ona wszystkie możliwe konfiguracje tabel: dowolna liczba kolumn, wierszy oraz możliwość scalania komórek zarówno w pionie, jak i poziomie. W tym celu w TEXu używany jest pakiet `multirow`. Konwertowany jest rozmiar tabel oraz cała zawartość tekstowa. Przykład konwersji przedstawiono na rysunku 17.



(a) Wersja PPTX. (b) Wersja TEX po konwersji.

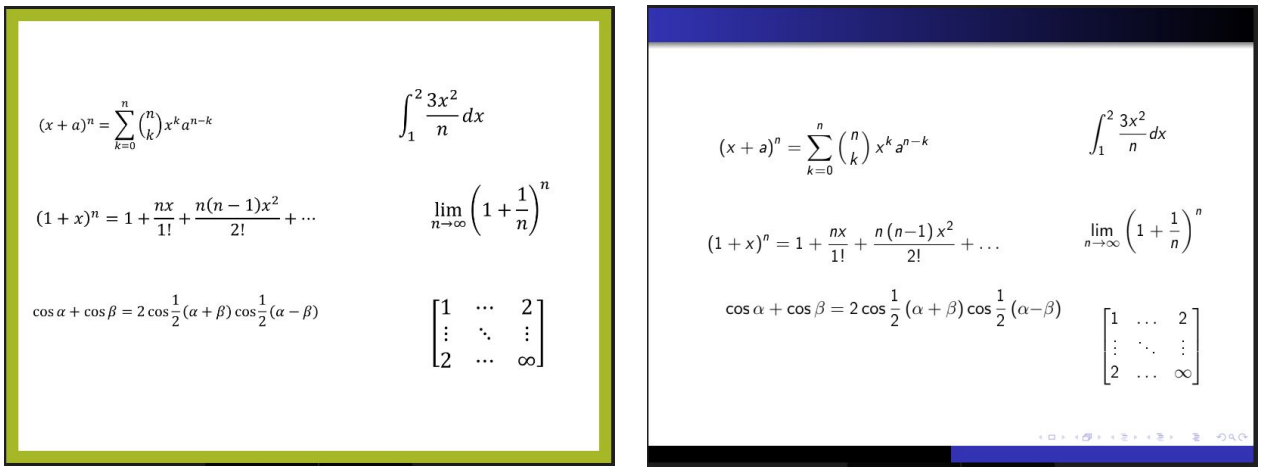
Rysunek 17: Przykład konwersji tabeli.

4.3.4 Formuły matematyczne

`MathConverter` jest najbardziej rozbudowanym ze wszystkich zaimplementowanych konwerterów. Wynika to z faktu, że używana biblioteka (Apache POI) nie ma dla nich żadnego

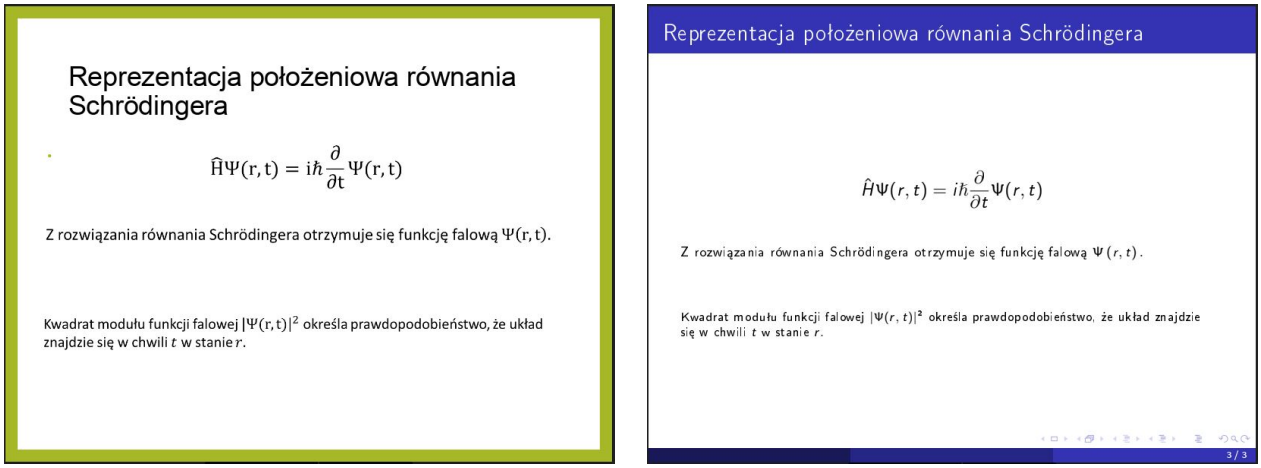
wsparcia, nie dostarcza więc poziomu abstrakcji nad zwykły kod źródłowy w języku XML. W związku z tym napisano własne parsery odpowiedzialne za analizę formuł matematycznych.

Wspierane są wszystkie notacje matematyczne, jak i prawie wszystkie symbole (w tym celu stworzono specjalną mapę translacyjną, która zawiera ponad 900 symboli). Konwertowane są zarówno równania w tekście, jak i poza nim (przykład poniżej na rysunku 18 oraz 19).



(a) Wersja PPTX. (b) Wersja TEX po konwersji.

Rysunek 18: Przykład konwersji formuł matematycznych jako osobnych pól.



(a) Wersja PPTX. (b) Wersja TEX po konwersji.

Rysunek 19: Przykład konwersji formuł matematycznych jako części tekstu.

4.3.5 Obrazki

`PictureConverter` wspiera formaty: BMP, GIF, JPEG, PNG. Obrazy odczytywane są z prezentacji PPTX, a następnie zapisywane w katalogu danej konwersji, skąd później są pobierane podczas generacji pliku PDF. Istnieje również możliwość ich pobrania (o czym w kolejnym rozdziale). Jeśli format obrazu nie jest wspierany, dodawane jest ostrzeżenie do wyników konwersji. Obrazy są odpowiednio skalowane oraz, jeśli zajdzie taka potrzeba, obracane. Przykładowy

wynik konwersji przedstawiono na rysunku 20.

(a) Wersja PPTX.

(b) Wersja TEX po konwersji.

Rysunek 20: Przykład konwersji obrazków.

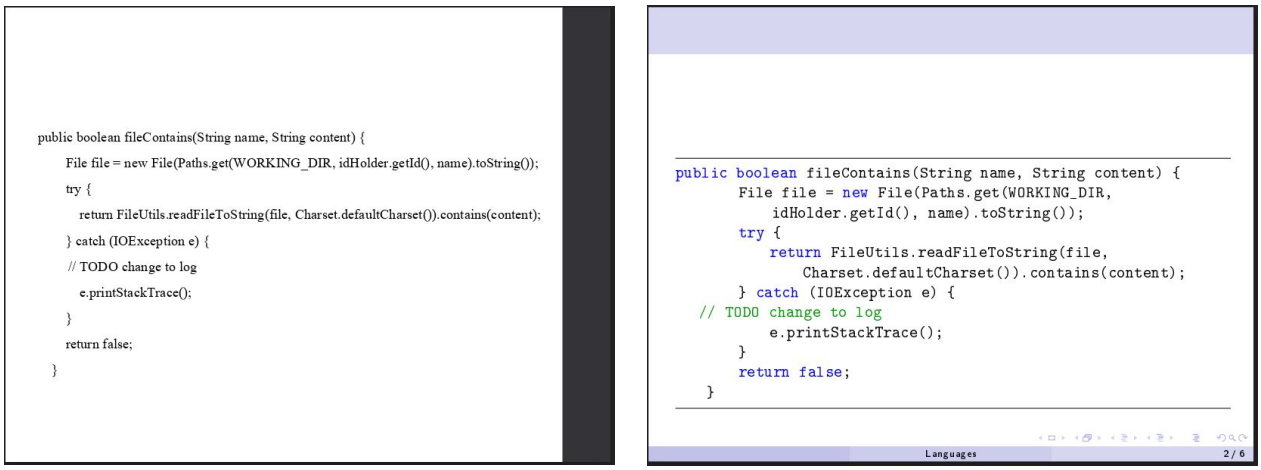
4.3.6 Kod źródłowy

Za obsługę rozpoznawania kodu źródłowego odpowiada klasa `CodeService`. Używa ona klasy `CodeDetector`, która zajmuje się wysyłaniem i obsługą żądań do serwera Flask. Wynikiem takiego żądania jest najbardziej prawdopodobny język programowania wraz z uzyskanym prawdopodobieństwem.

Zapytanie jest wysyłane tylko, jeśli tekst zawiera powyżej 100 znaków (by jakość klasyfikacji przez model była większa). Jeśli wartość prawdopodobieństwa jest powyżej 90%, fragment tekstu jest formatowany jako kod źródłowy. W przeciwnym wypadku to użytkownik wybiera, czy dany fragment ma być zmieniony.

Jako wynik konwersji zwracane są przewidywane języki programowania. Po ich uzyskaniu użytkownik może je dowolnie modyfikować. Sam proces automatycznej detekcji jest uruchamiany tylko raz, natomiast później to na podstawie wyborów użytkownika tekst jest formatowany. Dzięki temu tylko pierwsza konwersja może być stosunkowo wolna.

Wspierane języki to: C, C++, C#, CSS, Erlang, Go, HTML, Java, Javascript, Markdown, Objective-C, PHP, Perl, Python, Ruby, Rust, SQL, Scala, bash, Swift. Przykład detekcji kodu w języku Java przedstawiono na rysunku 21.



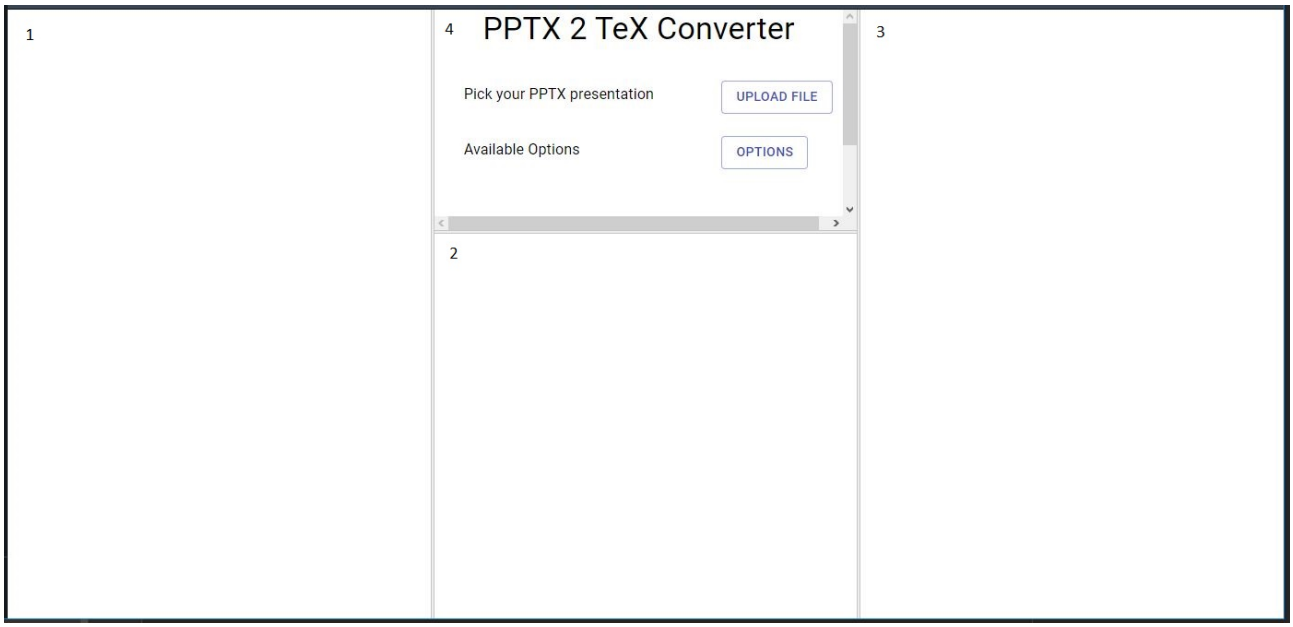
(a) Wersja PPTX.

(b) Wersja TEX po konwersji.

Rysunek 21: Przykład konwersji kodu źródłowego.

5 Opis użytkowy

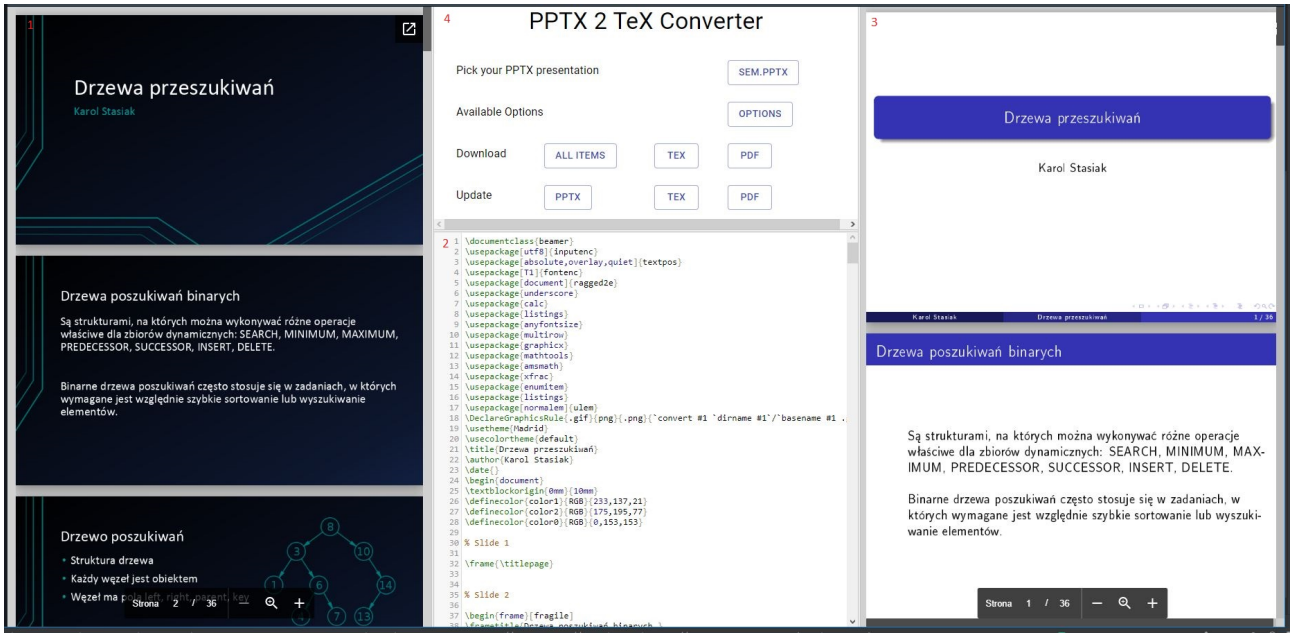
Aplikacja klienta została napisana z użyciem bibliotek React oraz Redux. Główna myśl towarzyszącą jej powstawaniu to prostota używania oraz przejrzysty interfejs, a przy tym duża funkcjonalność. Aplikacja została stworzona w taki sposób, by cała treść wyświetlana była na jednej stronie internetowej. Interfejs został przedstawiony na rysunku 22.



Rysunek 22: Główny interfejs aplikacji po wejściu na stronę.

Strona została podzielona na 4 części (podpisane na rysunku). Rozmiar każdego z nich jest konfigurowalny (poprzez przesunięcie myszką linii na granicy obszarów). W części centralnej oznaczonej numerem 4, znajduje się nazwa aplikacji oraz przyciski, odpowiedzialne za treści

wyświetlane na stronie. Po kliknięciu przycisku `UPLOAD FILE` pojawia się pole do wyboru prezentacji w formacie PPTX z lokalnego systemu plików. Po jej wyborze klient wysyła na serwer żądania konwersji. Użytkownik po krótkiej chwili widzi to, co znajduje się na rysunku 23. Możliwy jest także wybór opcji przed procesem konwersji.



Rysunek 23: Interfejs aplikacji po załadowaniu prezentacji.

W części 1 widoczna jest oryginalna prezentacja, którą wybrał użytkownik. Na dole znajduje się mały interfejs, na którym widoczna jest aktualnie wyświetlana strona prezentacji oraz przyciski do powiększenia slajdu (rysunek 24). Na górze po prawej stronie znajduje się przycisk, który umożliwia otwarcie tej prezentacji w nowym oknie przeglądarki.

Rysunek 24: Interfejs wyświetlanej prezentacji.

W części 3 wyświetlana jest prezentacja, która została utworzona z wyniku konwersji kodu TEX do formatu PDF. Interfejs jest identyczny jak dla części 1. Dzięki wyświetlaniu obu prezentacji obok siebie, użytkownik widzi na jednym widoku, jak jego prezentacja wygląda po konwersji. W obu przypadkach do wyświetlania treści został użyty element `<iframe>` [22]. Zawiera on osadzony widok innej strony. Do wyświetlania formatu PPTX oraz PDF, użyto narzędzia dostarczanego przez firmę `Google` o nazwie `Google Docs` [23].

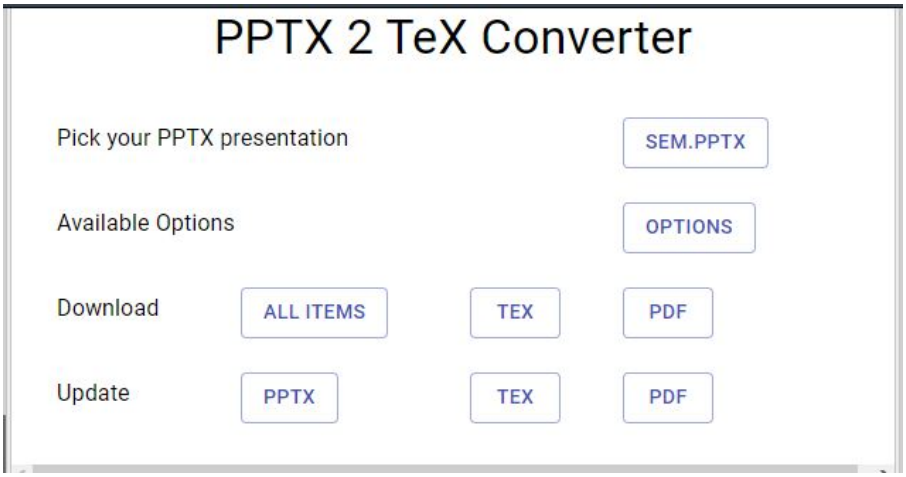
W części 2 wyświetlany jest kod TEX, czyli główny wynik konwersji. Przykład kodu przedstawiono na rysunku 25.

```
\includegraphics[width=41.33mm, height=45.98mm]{image3.gif}
\end{textblock*}
\end{frame}

% Slide 21

\begin{frame}[fragile]
\frametitle{Algorytm DSW}
\begin{textblock*}{109.3mm}(12.9mm,20.0mm)

\setlength{\leftmargini}{3.22mm}

\fontsize{11.7}{14.0}\selectfont
\begin{itemize}[label=\textcolor{color0}{•}]
\fontsize{11.7}{14.0}\selectfont \item
algorytm równoważący binarne drzewa poszukiwań\\
\fontsize{11.7}{14.0}\selectfont \item
powoduje że wysokość drzewa jest rzędu O$(\log n)$
\fontsize{11.7}{14.0}\selectfont
, gdzie n to liczba węzłów\\
\fontsize{11.7}{14.0}\selectfont \item
działa w czasie proporcjonalnym do liczby węzłów O(n)\end{itemize}
\end{textblock*}
\end{frame}

% Slide 22

\begin{frame}[fragile]
\frametitle{Algorytm DSW}
\begin{textblock*}{109.3mm}(12.9mm,20.0mm)

\fontsize{11.7}{14.0}\selectfont
\textcolor{color0}{I faza algorytmu:}\\
\setlength{\leftmargini}{3.22mm}

\fontsize{11.7}{14.0}\selectfont
\begin{itemize}[label=\textcolor{color0}{•}]
\fontsize{11.7}{14.0}\selectfont \item
zamieniamy całe drzewo w listę - powstaje wówczas drzewo, w którym węzły mają t
```

Rysunek 25: Kod TEX - wynik konwersji.

Do kodu przed każdym slajdem dodawany jest komentarz z numerem danego slajdu (w celu łatwiejszej identyfikacji treści). Sama składnia kodu jest kolorowana, wyświetlane są numery linii kodu, a kod jest edytowalny. W celu zapewnienia tych funkcjonalności, użyty został edytor `CodeMirror` [24].

Część opisana numerem 4 odpowiada za sterowanie treściami pokazywanymi na stronie. Jej wygląd po wyborze prezentacji przedstawiono na rysunku 26.



Rysunek 26: Nagłówek strony wraz z przyciskami.

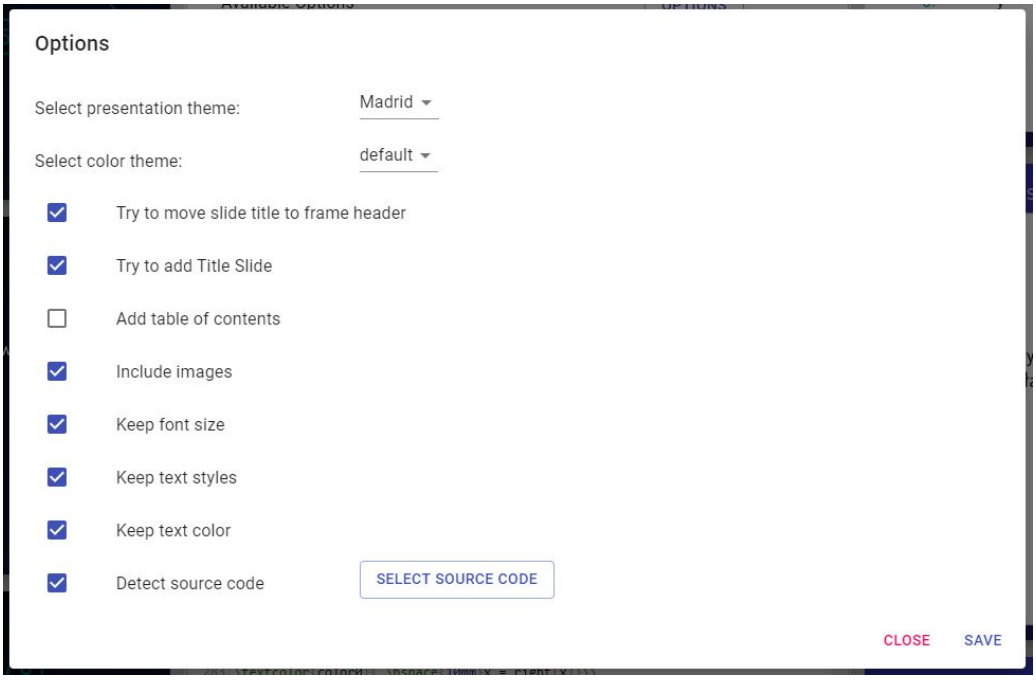
Po wyborze prezentacji, przycisk służący do jej wyboru zmienia nazwę z `UPLOAD FILE` na nazwę wybranego pliku. Za napisem `Download` znajdują się przyciski, odpowiedzialne za pobieranie wyników konwersji:

- `ALL ITEMS` - pobiera wszystkie wyniki konwersji spakowane do jednego archiwum zip, są to: prezentacja w formacie PPTX, prezentacja w formacie PDF, kod TEX, wszystkie obrazki, pliki powstałe podczas konwersji (np. logi),
- `TEX` - pobiera zawartość okna oznaczonego nr 2, jako plik TEX,
- `PDF` - pobiera prezentację w formacie PDF, widoczną w oknie nr 3.

Przyciski za napisem `Update` odpowiadają za odświeżanie zawartości okien. Przycisk:

- `PPTX` - ładuje ponownie prezentację w formacie PPTX widoczną w oknie nr 1,
- `TEX` - uruchamia ponownie proces konwersji prezentacji do kodu TEX i odświeża zawartość okna nr 2,
- `PDF` - tworzy plik PDF z zawartości okna kodu TEX oznaczonego nr 2 i odświeża zawartość okna nr 3.

Po kliknięciu przycisku `OPTIONS` pojawia się nowe okno w wyborem opcji konwersji przedstawione na rysunku 27. Opis dostępnych funkcji znajduje się w rozdziale 4.3.



Rysunek 27: Okno z wyborem opcji konwersji.

Po wybraniu przycisku `CLOSE` wszystkie zmiany w opcjach są odrzucane, natomiast po kliknięciu `SAVE` zmiany są zapisywane oraz ponownie generowany jest kod TEX. Odświeżane jest okno numer 2 i 3 zaprezentowane na rysunku 23.

Przycisk SELECT SOURCE CODE służy do wyboru opcji formatowania tekstu jako kodu źródłowego oraz ewentualnego języka programowania dla danego fragmentu. By móc dokonać takiego wyboru, najpierw należy wygenerować kod TEX z zaznaczoną opcją wykrywania kodu źródłowego. Podczas takiej konwersji używany jest serwer do wykrywania kodu, a jego wynik umieszczany jest w oknie przedstawionym na rysunku 28.

Rysunek 28: Okno z wyborem języka programowania dla kodu źródłowego.

W oknie tym dostępne są pola tekstowe ze slajdu, którego wybór znajduje się nad przyciskiem CLOSE. Jeżeli serwer wykrywający kod uznał, że dany fragment jest kodem źródłowym, automatycznie wybierany jest język z dostępnych opcji (wartością domyślną języka jest wartość TEXT, oznaczająca zwykły tekst). Jeśli wynik rozpoznania języka przekroczył 90%, zaznaczana jest opcja formatowania danego fragmentu tekstu jako kodu źródłowego w wybranym języku programowania. Użytkownik może dowolnie modyfikować wszystkie opcje wybrane przez automatyczną detekcję. Po wybraniu języka programowania, fragment tekstu (widoczny po lewej stronie okna) jest kolorowany, zgodnie ze składnią danego języka. Używany jest do tego edytor CodeMirror [24]. Kliknięcie przycisku SAVE powoduje zapis wszystkich zmian, natomiast przycisku CLOSE powoduje ich odrzucenie.

Jeśli podczas procesu konwersji wystąpią błędy, serwer generuje odpowiedź wraz z wiadomościami je opisującymi. Wiadomości takie są wyświetlane na ekranie użytkownika w formie Toast Messages na górze po prawej stronie. Zdefiniowano dwa typy wiadomości: ostrzeżenie oraz błąd. Ostrzeżeniem może być niewspierany format obrazka, natomiast błędem będzie brak możliwości konwersji. Przykładowe wiadomości znajdują się na rysunku 29.

(a) Ostrzeżenie o nieobsługiwanym formacie obrazka.

(b) Błąd podczas generacji pliku PDF.

Rysunek 29: Przykładowe wiadomości pojawiające się na ekranie użytkownika.

Ważnym aspektem interfejsu jest możliwość dowolnej edycji kodu TEX i generowania nowej prezentacji w formacie PDF. Umożliwia to ręczne wprowadzanie poprawek w wygenerowanym przez konwerter kodzie, bez potrzeby korzystania z zewnętrznego programu/edytora.

6 Wdrożenie aplikacji na serwer

W celu weryfikacji działania aplikacji zdecydowano się na umieszczenie jej w chmurze obliczeniowej. Wybrano usługi oferowane przez Amazon Web Services (AWS) [25], gdyż dostawca ten udostępnia swoje zasoby w darmowym pakiecie o nazwie Amazon Free Tier [26]. Cała aplikacja została umieszczona na jednej maszynie wirtualnej z użyciem Amazon EC2[13]. Dodatkowo, w celu łatwiejszej aktualizacji aplikacji na serwerze, skonfigurowano własny serwer CI/CD[14] o nazwie TeamCity [27], który automatycznie pobierał najnowsze zmiany z systemu kontroli wersji, tworzył nową paczkę instalacyjną, umieszczał ją na maszynie wirtualnej i uruchamiał aplikację.

6.1 Docker

W celu łatwiejszej dystrybucji aplikacji, zdecydowano się na jej konteneryzację z użyciem narzędzia Docker[15] [28]. Stworzono trzy obrazy: serwer Spring Boot (dodatkowo zawiera dystrybucję TEXa - TeXLive [29]), serwer Flask oraz aplikacja klienta React. Razem z kodem

[13] Amazon Elastic Compute Cloud (Amazon EC2) - usługa udostępniająca maszyny wirtualne na żądanie

[14] CI/CD (ang. Continuous Integration, Continuous Delivery) - zbiór zasad i kolekcja dobrych praktyk dotyczących pracy nad projektami informatycznymi, polegająca na ciągłym dostarczaniu w pełni funkcjonalnych, nowych wersji oprogramowania

[15] Docker - narzędzie, które pozwala umieścić program oraz jego zależności (biblioteki, pliki konfiguracyjne, lokalne bazy danych itp.) w lekkim, przenośnym, wirtualnym kontenerze, który można uruchomić na prawie każdym systemie operacyjnym

źródłowym do pracy dołączony jest plik `docker-compose.yaml` wraz ze wszystkimi definicjami. Dzięki temu cały proces uruchomienia aplikacji składa się z jednego kroku - uruchomienia komendy `docker-compose up`.

7 Podsumowanie

7.1 Wnioski

W tekście pracy pokazano, jak zbudowany jest format PPTX. Przedstawiono przykładowe struktury kodu XML oraz objaśniono jak tworzone są prezentacje. Wyjaśniono, jak dodawane są do slajdu jego pojedyncze elementy, takie jak: tekst czy formuły matematyczne. Pokazano również, w jaki sposób tworzone są dokumenty z użyciem kodu TEX. W szczególności, jakie funkcjonalności oferuje pakiet do tworzenia prezentacji o nazwie BEAMER.

W ramach pracy stworzono aplikację internetową umożliwiającą konwersję prezentacji w formacie PPTX na kod źródłowy TEX oraz jego graficzną reprezentację w formacie PDF. Konwersji poddawane są różne elementy obecne na slajdzie, wliczając w to: tekst, listy, obrazki, formuły matematyczne i tabele. Rozpoznawany jest także kod źródłowy umieszczony w tekście oraz automatycznie, dzięki sztucznej inteligencji, formatowany na odpowiedni język.

Aplikacja została napisana w architekturze klient-serwer. Główny serwer aplikacyjny zbudowano z użyciem języka programowania Java, natomiast drugi serwer, odpowiedzialny za rozpoznawanie kodu źródłowego, napisano z użyciem języka Python. Strona klienta, która powstała z użyciem biblioteki React, odznacza się dużą przejrzystością prezentowanych treści oraz jest łatwa w użyciu. Konwersja wielu elementów jest opcjonalna, dzięki czemu to użytkownik decyduje o wyglądzie prezentacji po konwersji.

Udało się w całości zrealizować założony cel pracy. Stworzona aplikacja jest w pełni funkcjonalna i z powodzeniem może być używana do konwersji.

7.2 Perspektywy rozwoju

Istnieją pewne elementy dostępne w formacie PPTX, które nie są poddawane konwersji. Przykładem mogą być różne kształty: strzałki, schematy blokowe czy wykresy. Kolejnym krokiem rozwoju pracy, może być dopisanie dla nich wsparcia.

8 Załączniki

Załącznikiem do pracy magisterskiej jest płyta CD, na której znajduje się kod źródłowy aplikacji, dokumentacja kodu oraz plik *praca_mgr.pdf*, będący elektroniczną wersją tej pracy.

Bibliografia

[1] Katharine Molloy, Timothy D. Griffiths, Maria Chait and Nilli Lavie
Inattentional Deafness: Visual Load Leads to Time-Specific Suppression of Auditory Evoked Responses
Journal of Neuroscience 9 December 2015, 35 (49) 16046-16054;
https://doi.org/10.1523/JNEUROSCI.2931-15.2015

[2] Fatemeh Bambaeeroo and Nasrin Shokrpour
The impact of the teachers 'non-verbal' communication on success in teaching
J Adv Med Educ Prof. 2017 Apr; 5(2): 51–59;
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5346168/

[3] Java - historia, (dostęp 20.01.2020)
https://www.freejavaguide.com/history.html

[4] TIOBE Index dla sierpnia 2020, (dostęp 20.01.2020)
https://www.tiobe.com/tiobe-index/

[5] Spring Framework, (dostęp 23.01.2020)
https://spring.io/projects/spring-framework

[6] Spring Boot, (dostęp 23.01.2020)
https://spring.io/projects/spring-boot

[7] Spring Data, (dostęp 23.01.2020)
https://spring.io/projects/spring-data

[8] Spring MVC, (dostęp 23.01.2020)
https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#mvc

[9] Prezentacja OOXML, (dostęp 14.02.2020)
http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf

[10] Popularność frameworków, (dostęp 14.02.2020)
https://existek.com/blog/top-front-end-frameworks-2020/

[11] React - historia, (dostęp 14.02.2020)
https://en.wikipedia.org/wiki/React_(web_framework)

[12] Redux, (dostęp 24.02.2020)
https://redux.js.org/introduction/getting-started

[13] Standard ECMA-376, (dostęp 14.02.2020)
http://www.ecma-international.org/publications/standards/Ecma-376.htm

[14] Historia TEXa, (dostęp 18.02.2020)
https://www.tug.org/whatis.html

[15] Historia LATEXa, (dostęp 18.02.2020)
https://www.britannica.com/technology/LaTeX-computer-programming-language

[16] BEAMER, (dostęp 18.02.2020)
https://en.wikipedia.org/wiki/Beamer_(LaTeX)

[17] Prezentacja funkcji BEAMERA, (dostęp 18.02.2020)
https://sunsite.icm.edu.pl/pub/CTAN/macros/latex/contrib/beamer/doc/beameruserguide.pdf

[18] CTAN, (dostęp 18.02.2020)
https://www.ctan.org

[19] Dokumentacja Flask, (dostęp 25.02.2020)
https://flask.palletsprojects.com/en/1.1.x/

[20] Guesslang, (dostęp 25.02.2020)
https://github.com/yoeo/guesslang

[21] Dokumentacja Guesslang, (dostęp 25.02.2020)
https://guesslang.readthedocs.io/en/latest/g

[22] Dokumentacja iframe, (dostęp 04.03.2020)
https://developer.mozilla.org/pl/docs/Web/HTML/Element/iframe

[23] Google Docs Viewer, (dostęp 04.03.2020)
https://gist.github.com/tzmartin/1cf85dc3d975f94cfddc04bc0dd399be

[24] CodeMirror, (dostęp 04.03.2020)
https://codemirror.net/

[25] Amazon Web Services, (dostęp 25.05.2020)
https://aws.amazon.com

[26] AWS - Free Tier, (dostęp 25.05.2020)
https://aws.amazon.com/free/

[27] TeamCity, (dostęp 25.05.2020)
https://www.jetbrains.com/teamcity/

[28] Docker, (dostęp 16.08.2020)
https://www.docker.com/

[29] TeXLive, (dostęp 17.07.2020)
https://www.tug.org/texlive/

Spis rysunków