Wydział Matematyki
i Nauk Informacyjnych

POLITECHNIKA WARSZAWSKA

PROJEKT TECHNICZNY

Z PRZEDMIOTU PROJEKT INDYWIDUALNY

na kierunku Inżynieria i Analiza Danych

Projekt aplikacji do analizy i wizualizacji danych dotyczących wrażeń synestetycznych

Łukasz Brzozowski
Numer albumu 291122

Data zakończenia projektu: 14.06.2018

Prowadzący: dr hab. inż. Jerzy Balicki, prof. PW

Warszawa 2018

Spis treści

Wprowadzenie

1. Dokumentacja architektoniczna
    1.1. Architektura rozwiązania
    1.2. Wykorzystane technologie
    1.3. Model wytwórczy
2. Dokumentacja techniczna
    2.1. Wymagania systemowe
    2.2. Biblioteki wraz z określeniem licencji
    2.3. Diagramy sekwencji
    2.4. Model danych
3. Testy jednostkowe

Podsumowanie

Wykaz literatury

PROJEKT TECHNICZNY

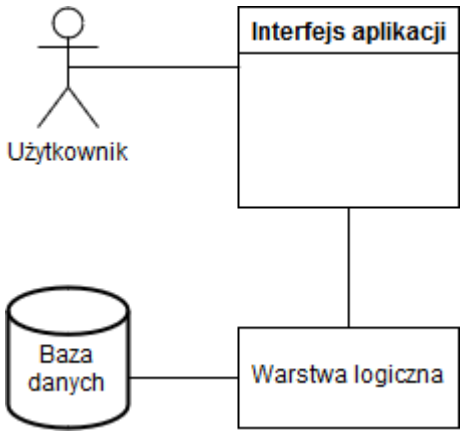
Wprowadzenie

W poniższym dokumencie znajdują się dokumentacja techniczna i architektoniczna aplikacji SynVisual służącej do generowania animacji na podstawie tekstu. Przedstawione zostały schematy działania aplikacji, omówiono wykorzystane biblioteki oraz opisano przeprowadzone testy.

1 Dokumentacja architektoniczna

1.1 Architektura rozwiązania

Na rysunku 1. przedstawiono architekturę systemu.



Rysunek 1: Schemat architektury rozwiązania

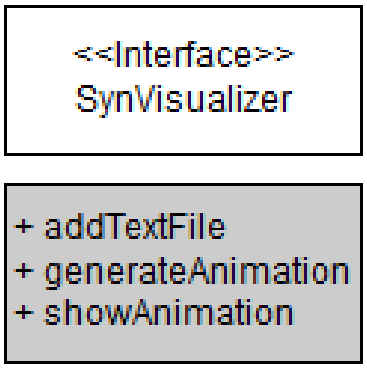
Aplikacja jest w pełni osadzona na komputerze użytkownika. Składają się na nią warstwa logiczna odpowiadająca za uruchamianie aplikacji oraz wyświetlanie interfejsu oraz baza danych przechowująca połączenia między kolorami a znakami.

Klient komunikuje się z aplikacją poprzez interfejs graficzny napisany w języku Java. Zapewnia to przenośność aplikacji oraz możliwość korzystania z niej na różnych systemach operacyjnych.

Osadzenie aplikacji na komputerze użytkownika umożliwia korzystanie z niej bez dostępu do Internetu, co gwarantuje dodatkową przenośność. Ponadto, rozmiar aplikacji i jej zaawansowanie są na tyle małe, że jej uruchomienie nawet na starszych komputerach nie powinno sprawiać problemu.

Interfejs aplikacji umożliwia wybranie pliku z tekstem, którego wizualizacja ma zostać przeprowadzona oraz umożliwia sterowanie odtwarzaniem animacji. Dodatkowo, dzięki osadzeniu bazy danych

na komputerze użytkownika, łatwe staje się modyfikowanie pliku zawierającego informacje o przyporządkowaniu kolorów do znaków, co umożliwia tworzenie własnych schematów kolorystycznych do wykorzystania przez aplikację.



Rysunek 2: Metody interfejsu aplikacji

1.2 Wykorzystane technologie

Zarówno interfejs użytkownika, jak i skrypty składające się na wartswę logiczną aplikacji wykorzystują język Java. Jego zaletami jest przenośność kodu oraz szeroki wybór bibliotek, m.in. do generowania animacji.

Za pomocą bibliotek java.swing oraz java.awt aplikacja złoży animacje na podstawie ramki powiązań znajdującej się w bazie danych. Każda litera tekstu zostanie automatycznie przełożona na kilka klatek animacji.

Te same biblioteki posłużą do stworzenia interfejsu całej aplikacji. Zapewniają one szeroki zakres możliwych rozwiązań graficznych oraz gwarantują przenośność aplikacji pomiędzy różnymi systemami. Użytkownik również może mieć wpływ na wygląd interfejsu aplikacji, z której korzysta.

Ponadto podczas projektowania aplikacji zostaną użyte standardowe biblioteki języka Java: java.lang, java.util.Random, java.util.Math, java.io.Reader, które zapewnią odpowiednie narzędzia do działania aplikacji.

1.3 Model wytwórczy

Proces wytwórczy aplikacji oparty będzie o zasady modelu kaskadowego (waterfall). Możemy w nim wyróżnić kolejne fazy, takie jak:

1. Określenie wymagań
2. Projektowanie

3. Implementacja
4. Testowanie
5. Wdrożenie
6. Utrzymanie

Przejście do kolejnej fazy następuje dopiero po zakończeniu poprzedniej.

Wymagania odbiorcy zostały ściśle wyspecyfikowane przed przystąpieniem do analizy i implementacji. Efekt prac nad każda faza jest prezentowany odbiorcy i dostosowywany do jego ewentualnych uwag. Zgodnie ze specyfiką systemu informatycznego powstającego w ramach projektu indywidualnego, faza utrzymania może zostać pominięta ze względu na krótki czas życia systemu.

Podczas fazy implementacji wykorzystywana będzie również metodologia pracy *Test Driven Development*. Przed powstaniem danej części systemu opracowywany jest zestaw testów, które stanowią pewien rodzaj uzupełnienia specyfikacji, według której jest następnie opracowywany kod aplikacji. Prowadzi to do powstawania systemów zawierających mniej błędów (o wyższej jakości kodu) i zgodnych z wymaganiami odbiorców.

2 Dokumentacja techniczna

2.1 Wymagania systemowe

Aby uruchomić aplikację, wymagany jest system operacyjny Linux, Windows lub MacOS wraz z oprogramowaniem pozwalającym na uruchamianie aplikacji Java.

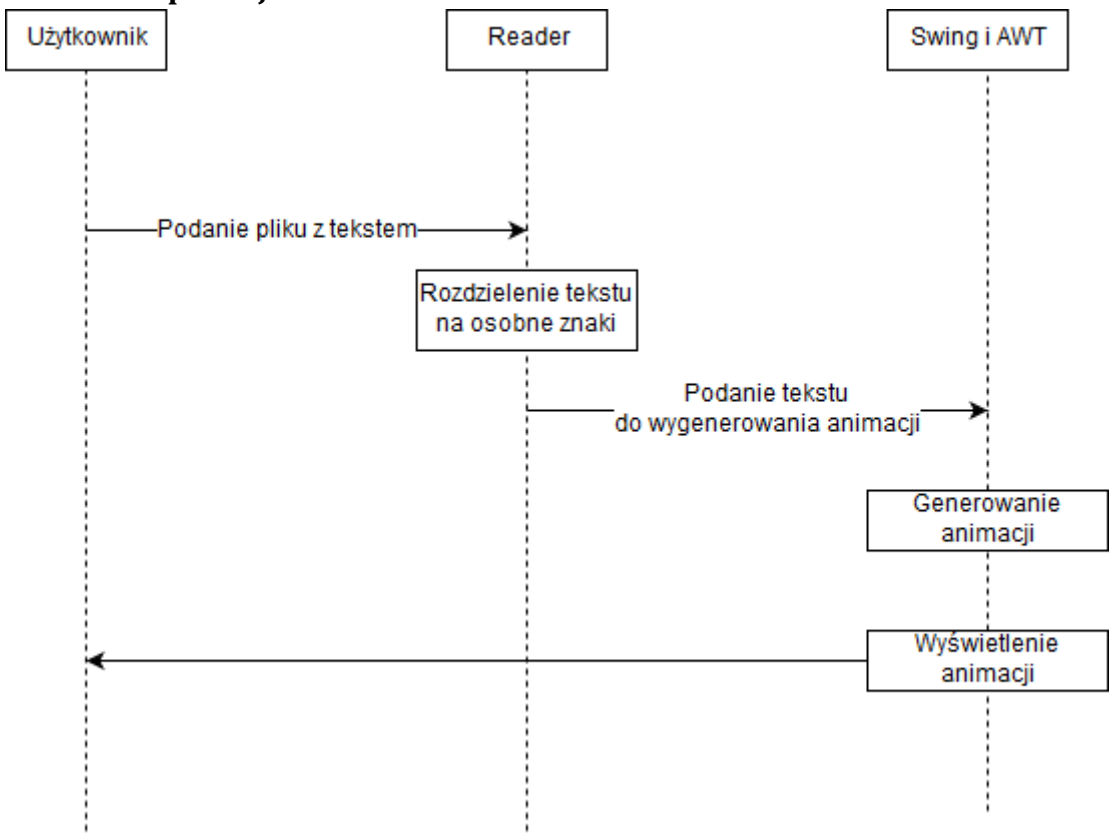
2.2 Biblioteki wraz z określeniem licencji

Nr.	Biblioteka	Opis	Licencja
1	Swing	Biblioteka do generowania animacji	Open Source
2	AWT	Biblioteka do generowania animacji	Open Source
3	Java.Reader	Biblioteka do odczytywania informacji z pliku tekstowego	Open Source

Tablica 1: Lista wykorzystanych bibliotek

2.3 Diagramy sekwencji

Działanie aplikacji



Rysunek 3: Schemat sekwencji działania aplikacji

Funkcjonalność	System(y)	Opis (wejście i wyjście)
Działanie aplikacji	Komputer użytkownika	Wejście: Uruchomienie przez użytkownia Wyjście: Wyświetlenie animacji na podstawie tekstu

Tablica 2: Schemat uruchomienia aplikacji

2.4 Model danych

Dokument z tekstem, na podstawie którego aplikacja ma dokonać wizualizacji, zostaje podany przez użytkownika poprzez wybranie odpowiedniego pliku w eksploratorze systemu. Plik musi być zapisany

rozszerzeniem .txt. Ponadto, aplikacja przechowuje plik .csv zawierający połączenia pomiędzy literami a odpowiadającymi im kolorami. Animacja zostaje wygenerowana w formacie .gif, jednak nie jest ona esportowalna poza aplikację.

3 Testy jednostkowe

Praca nad aplikacją przebiegała zgodnie z metodologią *Test Driven Development*, tj. przed powstaniem danej jej części (funkcji) powstawał zestaw testów, według którego opracowywany był kod. Praca nad daną funkcjonalnością rozpoczynała się od określenia przypadku testowego (odzwierciedlany jest on nazwa metody testującej). Metody testujące początkowo powstały bez implementacji. Następnie po utworzeniu pustych funkcjonalności powstała implementacja testów, których uruchomienie zgodnie z oczekiwaniami zakończyło się niepowodzeniem. Następnie metoda przyrostowa tworzone były pojedyncze minimalne funkcjonalności, które były poddawane testowaniu i refaktoryzacji. W ten sposób w duchu metodologii TDD powstały dobrze przetestowane funkcjonalności, zgodne z początkowymi oczekiwaniami.

Na funkcje aplikacji składają się:

-addTextFile() – funkcja dodająca plik tekstowy,

-readCSV() – funkcja pobierająca skojarzenia kolorów z bazy danych,

-generateAnimation() – funkcja generująca animację,

-showAnimation() – funkcja wyświetlająca animację.

Do wykonania testów poszczególnych funkcji został wykorzystane pakiet JTest, który jest integralną częścią pakietów Java.util i udostępnia skuteczne narzędzia do testowania programów napisanych w języku Java.

Moduły do testowania:

- Reader() – funkcja powinna zwracać błąd, gdy w pliku tekstowym są znaki spoza zbioru określonego w bazie danych, tj. spoza alfabetu angielskiego,
- generateAnimation() – funkcja nie powinna zwracać błędów w przypadku tekstów do 400 znaków,
- showAnimation() – wyświetlana animacja musi być płynna i utrzymywać ustalone tempo przez całą długość trwania animacji.

Podsumowanie

W niniejszym projekcie technicznym zawarto teoretyczny opis metod wykorzystanych podczas opracowywania aplikacji SynVisual służącej do generowania animacji na podstawie tekstu. Zawarto odpowiednią dokumentację techniczną wraz z diagramami sekwencji działania aplikacji oraz opisem testów jednostkowych. Kod został przygotowany w sposób, który pozwala na łatwe rozszerzanie istniejącej aplikacji, co umożliwia jej przyszły rozwój.

Wykaz literatury

1. https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html
2. https://www.javatpoint.com/java-swing
3. https://javastart.pl/baza-wiedzy/darmowy-kurs-java/grafika_awt_swing/wprowadzenie-awt-i-swing/
4. https://www.parasoft.com/products/jtest
5. https://docs.oracle.com/javase/7/docs/api/java/io/Reader.html
6. https://www.eclipse.org/windowbuilder/