<LanguageTutor>
Architecture Notebook

1. Cel (Purpose)

Celem dokumentu jest opis decyzji, ograniczeń, uzasadnień, znaczących elementów, a także innych nadrzędnych aspektów systemu, które kształtują jego projekt i implementację.

2. Cele Architektoniczne i Ograniczenia (Architectural goals and constraints)

Na podstawie informacji zawartych we wcześniejszych dokumentach: Wymagań Systemowych (LanguageTutor – Requirements.docx) i Wizji (Vision2.pdf) **wyróżnione** zostały następujące cele, które powinna spełniać architektura projektowanego Systemu:

*Funkcjonalne (*w kolejnym etapie zaimplementowane zostaną pierwsze dwa *):*

- **System umożliwia prowadzenie dialogów w formie głosowej**
- **System umożliwia wyświetlenie listy dostępnych dialogów użytkownikowi.**
- System umożliwia śledzenie postępu w nauce (szerzej: ma zdolność zapamiętywania określonych zdarzeń, które zaszły w Systemie)

Niefunkcjonalne:

- Płynne prowadzenie dialogu. Odpowiedź system poniżej 3 sekund przy maksymalnym obciążeniu systemu (tzn. 1000 użytkowników prowadzących dialog naraz) (Priorytet: **Wysoki**) – (*kat.* Performance)
- Konto użytkownika jest zabezpieczone przed nieautoryzowanym dostępem. (Priorytet: **Wysoki**) (*kat.* Security)
- System dostępny jest w różnych wersjach językowych. Są to język polski, angielski (opcjonalnie: włoski) (Priorytet: **Wysoki**) (*kat.* Usability)

Poniżej zamieszczono ograniczenia architektury, będące wynikiem decyzji podjętych w zakresie technologii, wybranej dla implementacji Systemu:

- System wymaga ciągłego dostępu do Internetu
- System jest dostępny tylko za pomocą przeglądarki Google Chrome z obsługą Java Script'u – dla realizacji funkcjonalności głosowej interakcji z Systemem

3. Decyzje I Uzasadnienia (Decisions and justifications)

Goal	How achieved (Tactics)
1. Płynne prowadzenie dialogu . Odpowiedź system poniżej 3 sekund przy maksymalnym obciążeniu systemu (tzn. 1000 użytkowników prowadzących dialog naraz)	A. Zrównoleglenie obciążenia pomiędzy osobne, niezależnie skalowalne jednostki logiczne. Dokładniej: dekompozycja elementów Systemu, odpowiedzialnych za umożliwienie prowadzenia dialogu, do osobnych komponentów rozmieszczanych niezależnie. B. Minimalizacja ilości danych przesyłanych pomiędzy aplikacją serwerową, a bazą danych Dokładniej: Użycie systemu Hibernate cache L2 w postaci biblioteki Ehcache dla przechowywania części danych pochodzących z bazy danych w pamięci serwera aplikacji C. Minimalizacja ilości danych wymienianych pomiędzy aplikacją kliencką, a serwerową. Dokładniej: Zastosowanie framework'a AngularJS umożlwiającego tworzenie aplikacji przeglądarkowych typu SPA ze wsparciem dla komunikacji REST z serwerem aplikacji.
2. Zabezpieczenie konta przed nieuprawionym dostępem	A. Autentykacja i autoryzacja. Dokładniej: Zastosowanie framework'a Spring Security zapewniającego elastyczne i przekrojowe mechanizmy wspierające ochronę przed nieautoryzowanym dostępem
3. System dostępny jest w różnych wersjach językowych. Są to język polski, angielski (opcjonalnie: włoski)	**A.** Zastosowanie mechanizmu zmiennych językowych Dokładniej: Korzystanie z utworzonych na każdą wersję językową osobnych pliku z tłumaczeniami tekstu wyświetlanego użytkownikom w widoku aplikacji. Ustalanie odpowiedniego pliku ma miejsce na podstawie aktualnych ustawień klienta.

4. Mechanizmy Architektoniczne (Architectural Mechanisms)

Poniżej opisano rozwiązania technologiczne, które umożliwią realizację taktyk opisanych w poprzednim punkcie.

1A. Zrównoleglenie obciążenia pomiędzy osobne, niezależnie skalowalne jednostki logiczne.

Tworzony system, na odpowiednio wysokim poziomie abstrakcji, traktowany powinien być jako realizacja architektury typu klient-serwer. Klient to aplikacja przeglądarkowa, która komunikuje się z aplikacją serwerową w celu realizacji zadań systemu (dokładniej: wymiany danych).

Analizując System bardziej szczegółowo, jesteśmy w stanie wyróżnić cztery główne komponenty Systemu:

a. Aplikacja kliencka działająca w przeglądarce użytkownika. Służy jako warstwa prezentacji, dostarczając graficznego interfejsu użytkownika, służy jako punkt wejściowy do Systemu dla użytkowników końcowych. Komunikuje się ona z aplikacją serwerową w celu wymiany danych. Zaimplementowana w technologii JavaScript i HTML.

b. Aplikacja serwerowa. Centralny punkt Systemu odpowiedzialny za realizację części logiki biznesowej systemu, autoryzację, dostarczanie danych dla aplikacji przeglądarkowej, komunikację bazą danych, a także z usługą prowadzenia dialogów. Bardzo łatwo skalowalna ze względu na swoją bezstanowość. Zaimplementowana w oparciu o język i technologie Java.

c. Usługa sieciowa (web service) prowadzenia dialogów. Służy do dostarczania funkcjonalności prowadzenia dialogu na wybrany temat z botem. Bardzo dobrze skalowalna ze względu na bardzo ograniczoną stanowość. Zaimplementowana w jeżyku C# w środowisku .NET.

d. Baza danych. Odpowiedzialna za składowanie i udostępnianie trwałych danych w Systemie.

Na każdy z tych komponentów można patrzeć, jako pewna usługa świadczona innym komponentom, bądź użytkownikowi końcowemu (w przypadku aplikacji klienckiej). Oznacza to, że na niższym poziomie abstrakcji, architekturę Systemu możemy określić jako zorientowaną na usługi – SOA.

Co ważne, każdy z tych komponentów może być rozmieszczany i skalowany niezależnie.
Obsługa funkcjonalności dialogu, wymaga udziału każdego z tych komponentów. Widać zatem, że obciążenie związane z realizacją tej funkcjonalności zostało rozdzielone pomiędzy kilka fizycznie niezależnych jednostek.

1B. Minimalizacja ilości danych przesyłanych pomiędzy aplikacją serwerową, a bazą danych

Zostanie to zrealizowane poprzez zastosowanie/uaktywnienie pamięci podręcznej L2 (drugiego poziomu) w konfiguracji framework'a ORM (w tym przypadku Hibernate) dla przechowywania stanu obiektów trwałych (składowanych w bazie danych) w pamięci serwera na okres dłuższy niż pojedyncza transakcja (realizowana zazwyczaj w ramach pojedynczego żądania klienta aplikacji) jak ma to miejsce w konfiguracji domyślnej i pamięci podręcznej L1. Rozwiązanie to pozwoli znacząco zredukować liczbę żądań wysyłanych z aplikacji serwerowej do bazy danych, gdyż aplikacja będzie miała dostęp z własnej pamięci do znacznej części danych. Rozwiązanie to szczególnie dobrze sprawdza się w sytuacji, gdy przeważającym typem operacji na danych trwałych jest odczyt. Jest to sytuacja z jaką mamy do czynienia w przypadku aplikacji LanguageTutor (np. odczyt definicji dialogów i ich kategorii, danych użytkowników, itp.). Jedną z implementacji Cache'a L2, która może zostać wykorzystana to biblioteka EhCache.

1C. Minimalizacja ilości danych wymienianych pomiędzy aplikacją kliencką, a serwerową.

Dzięki zastosowaniu komunikacji w stylu REST, pomiędzy aplikacją serwerową, a aplikacją kliencką działającą w przeglądarce i opartą o framework AngularJS, znacznie zredukowana zostaje ilość danych przesyłanych przez siec, co wpływa znacząco na całościowy czas odpowiedzi serwera na żądania klienta.
W tej technice komunikacji, aplikacja kliencka i serwerową w większości przypadków wymieniają pomiędzy sobą tylko dane (najczęściej w formacie JSON), natomiast aplikacja kliencka sama jest w stanie wyświetlać kolejne widoki na podstawie wcześniej pobranych z serwera plików java scriptu i HTML. Dodatkowy zysk zauważalny jest szczególnie w sytuacjach, kiedy widok zawiera pewien stały szablon (w którym umieszczona jest treść konkretnego widoku), który w tym przypadku nie wymaga przeładowania (ponownego przesłania przez siec z każdym żądaniem). Sytuacja taka ma miejsce w większości dzisiejszych aplikacji webowych, włączając projektowany System – LanguageTutor. Opisana charakterystyka systemów webowych nosi miano SPA (ang. Single Page Application)

2A. Autentykacja i autoryzacja

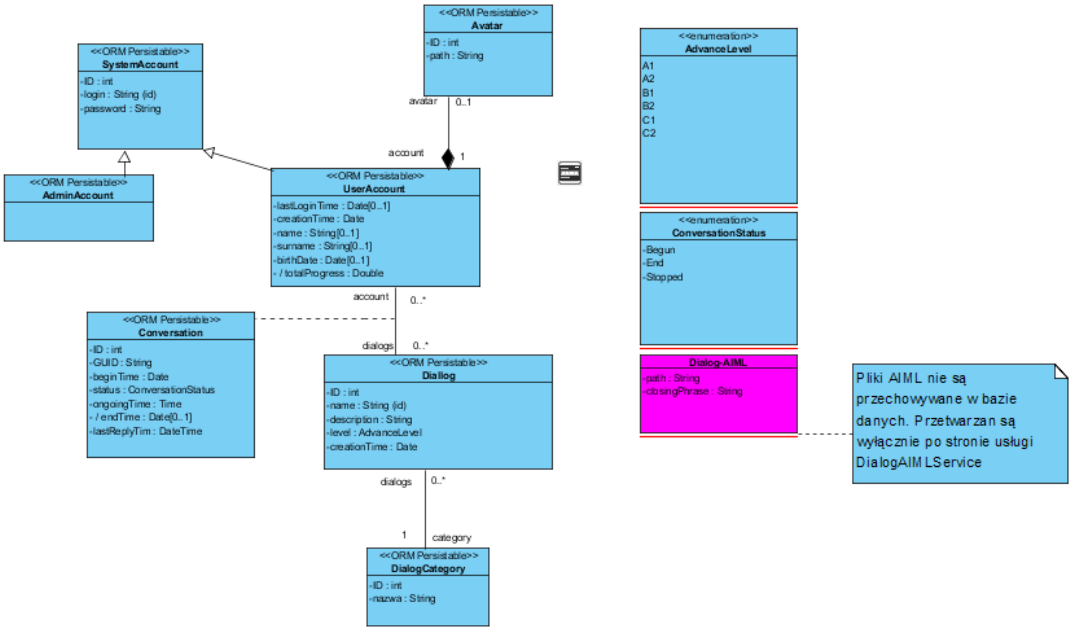
Fundamentem autoryzacji w systemie będzie framework Spring Security. Autoryzacja dokonywana będzie w aplikacji serwerowej w oparciu o standardowe mechanizmy protokołu http (sesja, ciasteczka, itp.). Zaimplementowana zostanie na dwóch poziomach: w warstwach kontrolerów (zabezpieczenie dostępu do konkretnych adresów URL) i usług (zabezpieczenie dostępu do określonych operacji w sytuacji poprawnej autoryzacji dostępu do określonego adresu URL).

3A. Zastosowanie mechanizmu zmiennych językowych po stronie widoku (JavaScript)

Dla każdego języka wspieranego przez LanguageTutor (przynajmniej angielski i polski), utworzone zostaną osobne pliki tzw. słowników zawierające tłumaczenie zmiennej widoku (klucza). Każda treść mogąca występować w różnych jeżykach, będzie w widokach HTML oznaczana za pomocą zmiennych językowych (abstrakcyjnych kluczy), podmienianych w czasie działania aplikacji na rzeczywiste, znaczące wartości pochodzące z pliku słownika dla w żądanego języka (w zależności od ustawionych preferencji językowych użytkownika.

5. Kluczowe Abstrakcje (Key abstractions)

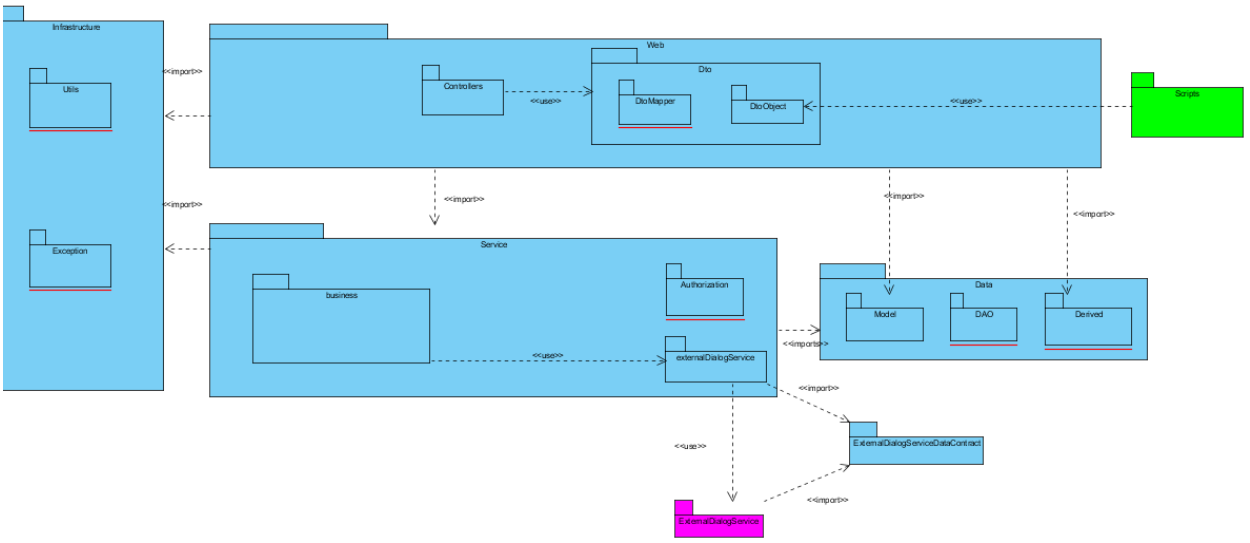
Poniżej zamieszczono diagram klas przedstawiający kluczowe abstrakcje w projektowanym systemie (perspektywa informacyjna). Implementowane elementy składowe przeznaczone do uruchomienia po stronie użytkownika (w przeglądarce internetowe oznaczono na **ZIELONO**), na serwerze aplikacji Tomcat na **NIEBIESKO**, usługi sieciowe DialogExternalService na **RÓŻOWO**



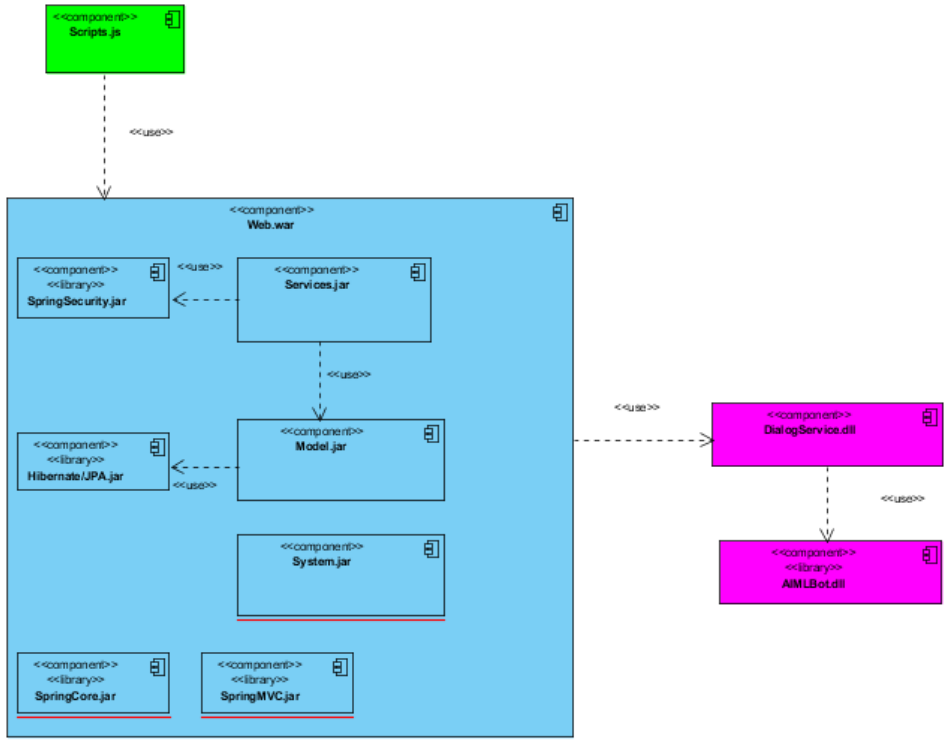
6. Widoki Architektoczniczne (Architectural views)

Poniżej zamieszczono widoki architektoniczne.

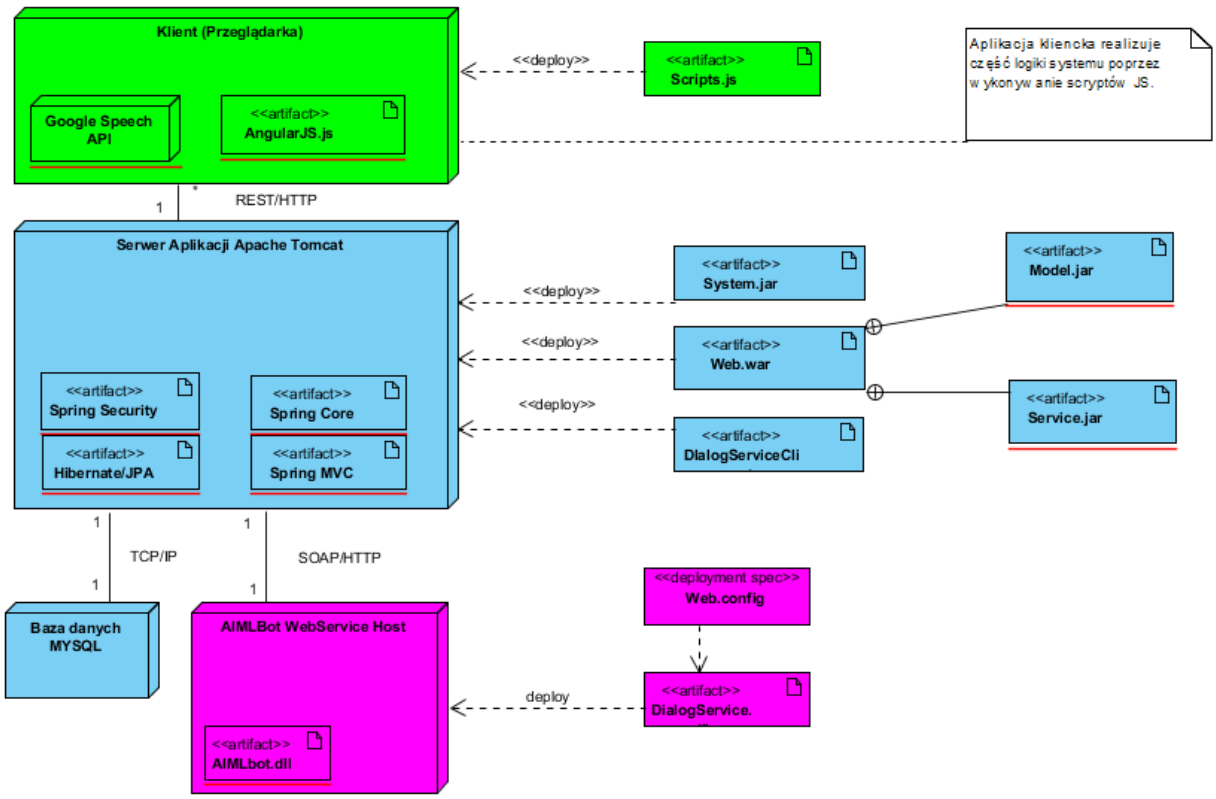
- Diagram pakietów (perspektywa wytworzenia)



- Diagram komponentów (perspektywa wytworzenia)



- Diagram rozmieszczenia (perspektywa wdrożenia)



Poniżej zamieszczano wymagania na fizyczne węzły przeznaczone do uruchamiania na nich systemu LanguageTutor. Wymagania podzielone zostały na wymagania w zakresie oprogramowania, jak i sprzętu.

Węzeł	Wymagania
Węzeł Serwera Aplikacji Apache Tomcat	Oprogramowanie: 1. Java JRE 1.7 2. Java SDK 3. Apache Tomcat v 7.0 4. Linux / Windows OS Sprzęt: 1. procesor – conajmniej : 2 x AMD Opteron 1.5GHz 2. pamięc RAM > 4GB
Węzeł Usługi AIMLBot WebService	Oprogramowanie: 1. .Net Framework (v. min. 4.0)

	2. IIS (v. min. 7.0) Sprzęt: 1. procesor – co najmniej : 2 x AMD Opteron 1.5GHz 2. pamięc RAM > 2GB
Węzeł klienta (Przeglądarki)	Oprogramowanie: 1. W pełni zaktualizowana przeglądarka Google Chrome z włączoną obsługą Java Scriptu (v. min. 25) Sprzęt – narzucone przez wymagania przeglądarki Google Chrome.

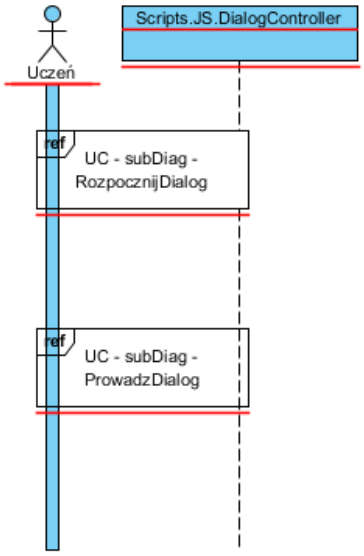
7. Realizacje przypadków użycia (Use-case realizations)

Poniżej przedstawiono realizację dwóch przypadków użycia wybranych do implementacji:

- **System umożliwia prowadzenie dialogów w formie głosowej (Przeprowadź dialog)**
- **System umożliwia wyświetlenie listy dostępnych dialogów użytkownikowi. (Wyświetl listę dialogów)**

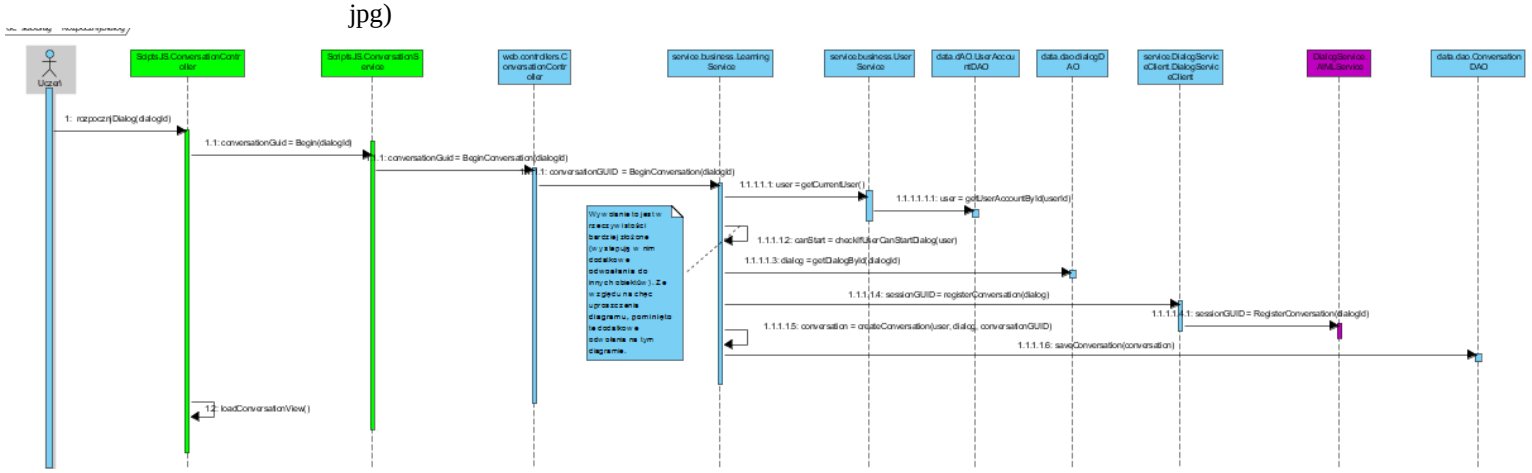
Na realizację przypadku użycia składają się diagram sekwencji wraz z odpowiadającym diagramem klas.
Ważne: Na **diagramach klas** pominięto skrypty JavaScript'u wykonujące się w przeglądarce użytkownika

1. Przeprowadź dialog

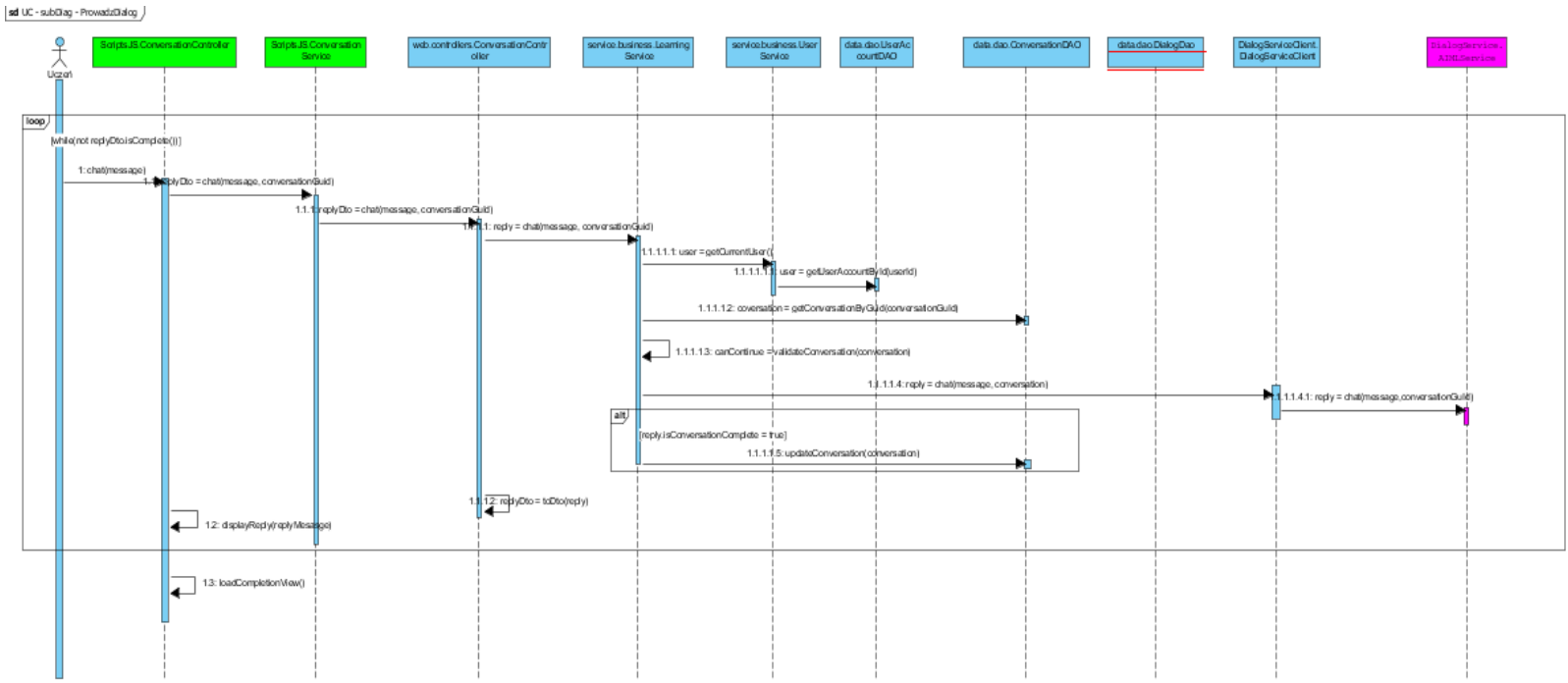


a. UC - subDiag - RozpocznijDialog (Diagram zostanie przesłany osobno na maila w postaci pliku

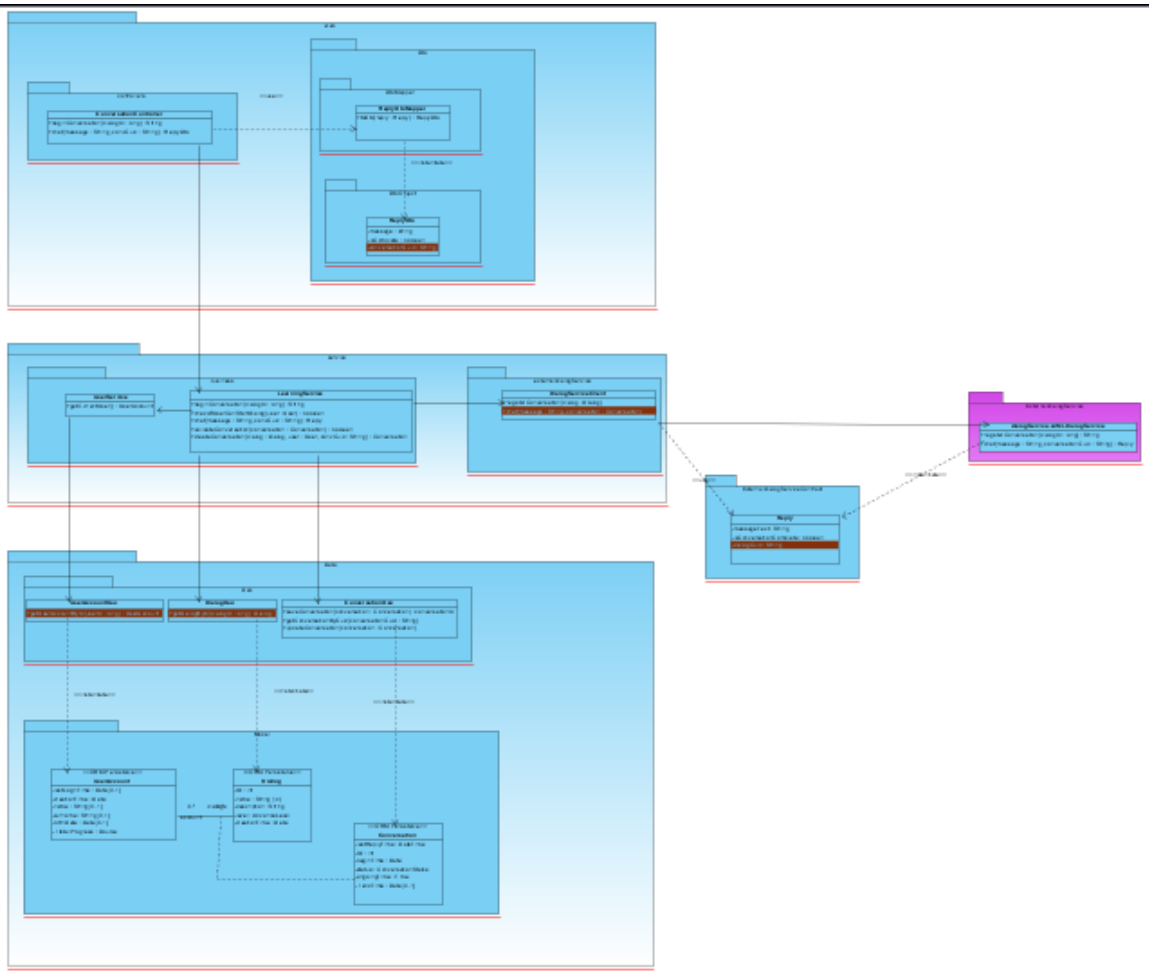
jpg)



b. UC - subDiag – ProwadzDialog (Diagram zostanie przesłany osobno na maila w postaci pliku jpg)

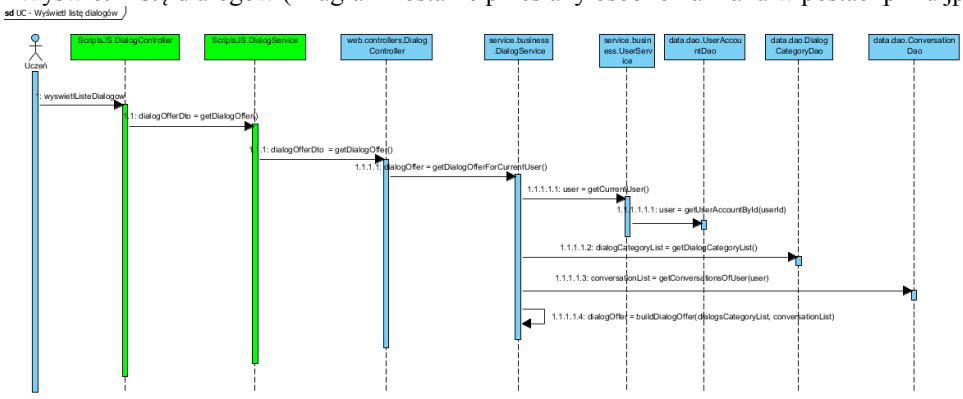


c. Przeprowadź dialog - UC realization – ClassDiagram (Diagram zostanie przesłany osobno na maila w postaci pliku jpg)



2. Wyświetl Listę Dialogów

a. UC - Wyświetl listę dialogów (Diagram zostanie przesłany osobno na maila w postaci pliku jpg)



a. Wyświetl listę dialogów - UC realization – ClassDiagram (Diagram zostanie przesłany osobno na maila w

postaci pliku jpg)