Ontologia Telefonów Komórkowych

Szymon Kołodziejczak
Tomasz Fengier
Tomasz Kot

Spis Treści

Wstęp **3**

Zastosowane technologie **3**

Front-end 3

Angular 3

SASS 4

Backend 5

Spring 5

Apache Jena 5

Ontologia 6

Protégé 6

Cellfie 6

LibreOffice Calc 6

Architektura rozwiązania **6**

Front-end **7**

Backend **10**

Ontologia **11**

Struktura 11

Import danych 12

Podsumowanie **13**

Wstęp

Aplikacja umożliwiająca wyszukiwanie telefonów oparta na ontologii telefonów komórkowych w formacie RDF. Użytkownikom udostępniona została możliwość formułowania zapytań filtrujących produkty na podstawie kryteriów takich jak model, marka, system operacyjny, cena i parametry sprzętowe. Projekt został zrealizowany w postaci aplikacji sieciowej.

Zastosowane technologie

Front-end

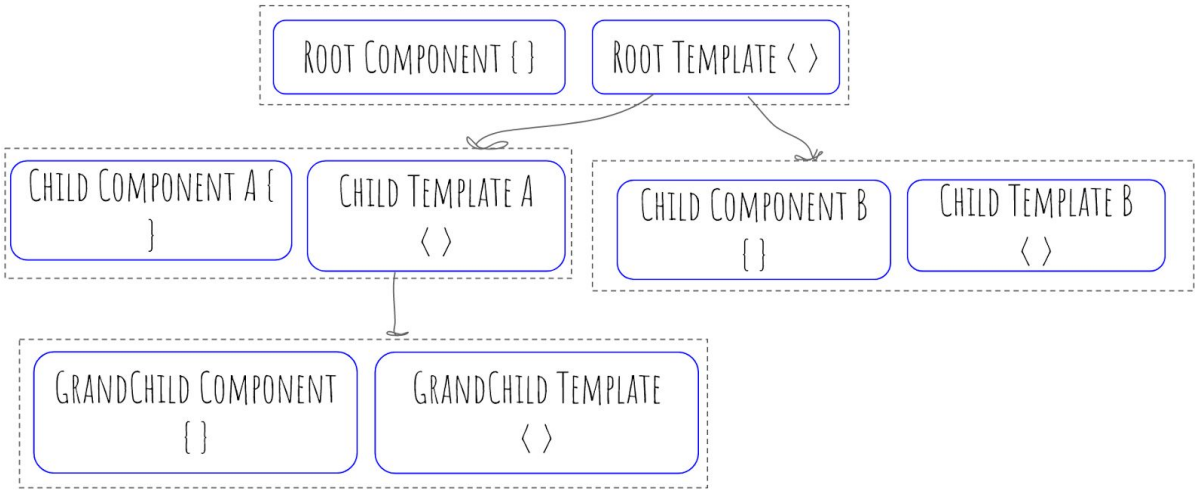
Angular

Angular to framework wspierany i firmowany przez google. Wspomaga on tworzenie i rozwój aplikacji SPA. Single Page Application to aplikacja, w której cały kod potrzebny do działania (HTML, CSS, JavaScript) przesyłany jest na początku lub dodawany dynamicznie w kawałkach, zwykle w odpowiedzi na interakcje generowane przez użytkownika.

Obecnie na rynku znajduje się wiele frameworków czy bibliotek, które są alternatywami dla angulara; przykładami może być Vue lub React. Zdecydowano się na Angular ponieważ framework ten narzuca pewne formy, ramy programistyczne, dzięki którym kod aplikacji jest łatwy w utrzymaniu i pielęgnacji.

Angular wymusza również Typescript - nadzbiór języka JavaScript, który wprowadza opcjonalne typowanie. Posiadanie informacji o typach zawsze jest lepsze niż ich brak.
Kompilator sprawdzający poprawność typowania jest w stanie wyeliminować całą gamę potencjalnych błędów. Edytory tekstu mają większą możliwość wnioskowania na temat kodu, co przyspiesza i ułatwia pracę programisty. Sam kod Typescriptu jest potem zamieniany na czysty javascript.

Architektura projektu w angularze jest oparta na komponentach. Każdy komponent ma swoją logikę (kod w Typescripcie), szablon HTML i plik arkusza stylów, którego reguły obowiązują tylko i wyłącznie w obrębie komponentu. Komponenty można dowolnie zagnieżdżać.



Prostokąt przerywaną linią reprezentuje komponent.
Ich zagnieżdżanie odbywa się poprzez umieszczenie tagu html (selektora komponentu, np.<mój></mój>) w widoku rodzica.

Angular cechuje modularność – komponenty muszą zostać zaimplementowane w ramach konkretnego modułu. Każdy moduł może eksportować wskazane komponenty i być importowany przez inne moduły.

SASS

Autorzy pracy nie skorzystali z żadnego gotowego frameworku CSS, których przykładów jest sporo - Bootstrap czy Material Design. Wszystkie reguły CSS zostały napisane od zera przy pomocy tytułowego rozszerzenia języka CSS.

Sass (Syntactically Awesome Style Sheets) jest preprocessorem CSS. Istnienie takich preprocessorów (innym przykładem może być LESS) to odpowiedź na brak podstawowych mechanizmów w CSS znanych z tradycyjnych języków programowania. Sass oferuje bogatą składnie - możliwe jest tworzenie zmiennych, pętli, instrukcji warunkowych
czy zagnieżdżanie reguł. Preprocessory koniec końców przekompilowują napisany kod
do formatu zgodnego ze standardem CSS.

Zastosowanie takich preprocessorów jest bardzo popularną praktyką i w znaczący sposób ułatwia pisanie reguł jak i utrzymywanie kodu.

Backend

Pierwotnie backend napisany został w Node.js z wykorzystaniem frameworka express.js oraz biblioteki rdfstore-js odpowiadającej za przechowywanie danych w postaci RDF oraz przetwarzanie zapytań SPARQL. Takie rozwiązanie ułatwiało komunikację pomiędzy frontendem i backendem umożliwiając wykorzystanie tego samego formatu danych (JSON) bez konieczności konwersji. W trakcie implementacji wyszły na jaw problemy wydajnościowe rdfstore-js (w przypadku niektórych zapytań zużycie zasobów systemowych wzrastało do poziomu uniemożliwiającego dalsze działanie programu), co wymusiło przejście na inny stos technologiczny wykorzystujący framework Spring oraz bibliotekę Apache Jena.

Spring

Jeden z najpopularniejszych frameworków wykorzystywanych do tworzenia aplikacji sieciowych w języku Java. W projekcie wykorzystany został Spring Boot umożliwiający konfigurację aplikacji przy wykorzystaniu kodu w Javie (w odróżnieniu od używanej dawniej konfiguracji w XML) oraz zapewniane przez Spring mechanizmy wstrzykiwania zależności.

Apache Jena

Biblioteka umożliwiająca przechowywanie i przetwarzanie danych w formacie RDF za pomocą API w języku Java lub w postaci samodzielnego serwera (Fuseki). Jena obsługuje w zapisane w wielu formatach, między innymi: TURTLE, JSON-LD i RDF Binary. Możliwe jest również przetwarzanie danych przy pomocy silników wnioskujących.

Ontologia

Protégé

Darmowe narzędzie open-source do projektowania ontologii. Rozwijany od 1999 roku przez Stanford. Udostępnia graficzny interfejs użytkownika oraz narzędzia do walidacji i wnioskowania na podstawie zdefiniowanych faktów. Protégé udostępnia interfejs do definiowania m.in. IRI bytów, klas, atrybutów, relacji. Środowisko może być rozszerzane o wtyczki pozwalające na zwiększenie funkcjonalności. Mogą to być moduły do importowania danych, formułowania zapytań SPARQL, dodatkowych widoków, czy narzędzi wnioskujących.

Cellfie

Wtyczka do protégé pozwalająca na generowanie faktów na podstawie tabel w formacie .xlsx. Ma możliwość dodawania do ontologii indywiduów, klas i zależności między nimi. Definiowanie faktów opiera się na zastosowaniu własnego języka reguł. Reguły te mają

postać zależności pomiędzy komórkami w poszczególnych kolumnach. Plik z regułami może być zapisany w formacie JSON.

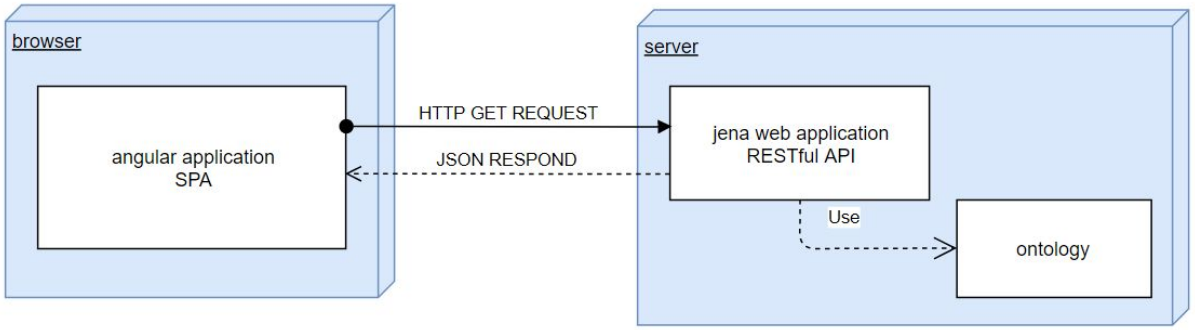
LibreOffice Calc

Narzędzie open-source do obróbki arkuszy kalkulacyjnych. Udostępnia funkcję zapisu i odczytu plików w formatach .csv, .xlsx, czy .odf. Program został użyty do importu danych telefonów, wstępnej obróbki i eksportu dla wtyczki cellfie.

Architektura rozwiązania

Rysunek architektury jest tak samo prosty jak architektura rozwiązania.
Front-end i back-end są osobnymi projektami, które działają niezależnie.
Kontroler aplikacji po stronie serwera obsługuje jedynie żądania GET, które nie są w żaden sposób autoryzowane. Serwer jest bezstanowy, co w tym kontekście oznacza, że za każdym razem odpowiedź będzie wyglądać tak samo. Odpowiedzi na żądanie są w formacie JSON.

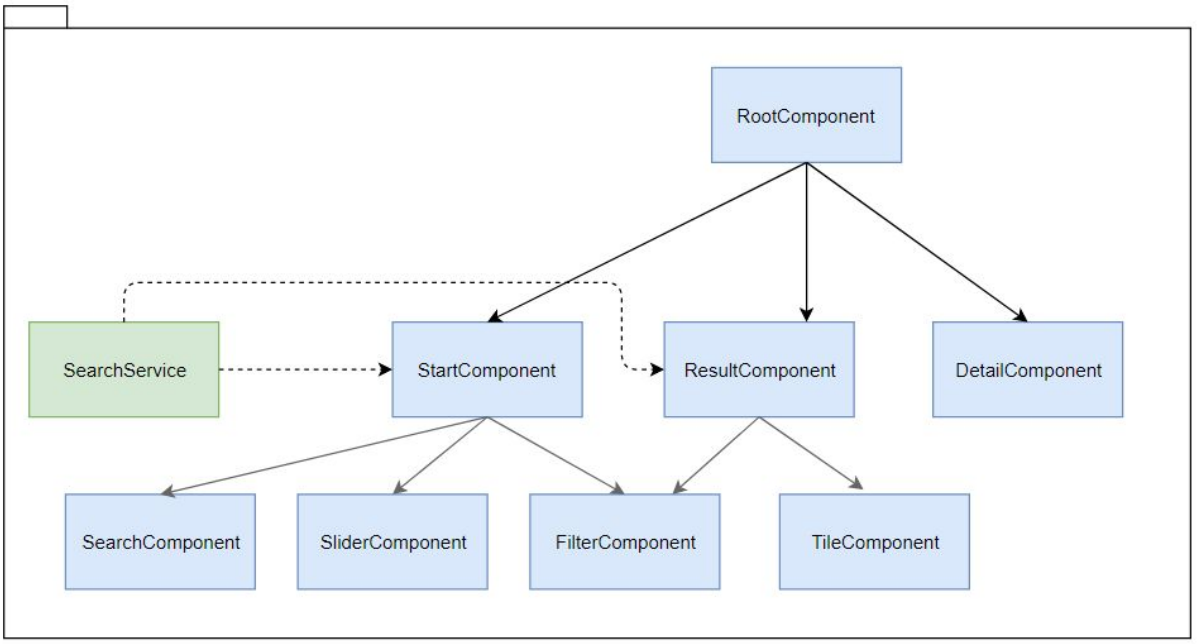
Po stronie serwera aplikacja internetowa podczas inicjalizacji odczytuje ontologie z pliku (jest ona przechowywana lokalnie). Wszystkie dane znajdują się w ontologii, nie ma żadnej bazy danych.



Użytkownik przy pomocy interfejsu graficznego generuje żądania GET do serwera o dane. Aplikacja SPA w angularze

Front-end

Całość modułu aplikacji przedstawiono poniżej:



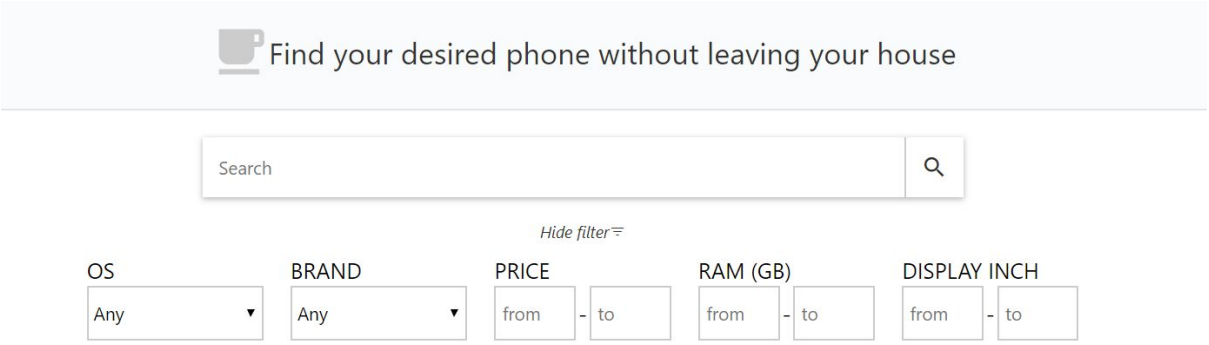
Moduł aplikacji przedstawiono przy pomocy drzewa komponentów.

Aplikacja SPA składa się z 3 widoków:

1. widok startowy (StartComponent)
2. widok listujący wyniki (ResultComponent)
3. widok prezentujący szczegóły modelu (DetailComponent)

To, który widok będzie widoczny dla użytkownika zależy od wewnętrznego routingu aplikacji. Komponent filtru jest używany w dwóch miejscach: widoku startowym i wynikowym.
Do tych widoków wstrzykiwany jest również jedyny serwis - SearchService. Odpowiada za wysyłanie żądań do serwera. Można go interpretować jako klasę statyczną dostępną w obrębie całego modułu.

Widok startowy



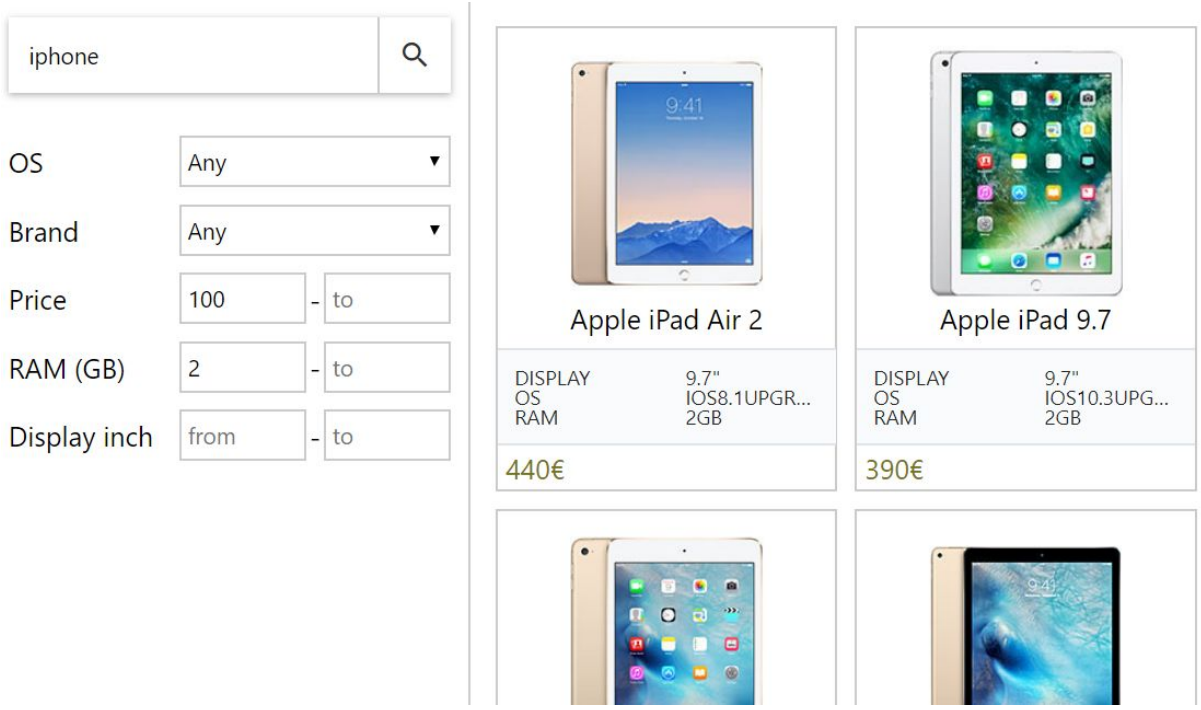
Widok startowy składa się z głównej kontrolki, przy pomocy której użytkownik może znaleźć konkretny model telefonu. Pod spodem widoczny jest filtr, w którym istnieje możliwość szukania telefonu na podstawie jego parametrów. Wszystkie parametry są opcjonalne. Na samym górze widoku znajduje się slajder, który jest zwykłym dodatkiem.

Wypełnione pola są używane do generowania URL w notacji matrix (czyli para wartość-klucz oddzielone są średnikiem). Poniżej przedstawiono przykład.

Rysunek przedstawiający przykładowe zapytanie użytkownika. Wypełnione pola zostały wyróżnione kolorem niebieskim przy pomocy narzędzia deweloperskiego.

Jedynie zdefiniowane parametry zostały uwzględnione w linku.

Parametr ‘search’ przekierowuje użytkownika do widoku ResultComponent (nazwa może być myląca, ponieważ istnieje w aplikacji SearchComponent - jest on jednak formularzem składającym się wyłącznie z kontrolki tekstowej). Przy inicjalizacji ResultComponent wywoływana jest metoda serwisowa, do której przekazywany jest pobrany URL. Uzyskanie odpowiedzi wiąże się z budowaniem widoku:



Po lewej stronie znajduje się filtr, tym razem w pionowej postaci. Filtr jest wypełniony parametrami przekazanymi w URL.
Kliknięcie w telefon wiąże się z nawigacją do widoku szczegółowego w którym to wylistowana jest pełna specyfikacja:

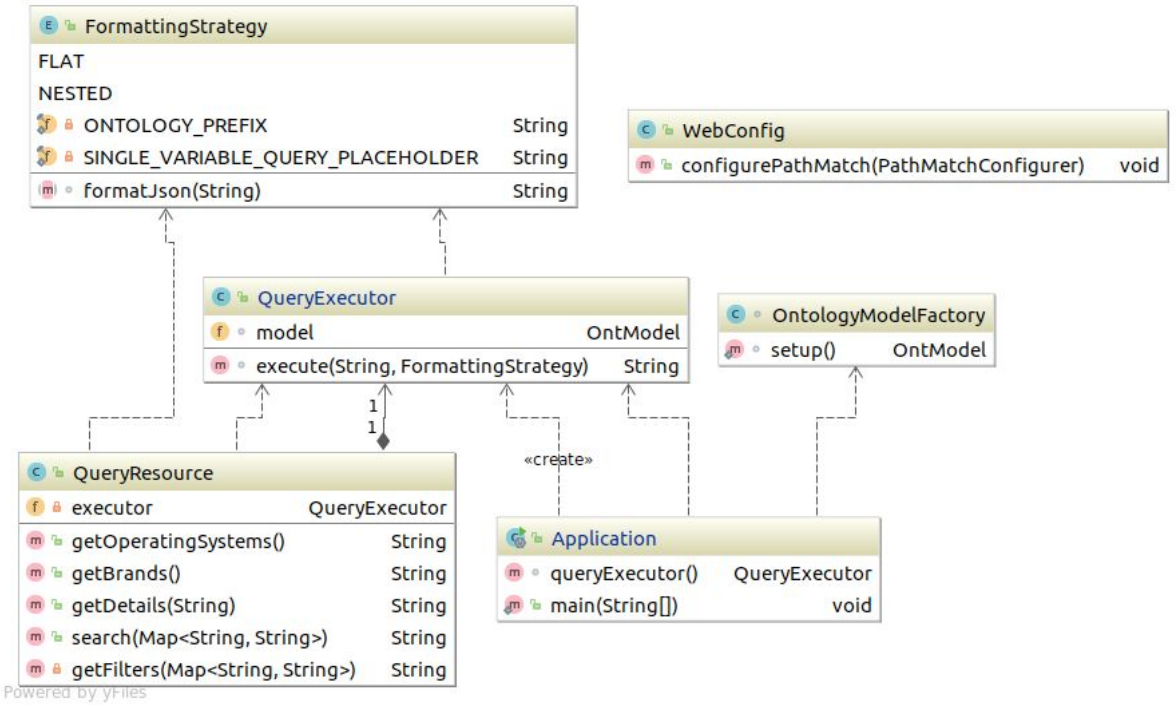
Fragment widoku szczegółowego.

URL tego widoku wygląda następująco:

Każdy telefon w ontologii ma unikalny identyfikator: połączenie marki i modelu.
W urlu pod query parametrem ‘model’ kryje się właśnie ten identyfikator, który jest przekazywany w żądaniu GET do serwera.

Wszystkie widoki aplikacji zostały przedstawione. Aplikacja jest prosta i można ją sprowadzić do postrzegania jako interfejs graficzny dla użytkownika do tworzenia zapytań SPARQL o telefony z ontologii.

Backend



Backend składa się z 6 klas:

- Application - stanowi tzw. punkt wejścia aplikacji uruchamiający serwer aplikacyjny Tomcat oraz wstrzykuje bean QueryExecutor.
- FormattingStrategy - definiuje typ wyliczeniowy zajmujący się sprowadzaniem wyników zapytania do postaci oczekiwanej przez frontend. Może przyjąc 2 wartości - NESTED oraz FLAT.
- OntologyModelFactory - klasa odpowiedzialna za inicjalizację ontologii danymi z pliku.
- QueryExecutor - bean przetwarzający zapytania SPARQL będący wrapperem na klasę OntModel dostarczaną przez Apache Jena.
- QueryResource - klasa definiująca REST API oraz tworząca zapytania SPARQL na podstawie zapytania HTTP.
- WebConfig - klasa konfiguracyjna umożliwiająca obsługę *matrix parameters*.

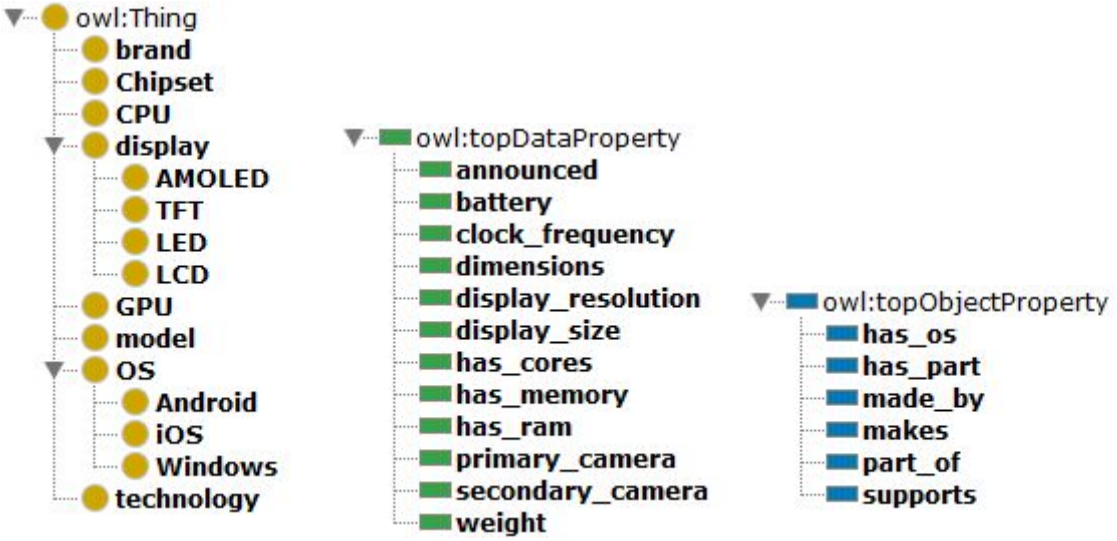
REST API udostępnia 4 metody:

- getBrands - zwraca listę marek telefonów
- getOperatingSystems - zwraca listę systemów operacyjnych
- getDetails(String model) - zwraca informacje o wybranym modelu telefonu
- search(Map<String, String> matrixVars) - wyszukuje telefony spełniające warunki podane w zapytaniu jako *matrix parameters*. Obsługiwane parametry to: price_from (dolna granica ceny), price_to (górna granica ceny), ram_from (minimalny rozmiar pamięci RAM w GB), ram_to (maksymalny rozmiar pamięci RAM w GB), os (system operacyjny)), brand (marka telefonu), display_from (minimalny rozmiar wyświetlacza

w calach), display_to (maksymalny rozmiar wyświetlacza w calach), query (nazwa modelu).

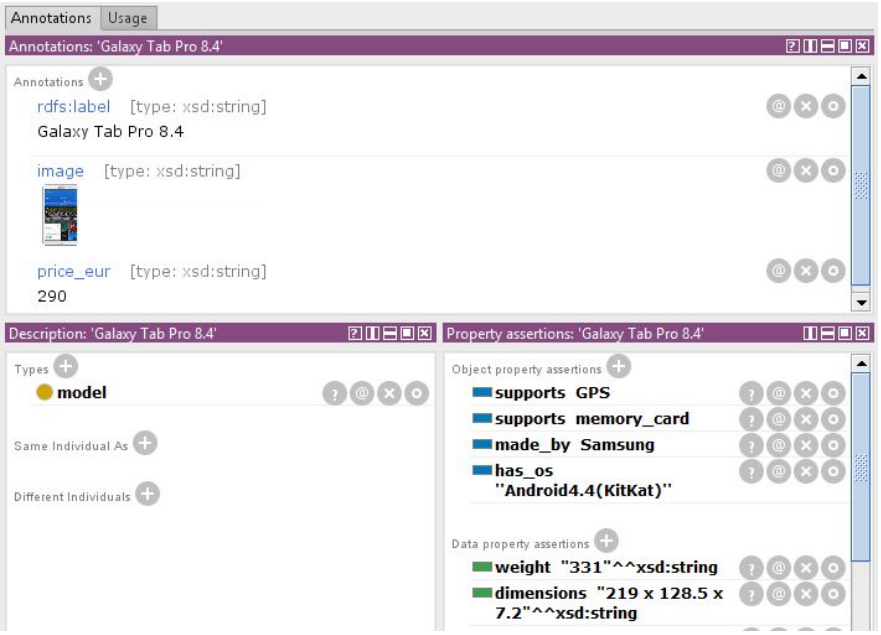
Ontologia

Struktura



Podstawowym pojęciem w ontologii z punktu widzenia systemu jest model telefonu. Zdecydowano że modele będą modelowane za pomocą indywiduów należących do klasy model, a nie jako podklasa. Podobnie każdy producent, procesor, karta graficzna i system operacyjny. IRI danego telefonu jest sklejeniem nazwy producenta i modelu, ponieważ istnieją modele różnych producentów o tej samej nazwie. Każdy rodzaj wyświetlacza (głębia kolorów, rodzaj eranu dotykowego) został zamodelowany jako oddzielny byt należący do klasy display. Jest ona podzielona na podklasy ze względu na rodzaj matrycy. Analogicznie systemy operacyjne - każdy z nich należy do nadrzędnej klasy OS, ale wyróżniane są dodatkowo trzy podklasy Android, iOS, Windows. W ontologii istnieją takie systemy, które nie należą do żadnej z tych podkategorii, a jedynie do głównej. Dodatkowo wyróżnia się pewien zbiór technologii, które może obsługiwać dany model. Należą do nich m.in. dual sim, LTE, NFC, czy radio. Aby móc wyszukiwać modele spełniające określone kryteria, zostały zdefiniowane relacje między bytami. Najważniejsze z nich to has_os, definiująca jaki system operacyjny ma dany model, has_part łączy model z każdą jego materialną częścią, made_by określa kto jest producentem, a supports opisuje jakie technologie obsługuje dany model. W ogólności nie każdy jest połączy każdą z tych relacji z innym indywiduem, poza producentem, który jest zdefiniowany dla każdego z nich. Aby umożliwić swobodniejsze konstruowanie zapytań SPARQL zostały zdefiniowane też relacje odwrotne. Proste wartości liczbowe zostały zamodelowane za pomocą data properties. Najważniejsze z nich to battery określający typ baterii (Li-Ion, Li-Po…), has_memory i has_ram określające odpowiednio ilość pamięci wewnętrzej i RAM w gigabajtach, display_size określa przekątną w calach, weight wagę w gramach. Ze względu na ograniczoną funkcjonalność narzędzia niektóre

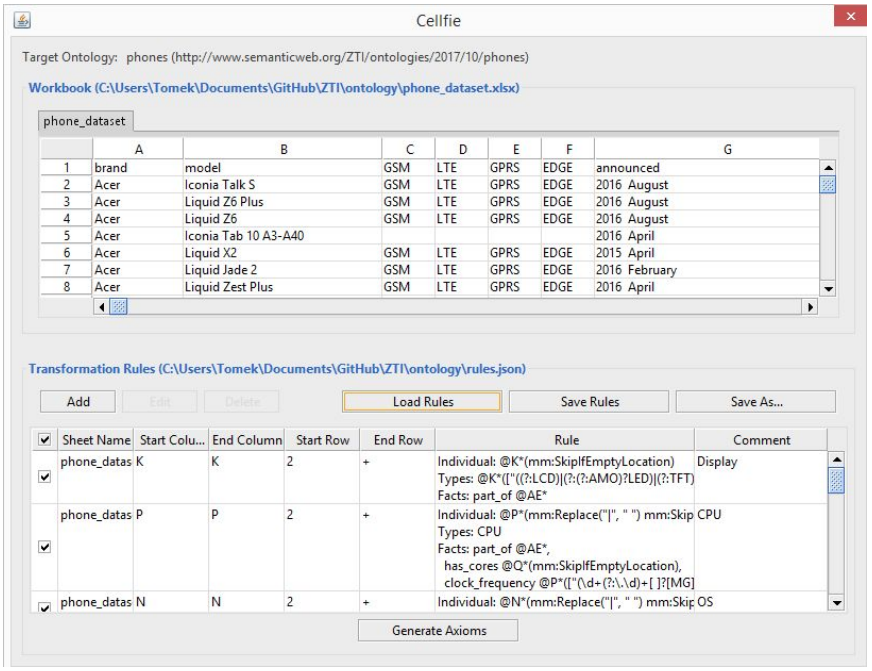
relacje typu data properties bez podanej wartości zostały zamodelowane jako puste ciągi znaków.



Powyżej przedstawiono przykładowy model z jego detalami w edytorze protege.

Import danych

Jako źródło danych wybrano gotową bazę danych z GSMArena. Dane zostały wstępnie przetworzone w programie LibreOffice Calc i zapisane w formacie .xlsx. Utworzony został szablon otologii w pliku template.owl, zawierajacy definicje klas i zależności między nimi. Dodanie poszczególnych indywiduów odbywa się przez wtyczkę cellfie. Utworzony plik .xlsx jest wczytywany wraz z dodatkowym plikiem rules.json, w którym przygotowane zostały reguły przetwarzania. Dane są dodawane do ontologii i zapisywane w pliku phones.owl.



Podsumowanie

Założeniem projektu było zaimplementowanie prostej, ale użytecznej aplikacji. Zbudowany system spełnia założone wymagania funkcjonalności, jednak ze względu na napotkane problemy konieczna była zmiana technologii. Dostępne obecnie rozwiązania dla języka JavaScript okazały się niewystarczające do zrealizowania pełnej funkcjonalności systemu. Konieczne była zatem zmiana technologii backendu na Javę. Narzędzia do importu ontologii pozwoliły na dość łatwe utworzenie bazy RDF, jednak w trakcie pracy napotkano wiele błędów i niedogodności w działaniu używanych programów, w szczególności wtyczki cellfie. Lepszym rozwiązaniem mogłoby być użycie innych technologii. Frontend został zrealizowany bez większych problemów i spełnia swoje zadanie.