Politechnika Poznańska

Hurtownie Danych

Dopasowanie baz referencyjnych TERYT i Poczty Polskiej

Marek Smorąg
Bartosz Rodak
Tomasz Dzięcioł
Anna Lehnhardt

17 czerwca 2021

Spis treści

1 Opis problemu

Celem projektu jest opracowanie mechanizmu, który jest w stanie po wypełnieniu części formularza zawężać dostępność wyboru w pozostałych polach w oparciu o pełny zestaw danych adresowych (województwo, powiat, gmina, miejscowość, kod pocztowy, nazwa ulicy, numer budynku).

Dane adresowe pobierane są z baz dostarczanych przez dwa różne podmioty GUS (baza TERYT) oraz Poczta Polska (baza Pocztowych Numerów Adresowych – PNA). Obie bazy nie mają wspólnych identyfikatorów w postaci kluczy sztucznych (np. ID z TERYT) Po pobraniu dane są zapisywane w bazie MySQL, która umożliwia odczyt, aktualizację oraz przechowywanie danych historycznych.

Główną trudnością zadania jest spięcie danych z dwóch różnych baz na podstawie zawartości oraz radzenie sobie z zaobserwowanymi anomaliami.

2 Technologie

Do wykonania projektu wykorzystano następujące technologie:

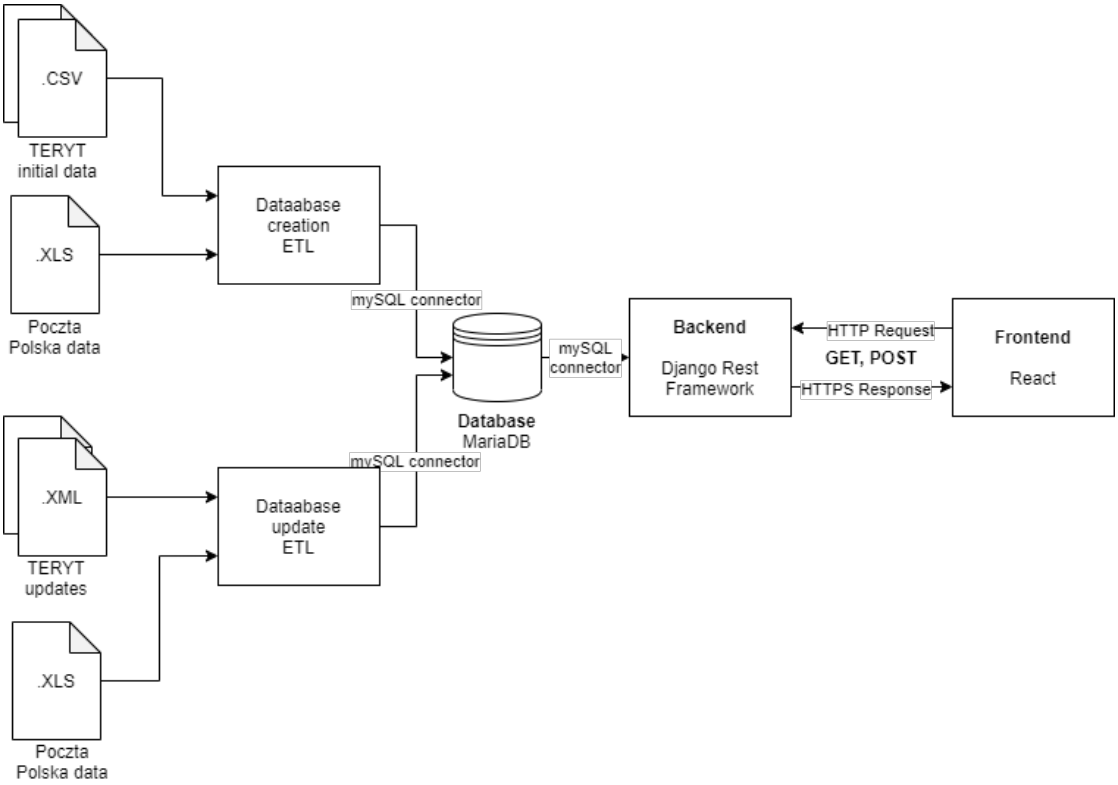
- baza danych: MySQL
- backend: Python, Django
- frontend: React, Material-UI
- prototyp projektu: Python, Flask, WTForms

3 Opis rozwiazania

3.1 Architektura systemu

Wejście systemu stanowią pliki płaskie pochodzące od dwóch niezależnych instytucji. Dostarczone dane są poddawane procesom ETL, aby ostatecznie umieszczone zostały w bazie hurtowni danych przyjmując opracowaną strukturę. Dzięki temu dane przygotowane są do wykorzystania ich w procesach analitycznych. Możliwe jest również dostarczenie plików aktualizacyjnych, które zostają użyte do zmodyfikowania informacji przechowywanych w hurtowni. Warstwa analityczna składa się z części serwerowej i klienckiej. Serwer (backend) odpowiedzialny jest za przetwarzanie załadowanej bazy danych w celu wyłuskania odpowiedzi na odebrane żądanie wysłane przez warstwę kliencką (frontend).

Opisany w ten sposób ogólny schemat hurtowni danych jest zaprezentowany na rysunku (Rys. 1).



Rysunek 1: Schemat poglądowy hurtowni danych

3.2 Architektura bazy danych

Baza danych podzielona jest na dwie części. Pierwsza część odpowiada za dane dotyczące lokalizacji i kodów pocztowych z Poczty Polskiej. Druga część odpowiada za lokalizacje pobrane z bazy TERYT.

Dane z Poczty Polskiej podzielone są na: województwa, powiaty oraz resztę (kody pocztowe wraz z ulicami, miejscowościami i gminami)

Dane z bazy TERYT przechowywane są w następującej strukturze. Tabele podzielone są na województwa, powiaty, gminy, miejscowości oraz ulice. W celu zapewnienia kompatybilności z backendem napisanym w Django, do tabel wygenerowano unikalne identyfikatory. W przypadku tabeli z ulicami jest to zwykły auto increment. Natomiast przy gminach oraz powiatach generowane zostają identyfikatory na podstawie unikalnych zbiorów jednostek administracyjnych, pod który dana podjednostka podlega (id województwa, id powiatu, id gminy). Na przykład dla każdego powiatu, unikalną parą będzie id województwa oraz id powiatu z bazy TERYT, które samo w sobie nie jest unikalne. Przy połączniu id wojewodztwa, równego 1 i id powiatu równego 12 otrzymany zostaje unikalny identyfikator 001012.

Aby przechowywać historię modyfikacji, do każdej z tabel zostały dodane kolumny: data modyfikacji oraz data wygaśnięcia, które sygnalizują kiedy dana wartość została zmodyfikowana oraz do kiedy jest ona aktualna. Prócz tego, żeby mieć wgląd na to jak dane wyglądały przed modyfikacją, dodano informujące o tym kolumny. Przykładem jest kolumna `nazwa przed` dla tabeli `trt_gminy`, która informuje o poprzedniej nazwie gminy obowiązującej przed modyfikacją.

Rysunek 2: Projekt bazy danych aplikacji

3.3 Opis procesu ETL

3.3.1 Utworzenie bazy danych

Bazę danych tworzona jest za pomocą skryptu wczytującego pliki SIMC, ULIC, TERC oraz slownik poczty-polskiej. Pliki SIMC, ULIC i TERC to pliki z aktualnymi danymi, które dotyczą podziału administracyjnego w Polsce. Można te pliki pobrać ze strony Głównego Urzędu Statystycznego. Pliki TERC zawierają dane odnośnie id województwa, nazwy województwa, id powiatu, nazwy powiatu, id gminy oraz nazwy gminy. Pliki SIMC zawierają informacje na temat miejscowości - nazwa miejscowości, id miejscowości oraz informacje na temat jednostek administracyjnych, do których miejscowość należy - id gminy, id powiatu, id województwa. Pliki ULIC zawierają dane na temat ulic - cecha ulicy (ul., al. pl.), nazwa ulicy, oraz informacje na temat jednostek administracyjnych, do których ulica należy - id miejscowości, id gminy, id powiatu, id województwa.

Skrypt łączy się konektorem z bazą MySQL, tworzy w niej tabele oraz nakłada klucze na odpowiednie kolumny. Następnie dla danych pochodzących z bazy TERYT wiersz po wierszu wczytuje i załadowuje dane do odpowiednich tabel w bazie danych. Dla Poczty Polskiej skrypt wczytuje dane wiersz po wierszu do jednej tabeli, następnie, bazując na powstałej tabeli, dzieli ją na 3 części - część z samymi województwami, część z samymi powiatami, oraz część z resztą danych.

3.3.2 Aktualizacje bazy danych

Aktualizacja bazy danych odbywa się w następujący sposób. Skrypt odczytuje pliki znajdujące się we wskazanym wcześniej folderze, następnie grupuje je według tego, z jakiego źródła pochodzą oraz czemu odpowiadają (pliki TERC bazy TERYT informujące o zmianach w województwach, powiatach oraz gminach, pliki SIMC informujące o zmianach w miejscowościach, pliki ULIC informujące o zmianach w ulicach, oraz pliki Poczty Polskiej, odpowiedzialne za część danych pocztowych). Z plików TERYT, które są w postaci dokumentów XML, odczytywane są kolejno elementy, które są oznaczone flagami D,U,M. Flagi oznaczają kolejno - Dodanie, Usunięcie, Modyfikacja. Na podstawie flagi oraz danych w elemencie XML, dla flagi D dodawany jest nowy rekord do tabeli oraz jako datę modyfikacji dopisywana zostaje aktualna data. Dla elementu z flagą U, aktualizowana jest krotka w tabeli o datę wygaśnięcia, która informuje o usunięciu rekordu. Dla flagi M, w tabeli aktualizowana zostaje krotka o datę modyfikacji, zmieniane są aktualne cechy takie jak np. nazwa oraz zapisana w tym przypadku zostaje do kolumny `nazwa-przed` nazwa, która widniała przed modyfikacją.

Dla bazy Poczty Polskiej wczytywany jest plik zawierający dane połączone - kod pocztowy, ulica, miejscowość, gmina, powiat, województwo oraz utworzona zostaje oddzielna tabela tymczasową. Następnie na podstawie utworzonej tabeli,

oraz już istniejącej, sprawdzane są oba przekroje A - B oraz B - A, po czym aktualizowana zostaje baza danych o wartości dodane i wartości odjęte, tak jak w przypadku danych z bazy TERYT.

miejscowosc	cecha	nazwa	data modyfikacji	data wygasniecia	cecha przed	nazwa przed
980850	ul.	Grota Roweckiego	2021-06-08	null	ul.	Roweckiego
84333	ul.	Biwakowa	2021-06-08	null	ul.	Przyjazna
515922	ul.	Bursztynowa	2021-06-08	null	ul.	Perłowa
934783	ul.	Aleja Niepodległości	2021-06-08	null	al.	Niepodległości
937110	ul.	1-go Maja	2021-06-08	null	ul.	1 Maja
805459	ul.	Jana Kwiecińskiego	2021-06-08	null	ul.	Kwiecińskiego
504479	ul.	MUBEA	2021-06-08	null	ul.	Mubea

Tabela 1: Fragment tabeli trt ulice

3.4 API

3.4.1 Projekt API

Dostęp do danych przechowywanych we wstępnie przygotowanych bazach zapewniany jest poprzez bezstanowy interfejs Rest API. Interfejs udostępnia możliwość pobrania wstępnej niefiltrowanej listy rekordów danego typu z bazy (użyteczne w pierwszej fazie ładowania interfejsu użytkownika) oraz wyników filtrowanych, bazujących na wstępnie wprowadzonych danych. Dostęp do bazowej funkcjonalności interfejsu nie wymaga autoryzacji.

W celu zapewnienia lepszej wydajności w odpowiedziach przesyłanych przez interfejs zastosowano mechanizm paginacji. Serwer w odpowiedzi domyślenie przesyła pierwszą cześć wyniku (domyślnie ograniczoną do 50 rekordów) wraz z informacjami o adresie kolejnych części i sumarycznej ilości rekordów wynikowych.

3.4.2 Punkty dostępowe Rest API

Struktura punktów dostępowych udostępnianych przez API odpowiada dostarczanym danym. Dla każdej z kategorii – województwa, powiaty, itd. utworzony został punkt dostępowy. Dodatkowo utworzony został punkt zapewniający funkcjonalność automatycznego uzupełniania danych na podstawie danych już wprowadzonych. Wszystkie metody `POST` korzystają z takiej samej struktury danych

stanu formularza.

Województwa - `/api/voivodeships` Dostęp do danych województw wymaganych przez interfejs użytkownika. Obsługiwane metody:

- `GET` - dostęp do niefiltrowanych danych
- `POST` - dostęp do przefiltrowanych danych - wymagany obiekt stanu formularza

Powiaty - `/api/counties` Dostęp do danych powiatów wymaganych przez interfejs użytkownika. Obsługiwane metody:

- `GET` - dostęp do niefiltrowanych danych
- `POST` - dostęp do przefiltrowanych danych - wymagany obiekt stanu formularza

Gminy - `/api/communes` Dostęp do danych gmin wymaganych przez interfejs użytkownika. Obsługiwane metody:

- `GET` - dostęp do niefiltrowanych danych
- `POST` - dostęp do przefiltrowanych danych - wymagany obiekt stanu formularza

Miejscowości - `/api/municipalities` Dostęp do danych miejscowości wymaganych przez interfejs użytkownika. Obsługiwane metody:

- `GET` - dostęp do niefiltrowanych danych
- `POST` - dostęp do przefiltrowanych danych - wymagany obiekt stanu formularza

Ulice - `/api/streets` Dostęp do danych ulic wymaganych przez interfejs użytkownika. Obsługiwane metody:

- `GET` - dostęp do niefiltrowanych danych
- `POST` - dostęp do przefiltrowanych danych - wymagany obiekt stanu formularza

Kody pocztowe - `/api/codes` Dostęp do danych kodów pocztowych wymaganych przez interfejs użytkownika. Obsługiwane metody:

- `GET` - dostęp do niefiltrowanych danych

- `POST` - dostęp do przefiltrowanych danych - wymagany obiekt stanu formularza

Autouzupełnianie - `/api/complete` Dostęp do funkcjonalności automatycznego uzupełniania pól. Obsługiwane metody:

- `POST` - dostęp do uzupełnionego obiektu stanu formularza - wymagany obiekt stanu formularza

Obiekt stanu formularza jest płaskim obiektem JSON mogącym zawierać pola z formularza wraz z ich aktualnym stanem. Obsługiwane pola to:

- voivodeship
- county
- commune
- municipality
- street
- number
- code

3.4.3 Przykładowe zapytanie

Działanie API można pokazać na przykładowym zapytaniu, w tym przypadku metodą `POST` do punktu dostępowego `/api/municipalities`

Dane zapytania

```
{
    "voivodeship": "",
    "county": "",
    "commune": "Lubartów",
    "municipality": "",
    "street": "",
    "number": "",
    "code": ""
}
```

Odpowiedź


```
{
    "count": 40,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 384555,
            "name": "Annobór"
        },
        {
            "id": 384561,
            "name": "Annobór-Kolonia"
        },
        {
            "id": 384578,
            "name": "Baranówka"
        },
        {
            "id": 384584,
            "name": "Brzeziny"
        },

            ...

        {
            "id": 1020499,
            "name": "Za Torem"
        },
        {
            "id": 1020507,
            "name": "Zabiele"
        }
    ]
}
```

4 Mechanizm przeszukiwania

W ramach zapytań `POST` API wykonuje przeszukiwania na przechowywanych w bazie rekordach.

W celu zapewnienia wydajności funkcjonalność przeszukiwania zrealizowana została na metodach dostarczanych przez silnik bazodanowy. Relacja między tabelami bazy danych realizowana jest na Modelach dostarczanych przez Django Framework. Umożliwia to odwoływanie się do wartości z kluczy obcych jak do zagnieżdżonych obiektów.

Algorytm wyszukiwania można opisać dzieląc go na trzy głtówne fragmenty. Wykorzystanie poszczególnych fragmentów różnie się w zależności od wyszukiwanej danej i realizowanej funkcjonalności.

4.1 Dane podziału administracyjnego

Dane pochodzące bezpośrednio z systemu Teryt, tj. województwo, powiat, gmina, miasto i ulica, wykorzystywane są w każdym udostępnianym przez API punkcie końcowym. Dane te są używane jako baza dla wszystkich wyszukiwań, wzór wg którego dopasowywane zostają dane ze zbioru dostarczanego przez Pocztę Polską. Ze względu na konstrukcję systemu dane wprowadzone przez użytkownika dzielone są na już uzupełnione oraz właśnie uzupełniane.

Pierwsze, już wprowadzone, wybrane przy pomocy innego punktu dostępowego, np. wprowadzone wcześniej dane województwa istniejące już podczas wyszukiwania i wybierania gminy, mają pełną zgodność co do brzmienia z danymi zawartymi w zbiorze Teryt. Dane te wykorzystane są do porównań dokładnych, dodając kolejny warunek do budowanego zapytania, dla każdego uzupełnionego w momencie wyszukiwania pól formularza.

Drugie, dane w trakcie wprowadzania - do których dopiero dopasowane zostanie brzmienie konkretnego rekordu z danych Terytu. Tu używane jest porównanie niedokładne dostarczane przez Django, odpowiednik programistycznego zawiera się, pomijającego wielkość liter. Porównanie to stosowane jest jedynie w sugestiach kolejnych nazw, gdzie użytkownik będzie finalnie musiał wybrać brzmienie zgodne z referencyjną bazą Teryt.

4.2 Dane kodów pocztowych

Wyszukiwanie danych kodów pocztowych różni się od pozostałych danych. Tu szczególną uwagę należy zwrócić na dwa aspekty - format kodu oraz dopasowanie danych o jednostce administracyjnej zawierającej wyszukiwany kod. Źródła Poczty Polskiej, w tym kody, zawierają odniesienia do własnych danych admi-

nistracyjnych, posiadających różniące się od tych zawartych w źródłach Terytu brzmienia i strukturę.

Implementacja wyszukiwania w przypadku danych pocztowych zakłada dopasowanie głównych podziałów administracyjnych (województwo, powiat, gmina) w sposób dokładny (z dokładnością do kapitalizacji) - wynika to ze standaryzacji brzmień, identycznej struktury danych oraz braku stosowania skrótowców w obu zbiorach danych. Dopasowanie nazw miejscowości i ulic polega natomiast na dopasowaniu możliwie największej liczby elementów (słów) wchodzących w skład nazwy danej kategorii. Takie rozwiązanie wynika z występujących różnic w brzmieniu nazw rekordów, stosowania skrótowców oraz nazw dzielnic.

4.3 Autouzupełnianie

Funkcjonalność autouzupełniania polega na kaskadowym uzupełnianiu kolejnych pól struktury formularza, zaczynając od pól najbardziej szczegółowych - kod pocztowy, ulica, kończąc na województwie. Algorytm uzupełniania pól na podstawie kodu pocztowego odpowiada temu z wyszukiwania kodów pocztowych. Uzupełniane są jedynie pola dla których udało się znaleźć pojedyncze pasujące rozwiązanie, tj. dla samych kodów pocztowych oznacza to precyzję uzupełniania od uzupełnienia do gminy aż do uzupełnienia do ulicy włącznie.

4.4 Przykłady

Pola puste zapytań i odpowiedzi zostały pominięte w celu poprawy czytelności. Ze względu na rozszerzone zapytania i możliwość mapowania rekordów do modeli, przedstawiona została struktura logiczna zapytań.

4.4.1 Wyszukiwanie gminy

Dane odebranego żądania:

```
{
    "voivodeship": "LUBELSKIE",
    "county": "lubartowski",
    "commune": "",
    "municipality": "",
    "street": "",
    "number": "",
    "code": ""
}
```

Generowane zapytanie w bazie danych:

Wybierz z `TRT_Gminy`, gdzie:

- nazwa województwa dokładnie równa "LUBELSKIE"
- nazwa powiatu dokładnie równa "Lubartowski"

Odpowiedzą jest lista par identyfikatora i nazwy gmin w powiecie Lubartowskim w województwie Lubelskim.

4.4.2 Wyszukiwanie kodu pocztowego

Dane odebranego żądania:

```
{
    "voivodeship": "LUBELSKIE",
    "county": "lubartowski",
    "commune": "",
    "municipality": "",
    "street": "Bł. Ojca Honorata Koźmińskiego",
    "number": "",
    "code": ""
}
```

Generowane zapytanie w bazie danych:

Wybierz z `PP_Adresy`, gdzie:

- nazwa województwa adresu równa z pominięciem kapitalizacji wartości"LUBELSKIE" **i**
- nazwa powiatu adresu równa z pominięciem kapitalizacji wartości "Lubartowski" **i**
- nazwa ulicy adresu zwiera słowo
  - "Bł" **lub**
  - "Ojca" **lub**
  - "Honorata" **lub**
  - "Koźmińskiego"

Odpowiedzą jest lista par identyfikatora i kodu pocztowego w powiecie Lubartowskim w województwie Lubelskim, poprawnych dla ulicy "Bł. Ojca Honorata Koźmińskiego":


```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": 384555,
            "code": "21-100"
        },
    ]
}
```

4.4.3 Automatyczne uzupełnianie jednoznacznych pól

Dane odebranego żądania:

```
{
    "voivodeship": "",
    "county": "",
    "commune": "Buk",
    "municipality": "",
    "street": "",
    "number": "",
    "code": ""
}
```

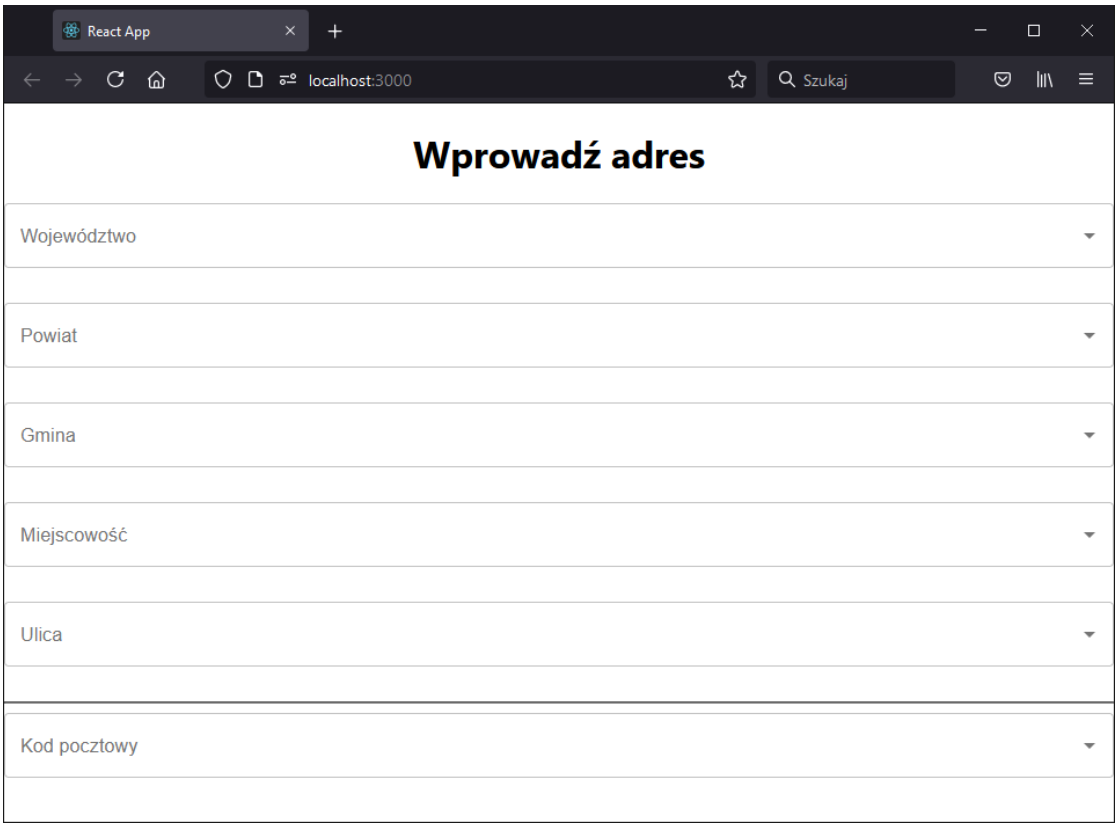
W bazie danych generowanych jest wiele zapytań, które próbują określić jednoznaczną wartość brakujących jednostek administracyjnych. Poszukiwanie opiera się na ograniczeniach równościowych odpowiednich pól otrzymanego żądania, które posiadają już zdefiniowaną wartość. Budowanie składni zapytania dla poszukiwania wartości konkretnej jednostki administracyjnej odbywa się analogicznie jak we wcześniej opisanych przykładach 4.4.1 i 4.4.2.

Odpowiedzą jest obiekt otrzymany w aktualnym żądaniu, który zostaje uzupełniony o brakujące oczywiste wartości:

```
{
    "voivodeship": "WIELKOPOLSKIE",
    "county": "poznański",
    "commune": "Buk",
    "municipality": "",
    "street": "",
    "number": "",
    "code": "64-320"
}
```

5 Interfejs użytkownika

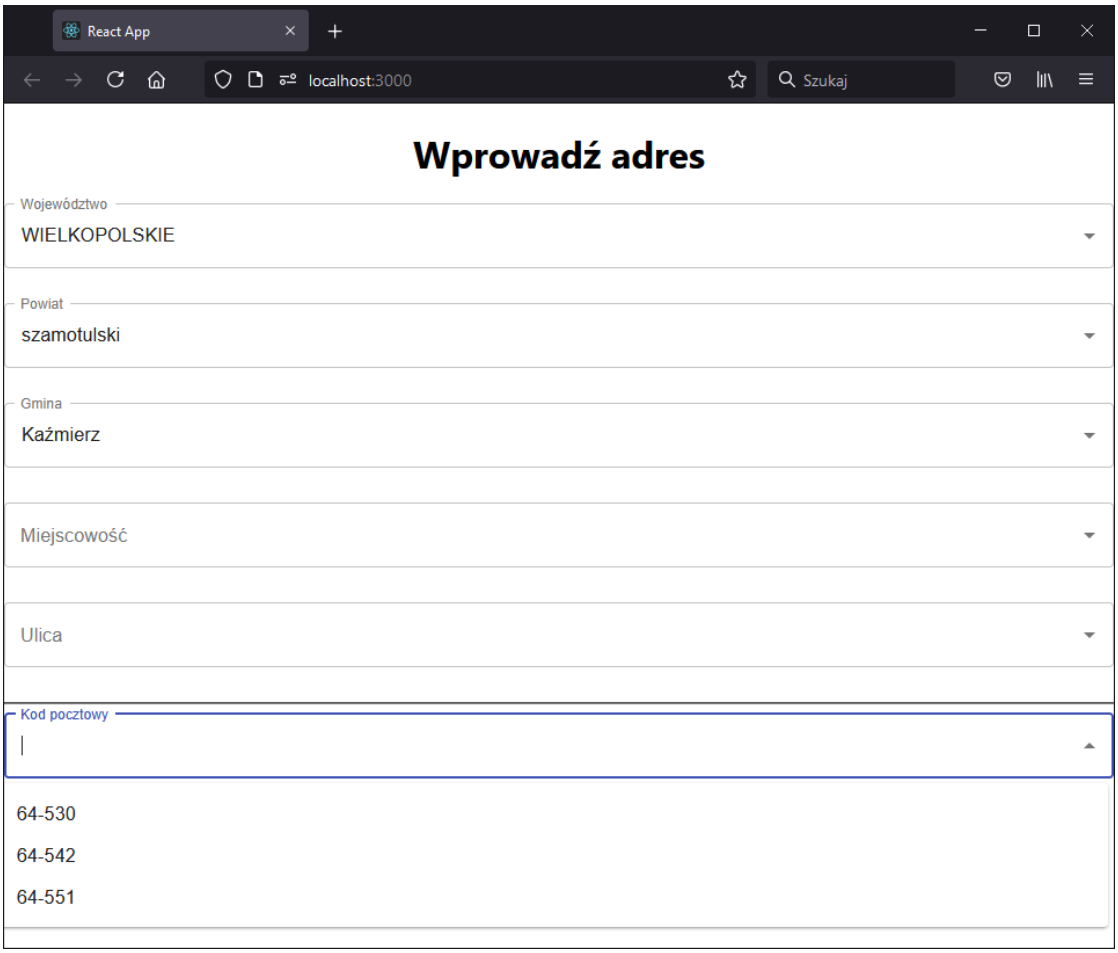
Końcowym elementem analizy danych zawartych w hurtowni jest interfejs graficzny uruchamiany w przeglądarce internetowej. Interfejs ten pośredniczy między użytkownikiem, który poszukuje informacji, a warstwą serwerową, odpowiedzialną za poprawne pozyskiwanie danych przechowywanych w hurtowni danych. Na obrazie (Rys. 3) przedstawiony został bazowy wygląd aplikacji.



Rysunek 3: Widok ekranu startowego

5.1 Funkcjonalność

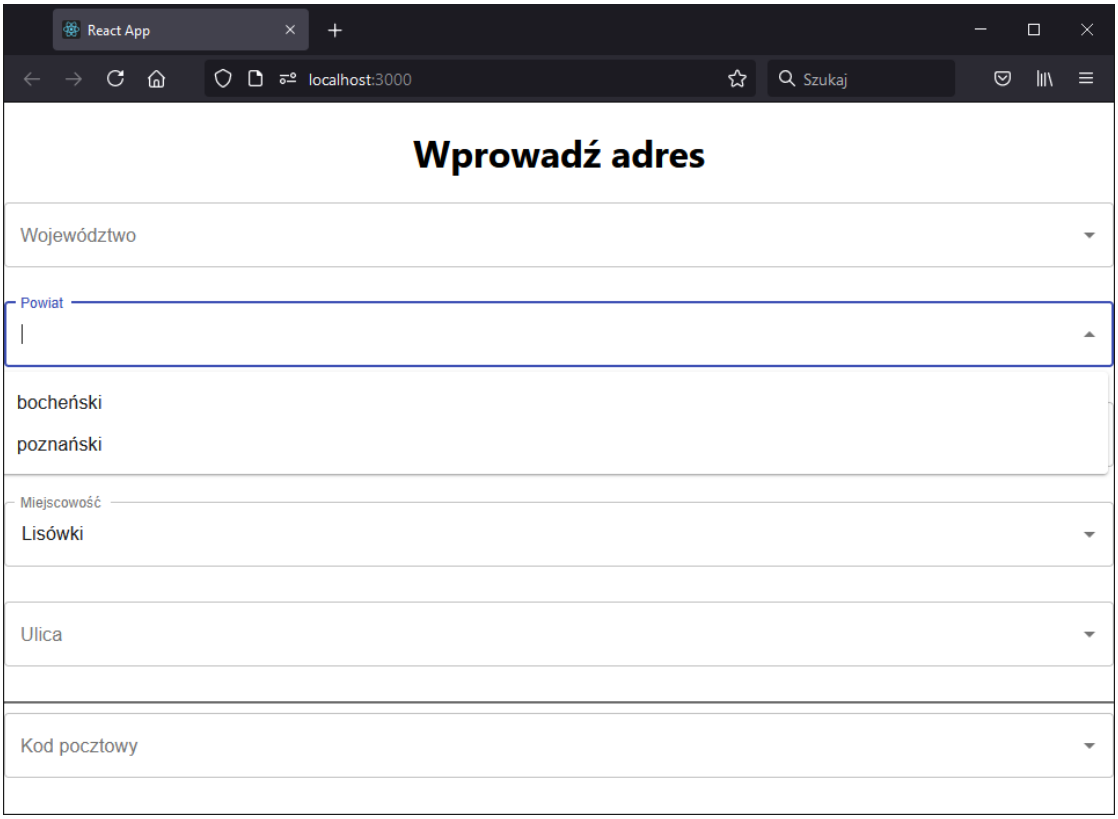
Celem aplikacji jest pozwolenie na wpisywanie nazwy danej jednostki terytorialnej, które wraz z kolejno podanymi znakami będzie podpowiadało w ostatecznym wyborze spośród dostępnego zbioru pasujących nazw. Oprócz ograniczania wyboru na podstawie wskazanego początkowego fragmentu nazwy obowiązuje ograniczenie ze względu na poprawną wspólną spójność pozostałych nazw administracyjnych szukanej lokalizacji. Końcowym efektem jest możliwość wyboru kodu pocztowego. Przykładowy efekt wypełniania formularza ukazany jest na obrazie (Rys. 4). Przy odpowiednio dokładnym wypełnieniu jednostek terytorialnych będzie sprowadzać się to do wyboru spośród jedynego możliwego kodu pocztowego.



Rysunek 4: Uzupełnianie danych TERYT w zachowanej kolejności

Aplikacja umożliwia również automatyczne uzupełnianie danych na podstawie tylko i wyłącznie podanego kodu pocztowego. W sytuacji, gdy wskazany kod pocztowy jednoznacznie identyfikuje na przykład daną gminę, to automatycznie uzupełnione zostają poprawne wartości dla województwa, powiatu i gminy. Ponieważ dalej kod przypisany jest wszystkim miastom we wskazanej gminie, to pole wyboru dla miejscowości zostaje nieuzupełnione.

Zastosowane mechanizmy autouzupełniania po stronie serwera służą także do zwolnienia użytkownika z konieczności podawania nazw na wszystkich poziomach jednostek terytorialnych. Wskazanie unikalnej nazwy wybranego poziomu szczegółowości lokalizacji pozwala na jednoznaczne uzupełnienie pól na wyższym szczeblu administracyjnym. Podając przykładowo unikalną w skali kraju nazwę miejscowości *Hel* następuje samoistne wypełnienie się pól dla gminy *Hel*, powiatu *puckiego* i województwa *pomorskiego*. Jeżeli wprowadzona nazwa nie jest jednak unikalna, to pola nie są wypełniane, aczkolwiek lista dostępnych opcji zostaje znacząco ograniczona (Rys. 5).



Rysunek 5: Uzupełnianie danych TERYT w dowolnej kolejności

5.2 Architektura

Aplikacja kliencka została wykonana przy użyciu biblioteki *React.js* należącej do języka *JavaScript*. Jej główną zaletą przydatną w stworzonym interfejsie jest interaktywność uzyskana poprzez tworzenie komponentów zawartości strony html w zależności od aktualnego stanu aplikacji. W przypadku opracowanego rozwiązania, dla danego pola wyboru stanem jest lista obecnych propozycji. W skutek wpisywania kolejnych liter nazwy w polu, aktywowane zostaje pobranie informacji z warstwy serwerowej, które to aktualizuje wspomnianą listę dostępnych nazw do wyboru. Ponieważ następuje zmiana wartości listy należącej do stanu komponentu, to następuje ponowne wyrenderowanie danego fragmentu strony. Dzięki temu odświeżeniu zawartości możliwe jest oglądanie aktualnej listy wyboru dla nazwy danej jednostki terytorialnej.

Kluczowym komponentem kodu aplikacji jest komponent nadrzędny, który odpowiedzialny jest za generowanie wszystkich kolejnych pól wyboru (Rys. 6). Kontroluje on w swoim stanie wprowadzone i zatwierdzone nazwy danych jednostek administracyjnych. Dzięki temu możliwe jest przekazanie aktualnego kontekstu przeszukiwania do pola wyboru danej jednostki terytorialnej, jeśli w innych polach wyboru zostały zatwierdzone już wartości dla innych szczebli administracyjnych.

```
class AddressVerifierComponent extends Component {
  constructor() {
    super();

    this.state = {
      voivodeship: null,
      county: null,
      commune: null,
      municipality: null,
      street: null,
      number: null,
      code: null
    }

    this.executeCompleteFetch = this.executeCompleteFetch.bind(this);
  }

  render() {
    return (
      <div id="main-component-div-address-verifier"
        className="main-component-div">
        <h1>Wprowadź adres</h1>
        <VoivodeoshipProperty initializedRequestParam={this.state}
          executeCompleteFetch={this.executeCompleteFetch} />
        <CountyProperty initializedRequestParam={this.state}
          executeCompleteFetch={this.executeCompleteFetch} />
        <CommuneProperty initializedRequestParam={this.state}
          executeCompleteFetch={this.executeCompleteFetch} />
        <MunicipalityProperty initializedRequestParam={this.state}
          executeCompleteFetch={this.executeCompleteFetch} />
        <StreetProperty initializedRequestParam={this.state}
          executeCompleteFetch={this.executeCompleteFetch} />
        <hr />
        <PostCodeProperty initializedRequestParam={this.state}
          executeCompleteFetch={this.executeCompleteFetch} />
      </div>
    )
  }
}

```

Rysunek 6: Podstawowy fragment kodu dla nadrzędnego komponentu `AddressVerifierComponent`

Dodatkową biblioteką wykorzystaną przy konstrukcji interfejsu jest biblioteka *Material-UI*. Dostarcza ona mnóstwo komponentów gotowych do wykorzystania w *React'ie*, które oferują szeroką funkcjonalność i mają nadany przyjemny styl graficzny. Pole wyboru zastosowane przy podawaniu danej nazwy administracyjnej jest utworzone jako komponent *Material-UI*, kod takiego komponentu przedstawiony jest na obrazie (Rys. 7). W elemencie `Autocomplete` wykorzystano możliwość definiowania funkcji uruchamianych na skutek różnych zdarzeń - w momencie wprowadzenia znaku (`onInputChange`), zatwierdzenia wyboru (`onChange`), aktywowania pola (`onOpen`). Domyślny styl komponentu i dostarczone drobne animacje przejścia podczas interakcji z nim są satysfakcjonujące. Dzięki temu nie jest konieczne ingerowanie w charakter i wygląd tego elementu.

```
render() {
  return (
    <div className="property-div">
      <Autocomplete
        value={this.props.initializedRequestParam.commune}
        onChange={(event, newValue) => {
          if (newValue === null) newValue = ""

          this.requestParam.commune = newValue;
          this.props.executeCompleteFetch(this.requestParam);
        }}
        onInputChange={(event, newValue) => {
          this.notOnInputChange = false;
          this.requestParam.commune = newValue;
          this.executeFetch();
        }}
        onOpen={(event) => {
          this.executeFetch();
        }}
        id="combo-box-commune"
        className="property-combo-box"
        options={this.state.communeOptions}
        renderInput={(params) => {
          return (
            <TextField {...params} label="Gmina" variant="outlined" />
          )
        }}
      />
    </div>
  )
}
```

Rysunek 7: Wykorzystanie komponentu `Autocomplete` biblioteki *Material-UI*