POLSKO-JAPOŃSKA AKADEMIA
TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria Oprogramowania i Baz Danych

Jakub Markiewicz
Nr albumu s25878

Elastyczny system do analizy wielowymiarowych danych z zastosowaniem grafiki trójwymiarowej

Praca magisterska napisana pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, lipiec, 2023

Streszczenie

Praca koncentruje się na problemie wizualizacji wielowymiarowych danych, które bez odpowiednich narzędzi są nieczytelne i tym samym stają się bardzo trudne w analizie. Zbadano istniejące na rynku narzędzia, które pozwalają na ich wizualizacje, aczkolwiek integracja z nimi własnego źródła danych jest trudna, a niekiedy nawet niemożliwa. Zauważono też brak możliwości filtrowania danych źródłowych, co powoduje wydłużenie procesu projektowania oraz konfigurowania wizualizacji. Brak sposobu na dynamiczne filtrowanie prowadzi również do redundancji danych. Te powody sprawiają, że poprawne zaprezentowanie danych wymaga specjalistycznej wiedzy, co w znaczny sposób zmniejsza dostępność narzędzia.

Na podstawie przeanalizowanych rozwiązań konkurencyjnych, opisano listę wymagań jakie narzędzie powinno spełniać i na ich podstawie zostało zaproponowane rozwiązanie. W dalszej części przedstawiono projekt prototypu jak i również opisano generyczne podejście do problemu niniejszej pracy.

Prototyp został stworzony w technologiach webowych, a każda integracja zewnętrznych źródeł oraz wizualizacja jest implementowana jako niezależna wtyczka. Przedstawione narzędzie obsługuje integracje z relacyjnymi bazami danych MySQL, PostgreSQL oraz nierelacyjną bazą danych MongoDB. Istnieje możliwość wizualizacji danych poprzez trójwymiarowy wykres słupkowy, punktowy, płaszczyznowy oraz kostkę OLAP.

Zakończeniem pracy jest podsumowanie, które zawiera informacje na temat spełnienia postawionych wymagań, zalet i wad zaproponowanego rozwiązania oraz dalsze możliwości rozwoju prototypu.

Słowa kluczowe:

wizualizacja 3D, wielowymiarowe dane, analiza danych

Spis treści

1. WSTĘP 5

1.1. Cel pracy 5
1.2. Rozwiązania przyjęte w pracy 5
1.3. Rezultaty pracy 5
1.4. Organizacja pracy 6

2. ISTNIEJĄCE ROZWIĄZANIA 7

2.1. Chart Studio od plotly.com 7
2.2. Looker Studio od Google 8
2.3. ChartBlocks 9
2.4. Live Editor AMCharts 10
2.5. Podsumowanie 11

3. PROPOZYCJA ROZWIĄZANIA 12

3.1. Wnioski po analizie istniejących rozwiązań 12
3.2. Wymagania 12
3.2.1 Filtrowanie danych 12
3.2.2 Integracja z różnymi źródłami danych 13
3.2.3 Wizualizacja danych 13
3.2.4 Zapis projektu w systemie 14
3.2.5 Udostępnianie skonfigurowanej wizualizacji 14

4. ZASTOSOWANE TECHNOLOGIE 15

4.1. HTML 15
4.2. CSS 16
4.3. Tailwind CSS 16
4.4. JavaScript 17
4.5. TypeScript 17
4.6. React.js 17
4.7. React Query Builder 18
4.8. Three.js 18
4.9. Vis.js 18
4.10. Lexpjs 19
4.11. Node.js 19
4.12. Next.js 19
4.13. MySQL 19
4.14. PostgreSQL 20
4.15. MongoDB 20
4.16. Prisma.js 20
4.17. Lerna.js 20

5. PROJEKT I IMPLEMENTACJA 22

5.1. Architektura rozwiązania 22
5.2. Baza danych 23
5.3. Wtyczki 25
5.3.1 Wtyczka wizualizacyjna 25
5.3.2 Wtyczka integrująca zewnętrzne źródło danych – klient 26
5.3.3 Wtyczka integrująca zewnętrzne źródło danych – serwer 26
5.4. Menedżer wtyczek 27

5.4.1 Tworzenie wtyczek 27
5.4.2 Rejestracja wtyczki 28
5.4.3 Udostępnione komponenty dla wtyczek 31
5.5. Aplikacja 32
5.5.1 Wyszukiwanie i wczytywanie wtyczek 32
5.5.2 Osadzanie wtyczek 33
6. ZASTOSOWANIE PROTOTYPU APLIKACJI 35
6.1. Logowanie 35
6.2. Panel główny 37
6.3. Tworzenie wizualizacji 37
6.4. Konfiguracja wizualizacji 38
6.4.1 Integracja z zewnętrznym źródłem danych 39
6.4.2 Konfiguracja wizualizacji 43
6.4.3 Publikacja wizualizacji 51
6.5. Dynamiczne filtrowanie 53
7. PODSUMOWANIE 57
7.1. Wady i zalety rozwiązania 57
7.2. Potencjalne kierunki rozwoju 58
7.3. Wnioski 58
BIBLIOGRAFIA 60
SPIS TABEL 62
SPIS RYSUNKÓW 63
SPIS LISTINGÓW 65

1. Wstęp

Dzisiejsze firmy dziennie zbierają niepoliczalną ilość danych przeznaczonych do różnych celów. Analiza rynku i zachowań klientów, optymalizacja procesów biznesowych, marketing i reklama to tylko niektóre z nich. Wraz z rosnącą ilością danych, coraz trudniej jest efektywnie nimi zarządzać i wyciągać z nich użyteczne wnioski [1].

Jednym z wyzwań, który idzie wraz z nadmiarem informacji, jest problem ich efektywnej wizualizacji. Wizualizacja danych odgrywa kluczową rolę w ich analizie, ponieważ przy użyciu odpowiednich narzędzi, pozwala na ich prezentację w sposób przystępny, dzięki czemu mogą zostać zrozumiane przez szersze grono odbiorców. W ten sposób łatwiej można dostrzec związki między danymi, zrozumieć trendy i modele, a także szybciej wyciągnąć wnioski.

Kolejnym wyzwaniem są dane dynamiczne, zmienne. Wymagają one wtedy rozwiązań, które pozwalają na integracje z bazą danych w sposób ciągły. Analiza danych dynamicznych jest szczególnie istotna, ponieważ umożliwia szybką reakcję na zmieniające się warunki rynkowe, czy też biznesowe. Stałe monitorowanie daje możliwość szybkiego podejmowania decyzji w sytuacjach tego wymagających.

1.1. Cel pracy

Celem pracy jest przeanalizowanie aktualnie dostępnych narzędzi wizualizacyjnych oraz zaproponowanie rozwiązania, które pozwoli integrować dane z różnych źródeł celem wygenerowania ich formy prezentacyjnej w przestrzeni trójwymiarowej.

1.2. Rozwiązania przyjęte w pracy

Prototyp został opracowany przy użyciu frameworku Next.js, który cechuje się zintegrowanym środowiskiem zawierającym warstwę serwera, jak i klienta. Zaproponowana implementacja pracuje na środowisku uruchomieniowym Node.js, a całość jest napisana w języku TypeScript, czyli typowanej wersji języka JavaScript. Miejscem przechowywania danych jest relacyjna baza danych PostgreSQL. Zarówno część serwera, jak i klienta będzie posiadać system wtyczek, który w generyczny sposób pozwoli na dynamiczne rozszerzanie funkcjonalności docelowej aplikacji.

1.3. Rezultaty pracy

Wynikiem pracy jest analiza wymagań stawianych w dziedzinie prezentacji informacji oraz zaproponowanie prototypu, który pozwoli na integracje z wieloma różnymi źródłami danych i ich wizualizacje w dowolny sposób. Prototyp posiada zestaw następujących funkcjonalności:

- Aplikacja domyślna zawierająca logowanie, rejestracje, tworzenie wykresów oraz system osadzania wtyczek,
- Wtyczki do integracji z bazami danych:
    - PostgreSQL,
    - MySQL,
    - MongoDB.

- Wtyczki do wizualizacji danych:
    - Kostka OLAP,
    - Wykres słupkowy 3D,
    - Wykres punktowy 3D,
    - Wykres płaszczyzny 3D.

1.4. Organizacja pracy

Rozdział drugi pokazuje istniejące na rynku podobne rozwiązania wraz ze wskazaniem ich mocnych oraz słabych stron.

Trzeci rozdział opisuje wnioski wyciągnięte na podstawie rozwiązań z rozdziału drugiego. Przedstawiono również wymagania stawiane nowemu rozwiązaniu oraz propozycje rozwiązania.

W rozdziale czwartym przedstawiono spis języków, bibliotek, narzędzi oraz serwisów użytych podczas implementacji prototypu.

Rozdział piąty skupia się na projekcie architektury rozwiązania wraz z szczegółowym opisem kluczowych funkcjonalności systemu.

Rozdział szósty prezentuje możliwości, jakie oferuje stworzony prototyp.

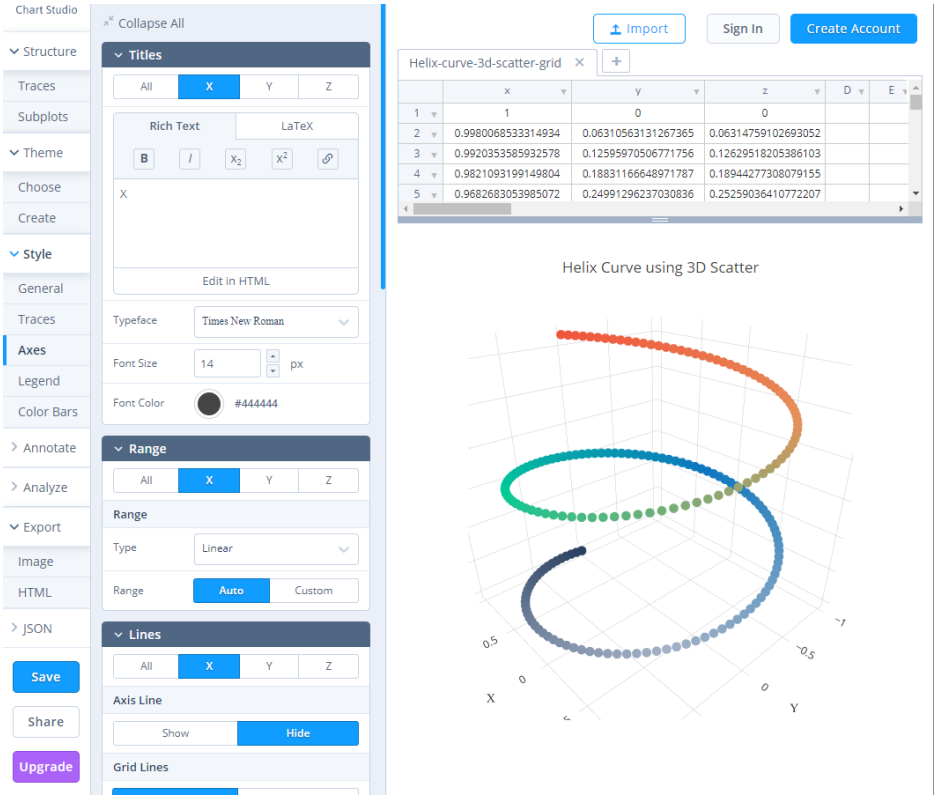
Podsumowanie pracy, listę wad i zalet oraz możliwe kierunki rozwoju rozwiązania przedstawiono w rozdziale siódmym.

2. Istniejące rozwiązania

W niniejszym rozdziale zostaną przedstawione istniejące na rynku rozwiązania, które dotyczą problemu opisanego w pracy. Zostaną pokazane ich mocne oraz słabe strony, a także poddane ocenie użyteczności w kontekście wizualizacji, jak i problemu dotyczącego integracji z wieloma źródłami danych.

2.1. Chart Studio od plotly.com

Chart Studio pozwala użytkownikom na wizualizacje danych na wykresach zarówno dwuwymiarowych, jak i trójwymiarowych. Skonfigurować wygląd wykresu można poprzez wybranie zestawu predefiniowanych styli, lub poprzez ręczne ustawienie odpowiednich pól. Poruszanie po interfejsie jest raczej intuicyjne, ale bardzo rozbudowane, przez co niekiedy ciężko dotrzeć do interesującej nas opcji. Interfejs został przedstawiony na rysunku 1.



Rysunek 1. Interfejs użytkownika Chart Studio od plotly.com. Źródło: [2]

Zalety:

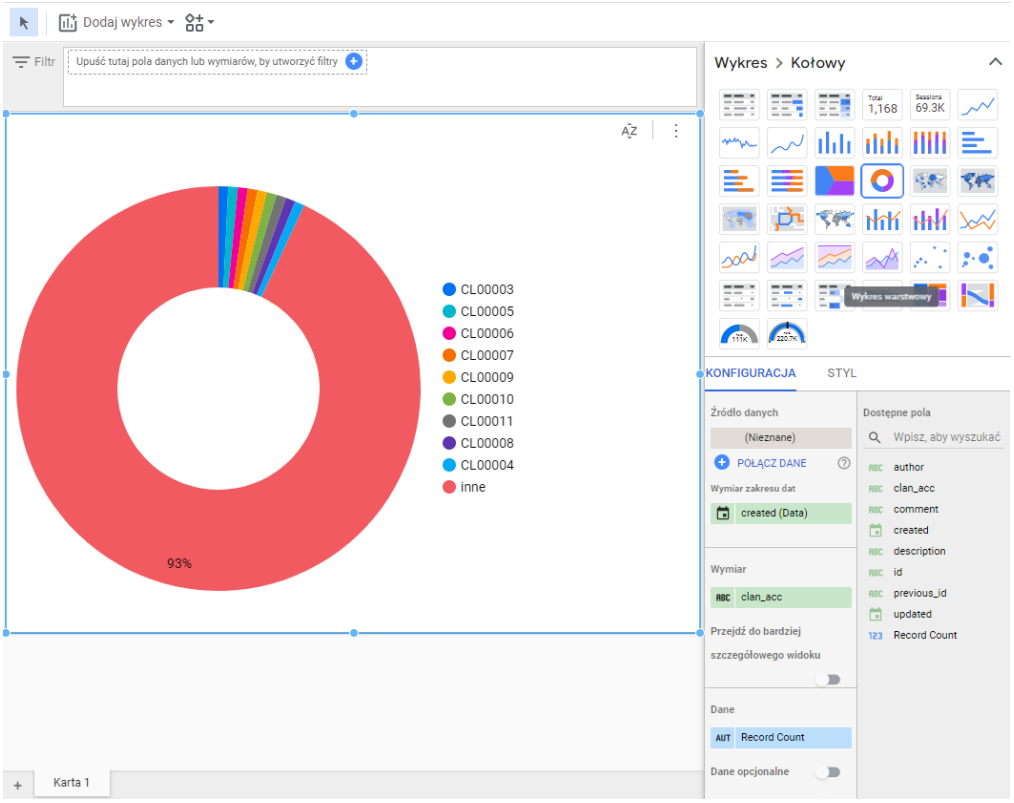
- Możliwość zapisywania projektów w serwisie,
- Szeroka gama możliwych do wyboru wizualizacji,
- Rozbudowana konfiguracja styli wykresu,
- Możliwość konfiguracji ustawień kamery.

Wady:

- Importowanie danych statycznych głównie z pliku CSV, integracja z relacyjną bazą danych możliwa tylko przez pobranie odpowiedniego narzędzia na komputer użytkownika,
- Dane z relacyjnych baz danych nie są pobierane „na żywo”, a co ustalony przedział czasowy,
- Filtrowanie danych z baz relacyjnych wymaga znajomości języka SQL – brak interfejsu, przez co rozwiązanie skierowane jest do bardzo zaawansowanych użytkowników,
- Integracja tylko z relacyjnymi bazami danych, brak możliwości integracji z rozwiązaniami nierelacyjnymi,
- Użytkownik wyświetlający wykres nie ma możliwości interakcji z danymi.

2.2. Looker Studio od Google

Looker Studio jest rozwiązaniem chmurowym do raportowania i tworzenia wykresów z różnych źródeł danych dostarczone przez korporację Google. Narzędzie jest trudne w obsłudze i w porównaniu do rozwiązania z poprzedniego podrozdziału, jest mało intuicyjne. Pozwala tylko na podstawową konfigurację wyglądu – styl linii i kolor. Większość dostępnych wykresów do użycia są wykresami dwuwymiarowymi. Interfejs został zaprezentowany na rysunku 2.



Rysunek 2. Interfejs użytkownika Looker Studio od Google. Źródło: [3]

Zalety:

- Możliwość pobierania danych z wielu źródeł, w tym z rozwiązań analitycznych od Google, np. Google Ads,
- Szeroka gama konektorów do zewnętrznych serwisów,
- Na podstawie wygenerowanego wykresu można od razu wygenerować raport,

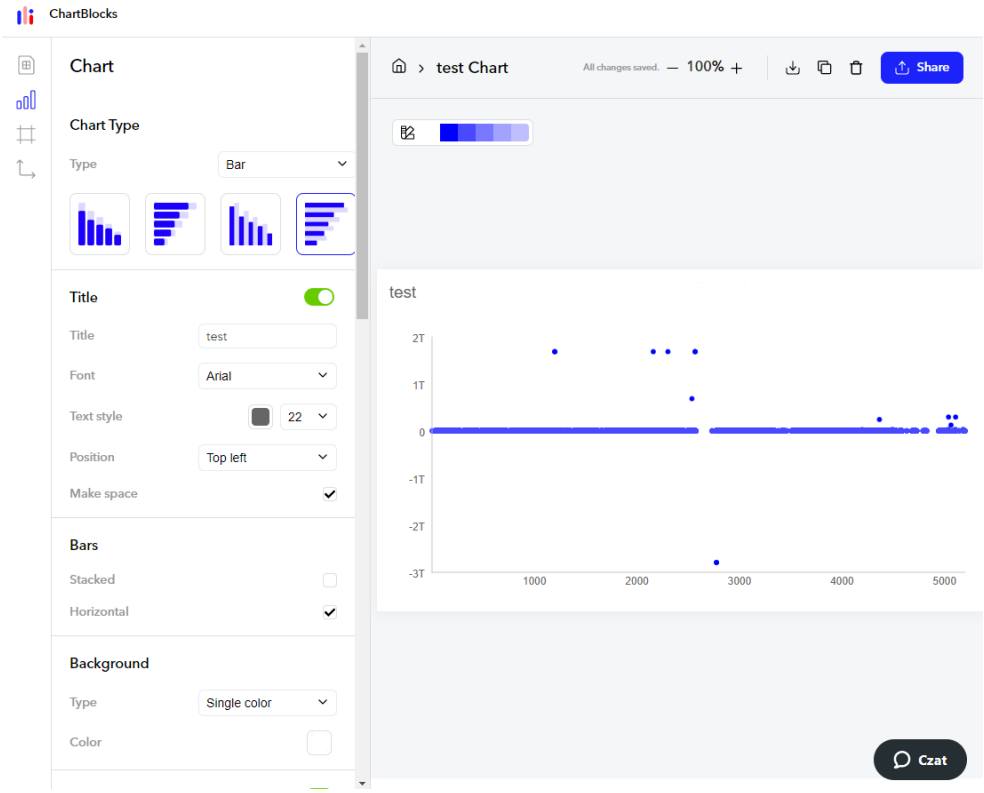
- Cały proces odbywa się w chmurze z poziomu przeglądarki.

Wady

- Nieintuicyjny interfejs, który jest trudny w obsłudze,
- Dane nie są pobierane w trybie ciągłym, a są odświeżane co skonfigurowany przedział czasowy,
- Połączenie z relacyjną bazą danych nie pozwala na przefiltrowanie wierszy oraz kolumn bez znajomości składni SQL.

2.3. ChartBlocks

Chart Blocks to kolejne rozwiązanie pozwalające na wizualizacje danych w postaci wykresów. Jest w pełni darmowe, ale oferuje tylko integracje z danymi statycznymi wpisanymi ręcznie, lub z pliku CSV. Nie pozwala ono jednak na jakiekolwiek filtrowanie zaczytanych danych. Pomimo, że interfejs, przedstawiony na rysunku 3, jest czytelny, schludny i intuicyjny, działa on wolno przy wgraniu zestawu danych, który zawiera zaledwie dwa tysiące pozycji.



Rysunek 3. Interfejs użytkownika ChartBlocks. Źródło: [4]

Zalety:

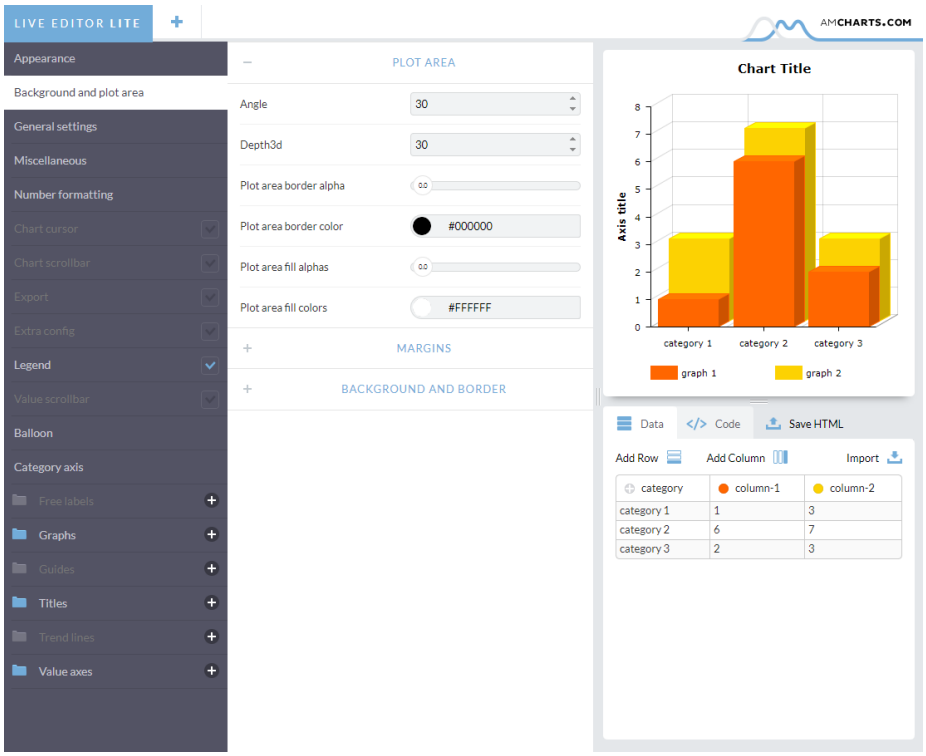
- W pełni darmowa platforma,
- Możliwość wygenerowania publicznego linku z zaprojektowanym wykresem,
- Czytelny i intuicyjny interfejs.

Wady:

- Rozwiązanie działa wolno przy większym zestawie danych,
- Dane można tylko wpisać manualnie, lub wgrać z pliku CSV,
- Brak możliwości filtrowania zaczytanych danych.

2.4. Live Editor AMCharts

Live Editor to edytor do generowania wykresów zarówno dwuwymiarowych, jak i trzywymiarowych na podstawie biblioteki AMCharts. Interfejs, widoczny na rysunku 4, jest przejrzysty i udostępnia szereg opcji, które można skonfigurować. Narzędzie jednak przez dużą ilość opcji do konfiguracji wydaje się być złożone i trudniejsze w obsłudze. Dodatkowo, nie daje ono możliwości zaczytania danych z innych źródeł niż z pliku CSV.



Rysunek 4. Interfejs użytkownika Live Editor AMCharts. Źródło: [5]

Zalety:

- Brak potrzeby zakładania konta,
- Duża ilość opcji do konfiguracji,
- Schludny i czytelny interfejs.

Wady:

- Brak możliwość zapisu projektu,
- Importowanie danych jedynie z pliku CSV,
- Brak możliwości filtrowania danych,

- Eksport wykresu możliwy tylko jako kod HTML do osadzenia na stronie.

2.5. Podsumowanie

Każdy z wymienionych ogólnodostępnych narzędzi oferuje jakąś unikalną funkcjonalność, która na pewno jest dużą zaletą danego rozwiązania. Dla niektórych jest to dopracowany interfejs, a dla innych wysoka konfigurowalność.

Wiele z podanych narzędzi nie pozwala jednak na szeroko pojętą integrację z bazami danych, a jak już pozwala, to ma ograniczone filtrowanie danych, przez co wymaga, aby dane były już gotowe u źródła. Jakakolwiek interakcja z danymi, jeśli jest możliwa, wymaga umiejętności posługiwania się językiem SQL.

Najważniejsze cechy na jakie możemy zwrócić uwagę przy przedstawionych rozwiązaniach to możliwość:

- tworzenia wykresów 3D,
- integracji z bazami danych,
- filtrowanie danych,
- filtrowania danych z bazy danych bez znajomości SQL,
- zapisu projektu w serwisie.

Wyżej wymienione cechy zostały zestawione względem wymienionych wcześniej narzędzi w tabeli 1.

Tabela 1. Zestawienie najważniejszych cech wymienionych narzędzi. Źródło: opracowanie własne.

	Chart Studio	Looker Studio	Chart Blocks	Live Editor AM
Wykresy 3D	Tak	Tak	Nie	Tak
Integracja z bazami danych	Tak (przez zew. program)	Tak	Nie	Nie
Filtrowanie danych	Tak	Tak	Nie	Nie
Filtrowanie danych z bazy bez SQL	Nie	Nie	-	-
Zapis projektu w serwisie	Tak	Tak	Nie	Nie

Podsumowując zestawienie z tabeli 1, żadne rozwiązanie nie pozwala na filtrowanie dynamicznych danych bez znajomości składni języka SQL. Z kolei narzędzia, które pozwalają na integrację z bazami danych, umożliwiają tylko integracje z tymi, które są opracowane w technologii relacyjnej, co oznacza, że niemożliwa jest integracja baz nierelacyjnych takich jak np. MongoDB.

W żadnym rozwiązaniu nie ma możliwości wystawienia konfiguracji, dzięki której użytkownik analizujący stworzony wykres, mógłby przefiltrować dane źródłowe, np. przy użyciu suwaków, czy predefiniowanych przycisków.

3. Propozycja rozwiązania

Rozdział prezentuje wnioski, jakie zostały wyciągnięte z analizy istniejących już narzędzi. Na ich podstawie opisane zostały wymagania, jakim nowe rozwiązanie powinno sprostać. Wymagania stanowią funkcjonalności, które docelowo powinien posiadać system do wizualizacji wielowymiarowych danych.

3.1. Wnioski po analizie istniejących rozwiązań

Po zsumowaniu mocnych i słabych stron istniejących narzędzi, można wyszczególnić listę funkcjonalności, które są pożądane w takiego rodzaju rozwiązaniach:

- Pełnia funkcjonalności działających bezpośrednio w przeglądarce internetowej bez konieczności instalowania programów na komputerze użytkownika,
- Możliwość generowania wykresów trójwymiarowych,
- Możliwość importowania danych z zarówno baz danych relacyjnych, jak i nierelacyjnych w czasie rzeczywistym,
- Filtrowanie zintegrowanych danych bez konieczności znajomości składni SQL,
- Filtrowanie danych z poziomu analizy gotowego, ówcześnie skonfigurowanego wykresu,
- Skalowalność – możliwość w ustandaryzowany sposób dołączania kolejnych metod integracji ze źródłami danych oraz wizualizacji,
- Możliwość zapisu skonfigurowanego projektu bezpośrednio w serwisie,
- Udostępnianie wykresu przy użyciu publicznego adresu URL.

3.2. Wymagania

Podrozdział definiuje wymagania, które stawiane są przed tworzonym prototypem. Każde z wymagań powstało na bazie najlepszych cech pozyskanych z analizy istniejących na rynku narzędzi wymienionych w poprzednim podrozdziale.

3.2.1 Filtrowanie danych

Filtrowanie danych jest kluczowym elementem integracji zewnętrznego źródła danych celem generowania wykresu. Brak tej funkcjonalności wymagałby od użytkownika wcześniejszego przygotowania danych i umieszczenia ich w osobnym miejscu przeznaczonym tylko dla konkretnego projektu. Można więc zauważyć, że taka sytuacja prowadzi do niechcianej redundancji danych.

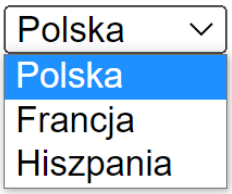
Brak znajomości składni SQL nie powinien dyskwalifikować użytkownika systemu od używania jego głównych funkcjonalności. Aplikacja zawierająca interfejs do nakładania kolejnych filtrów na dane źródłowe, pozwala na uproszczenie procesu generowania pożądanych wizualizacji. Generuje to jednak dodatkową trudność przy tłumaczeniu zaprojektowanych filtrów na różne rozwiązania bazodanowe.

Nie tylko filtrowanie danych podczas konfigurowania wizualizacji ułatwia ich późniejszą analizę, ale też możliwość filtrowania ich już z poziomu wyświetlanego wykresu. Takim sposobem, nie będzie wymagane tworzenie nowej wizualizacji dla każdego zmiennego parametru.

Rozważmy przykład, gdzie wizualizacja przedstawia dwuwymiarowy wykres słupkowy ilości sprzedaży względem lat. Przykładowy wykres przedstawiono na rysunku 6. W przypadku, gdy potrzebna jest bardziej szczegółowa analiza sprzedaży dla każdego kraju, domyślnie, konieczne byłoby wygenerowanie nowego wykresu dla każdego z nich.

Rysunek 5. Przykładowy wykres ilości sprzedaży względem lat. Źródło: opracowanie własne.

Jeżeli do wykresu dodana byłaby lista wyboru z dodatkowym filtrem, jakim jest nazwa kraju (rysunek 6), to po wyborze jednego z nich, dane mogłyby zostać w dynamiczny sposób ponownie przefiltrowane, a na ich podstawie wykres na nowo zostać wygenerowany. Takie rozwiązanie nie tylko usprawnia interakcje z wykresem, ale także eliminuje potrzebę tworzenia kolejnych wykresów dla każdego z krajów.



Rysunek 6. Przykładowa lista wyboru kraju, jako dodatkowy filtr. Źródło: opracowanie własne.

3.2.2 Integracja z różnymi źródłami danych

Przez dużą ilość rozwiązań przechowywania danych dostępnych na rynku, wymagana jest od aplikacji możliwość ustandaryzowanego dodawania kolejnych integracji ze źródłami danych. Docelowo, każda integracja powinna być wtyczką, która w sposób prosty może zostać dołączona do aplikacji przy jej uruchomieniu.

Rozpatrywanie integracji jako wtyczki jest spowodowane tym, że każda baza danych, czy inne rozwiązania do przechowywania informacji cechuje się różnymi sposobami łączenia. Dodatkowo, eliminuje to złożoność, która jest związana z integrowaniem do systemu coraz to nowych źródeł danych.

3.2.3 Wizualizacja danych

Analiza, czy prezentacja danych jest efektywniejsza przy użyciu różnych wizualizacji, które sprawiają, że zebrane dane stają się łatwiejsze w rozumieniu.

Dane mogą zostać zwizualizowane na wiele sposobów – wykresy punktowe, mapy cieplne, płaszczyzny itd. Dlatego wdrożenie każdej z osobna do jednej aplikacji coraz bardziej zwiększa jej kompleksowość. Podobnie jak w punkcie 3.2.3, istotne jest rozpatrzenie każdej z wizualizacji jako generycznej wtyczki osadzonej w systemie.

Przydatną funkcjonalnością jest również nakładanie punktów charakterystycznych na wykres. Przykładem takiego rozwiązania może być pokolorowanie słupków z rysunku 5 na czerwono, jeśli sprzedaż wyniosła mniej niż 60. W ten sposób, zwrócenie uwagi odbiorcy na istotne miejsca wykresu staje się prostsze, a sam wykres czytelniejszy.

3.2.4 Zapis projektu w systemie

Możliwość zapisu skonfigurowanego źródła danych, jak i wizualizacji bezpośrednio w systemie, bez potrzeby zapisu na urządzenie użytkownika, przynosi wiele zalet. Główną z nich jest możliwość rekonfiguracji projektu z każdego urządzenia z dostępem do Internetu. Nie wymusza to na użytkowniku przechowywania plików projektu, a każda stworzona wizualizacja jest przechowywana bezpośrednio w systemie i przypisana do jednego konta użytkownika.

3.2.5 Udostępnianie skonfigurowanej wizualizacji

Wymaganie udostępnienia wizualizacji jest uwarunkowane potrzebą jej wystawienia dla szerszego grona odbiorców. Przykładem może być sytuacja, gdzie wizualizacja jest częścią prezentacji marketingowej lub analitycznej, dlatego jej wyświetlenie nie powinno wymagać logowania się na konto użytkownika, które jest do niej przypisane.

4. Zastosowane technologie

Niniejszy rozdział opisuje technologie, przy pomocy których został opracowany prototyp rozwiązania.

4.1. HTML

HTML, czyli hipertekstowy język znaczników (ang. HyperText Markup Language), wykorzystywany jest do tworzenia stron internetowych. Język ten pozwala na strukturyzację i formatowanie zawartości dokumentu, takiego jak tekst, obrazy, multimedia i inne elementy interaktywne, aby umożliwić ich wyświetlanie w przeglądarce internetowej.

Autorem tej technologii jest Tim Berners Lee. W latach 90 opublikował pierwszą dokumentacje opisującą podstawowe funkcjonalności języka, a samo rozwiązanie zostało oparte na języku SGML (ang. Standard Generalized Markup Language). [6]

Elementy składające się na język HTML to zestaw specjalnych znaczników, które definiują typy zawartości, takie jak nagłówki, paragrafy, listy, tabele i wiele innych. Dodatkowo, do każdego ze znaczników można przypisać atrybuty, które umożliwiają określenie właściwości poszczególnych elementów.

Listing 1. Przykładowy akapit opisany w HTML. Źródło: opracowanie własne.

```
<p class="important-paragraph" contenteditable spellcheck="true">
    Zawartość akapitu
</p>
```

Listing 1 pokazuje sposób przykładowego użycia języka HTML w celu stworzenia akapitu. Budowa tego elementu po kolei składa sie na:

- Otwarcie bloku * *,
- Zestaw atrybutów opisujących własności akapitu:
    - *class* – opisuje przynależność elementu do grupy *important-paragraph*, która później może służyć do nadanie konkretnych cech wizualnych takich jak np. rozmiar czcionki, kolor tła itp.,
    - *contenteditable* – atrybut, który definiuje przeglądarce, że zawartość elementu może być zmieniona przez użytkownika,
    - *spellcheck* – uruchamia sprawdzanie pisowni na zawartości wpisywanej przez użytkownika.
- Zawartość,
- Zamknięcie bloku * *.

Ważną cechą HTML jest niezależność od platformy, na której jest interpretowany. Może być wykorzystywany na różnych urządzeniach i nie tylko w przeglądarkach internetowych. Język ten jest stale rozwijany przez organizację World Wide Web Consortium (W3C), która tworzy standardy dla technologii internetowych.

4.2. CSS

CSS (Cascading Style Sheets) to język służący do opisu stylów wyglądu i prezentacji elementów stron internetowych. Umożliwia oddzielenie zawartości strony od jej prezentacji, dzięki czemu tworzenie i utrzymanie stron internetowych staje się prostsze i bardziej elastyczne.

Język został wprowadzony w 1996 roku przez W3C i stał się kluczowym elementem projektowania stron internetowych [7]. Głównym jego celem jest umożliwienie projektantom i programistom tworzenie spójnego wyglądu stron internetowych, niezależnie od urządzenia na którym są przeglądane.

Słowo „kaskadowość" wchodzące w skład rozwinięcia skrótu CSS oznacza, że style mogą być definiowane na różnych poziomach struktury dokumentu i nakładane jeden na drugi, tworząc kaskadę. Na przykład, styl definiowany bezpośrednio w elemencie HTML w atrybucie *style* ma pierwszeństwo przed stylem zdefiniowanym w oddzielnym pliku. Przykład użycia CSS przedstawiono w listingu 2.

Listing 2. Przykładowe użycie języka CSS. Źródło: opracowanie własne.

```
.important-paragraph {
    font-size: 20px;
    background-color: gray;
}
```

Budowa bloku pokazanego w listingu 2 zaczyna się od nazwy grupy, której nadajemy własności wizualne (*important-paragraph*). Przedrostek, jakim w tym przypadku jest kropka przed nazwą grupy, wyszukuje elementy w HTML, które posiadają tę nazwę wpisaną w atrybut *class*. Wewnątrz klamer znajdują się nazwy własności, wraz z ich wartościami:

- *font-size* wskazuje, aby wielkość czcionki wyniosła *20px*
- *background-color* koloruje tło na kolor szary (*gray*)

4.3. Tailwind CSS

Tailwind CSS to framework CSS, który został opracowany przez Adama Wahtana, a jego pierwsza odsłona miała miejsce w 2017 roku [8]. Jego popularność wzrosła w ciągu ostatnich lat, a dzisiaj jest to jedno z najczęściej stosowanych narzędzi do projektowania stron internetowych.

Framework opiera się na filozofii projektowania bez interfejsu użytkownika, co oznacza, ze projektant ma większą kontrolę na kodem i stylami, a nie musi się przejmować takimi aspektami jak przyciski czy formularze. Zamiast tego, Tailwind CSS ofertuje zestaw klas CSS, które można zastosować do dowolnego elementu HTML, aby dostosować jego wygląd i styl.

Zestaw dostępnych klas w narzędziu jest podzielony na różne kategorie, takie jak marginesy, kolory, czy czcionki [7]. Dzięki temu projektanci mogą szybko i łatwo dostosować wygląd strony internetowej bez potrzeby pisania zewnętrznych styli CSS.

Przykładowo, użycie nazwy klasy *text-red-500* w atrybucie *class* w HTML, spowoduje pokolorowanie tekstu na kolor czerwony w odcieniu 500 za sprawą opisywanego frameworku, który odnajduje nazwy klas i automatycznie generuje z nich style CSS.

Jednym z największych atutów tego rozwiązania jest wydajność. Dzięki zastosowaniu technik optymalizacji CSS, jak m.in. skracanie i eliminacje nieużywanych stylów, pliki arkusza styli są mniejsze, przez co strony internetowe ładują się szybciej.

4.4. JavaScript

JavaScript jest językiem programowania, który opracował Brendan Eich w 1995 roku [9]. Początkowo był używany do prostych interakcji ze stronami internetowymi, jednak dziś jest on jednym z najpopularniejszych języków programowania na świecie. Jego użycie nie ogranicza się tylko do przeglądarki i uruchamiana po stronie klienta. Swoją funkcję spełnia również po stronie serwera.

Jako język programowania, JavaScript posiada wiele cech, które czynią go popularnych i wszechstronnym. Jest prosty w użyciu i nauce, a także łatwo dostępny dla początkujących programistów, ponieważ nie potrzebuje on rozległego środowiska uruchomieniowego. Sam język pozwala na stosowanie wielu różnych paradygmatów programowania, w tym programowania proceduralnego, obiektowego i funkcjonalnego. Język jest typowany dynamicznie, co jest główną różnicą względem innych popularnych języków programowania. Oznacza to, że typy zmiennych są określane w czasie wykonywania kodu, a nie na etapie jego kompilacji.

Wraz z rozwojem Internetu i postępem technologicznym, JavaScript stał się bardziej powszechny zaczął być wykorzystywany do tworzenia różnych typów aplikacji. Za sprawą pojawienia się wielu bibliotek i frameworków takich jak jQuery, Angular, czy React [10, 11, 12], stał się najpopularniejszym językiem technologii webowych, który pozwala tworzyć zaawansowane aplikacje mogące działać tylko w przeglądarce, bez potrzeby instalowania zewnętrznych programów komputerowych.

4.5. TypeScript

TypeScript [13] to język programowania stworzony przez Microsoft, który został zaprojektowany jako rozszerzenie języka JavaScript.

Język w porównaniu do JavaScript, bazuje na silnym typowaniu, co oznacza, że każda wartość ma określony typ i jest sprawdzana podczas kompilacji. Dzięki temu, błędy typowania są wykrywane na etapie pisania kodu, co umożliwia szybsze debugowanie i zmniejsza ilość błędów na etapie uruchamiania aplikacji. Typowanie umożliwia także łatwiejsze zarządzanie kodem i ułatwia pracę z większymi projektami.

TypeScript jest w pełni kompatybilny z językiem JavaScript, co umożliwia programistom łatwe przejście z jednego języka na drugi. Umożliwia również korzystanie z istniejących bibliotek JavaScript oraz narzędzi z ekosystemu JavaScript.

4.6. React.js

React.js [12] to biblioteka JavaScript wykorzystywana do tworzenia funkcjonalnych interfejsów użytkownika na stronach i aplikacjach internetowych.

Jako biblioteka, zapewnia programistom łatwość tworzenia interaktywnych interfejsów, które są łatwe w użyciu i utrzymaniu. React opiera się na deklaratywnym podejściu, co oznacza, że to

programista określa, jak powinien wyglądać widok, a biblioteka zajmuje się jego wyświetlaniem i aktualizowaniem.

React.js umożliwia tworzenie aplikacji, które są łatwe w utrzymaniu i rozbudowie. Biblioteka ta posiada wiele narzędzi, które umożliwiają programistom tworzenie aplikacji internetowych, w tym wirtualnego DOM-u (Document Object Model), który umożliwia efektywną aktualizację interfejsu użytkownika. React.js wykorzystuje również JSX [14], który pozwala na pisanie kodu JavaScript w połączeniu z elementami HTML, co ułatwia tworzenie aplikacji internetowych.

Od czasu powstania w 2011 roku, React.js przeszedł wiele zmian i ewolucji. React.js stał się coraz bardziej popularny wśród programistów, a także zyskał poparcie wielu firm, takich jak Airbnb, Netflix, Instagram i wiele innych [15].

4.7. React Query Builder

React Query Builder [16] to biblioteka open source opracowana w języku JavaScript jako moduł do Reacta. Umożliwia tworzenie interfejsów użytkownika, które są przeznaczone do budowania złożonych zapytań do baz danych.

Funkcjonalności wchodzące w skład tego narzędzia, to m.in. operatory logiczne, wybór kolumn, zapisu i ładowania zapytań, a także walidacji, co zapobiega wprowadzaniu błędnych danych. Zaprojektowane zapytanie przez użytkownika może zostać przekonwertowane na zapytanie SQL, a także na format zapytań do baz nierelacyjnych takich jak na przykład MongoDB.

4.8. Three.js

Three.js [17] to biblioteka open source oparta na technologii WebGL służąca do tworzenia grafiki trójwymiarowej w aplikacjach internetowych. Zapewnia funkcjonalności takie jak tworzenie obiektów 3D, animacje, oświetlenie, nakładanie tekstur, a także integrację z różnymi narzędziami i platformami. Biblioteka oferuje niski próg wejścia w stworzenie elementów trójwymiarowych możliwych do wyświetlenia w przeglądarce, za sprawą dużej ilości funkcjonalności i bogatej dokumentacji.

WebGL to specyfikacja OpenGL dostępna w przeglądarkach internetowych. Ta technologia pozwala na tworzenie aplikacji 3D w przeglądarce bez konieczności korzystania z wtyczek i innych dodatkowych narzędzi.

4.9. Vis.js

Biblioteka Vis.js [18] umożliwia tworzenie interaktywnych wizualizacji w przeglądarce internetowej. Jest to narzędzie dostępne jako open source, które udostępnia użytkownikom metody, dzięki którym zaledwie kilkoma linijkami kodu można stworzyć interaktywną prezentacje danych. Możliwe do stworzenia wizualizacje to między innymi:

- grafy,
- osie czasu,
- wykresy dwuwymiarowe,
- wykresy trójwymiarowe.

4.10. Lexpjs

Lexpjs [19] to lekki interpreter wyrażeń matematycznych napisany w języku JavaScript i udostępniony jako biblioteka open source.

Obsługuje proste wyrażenia matematyczne dla dodawania, odejmowania, mnożenia, dzielenia, operacji modulo, bitowych oraz logicznych. Posiada również zestaw wbudowanych funkcji matematycznych takich jak *abs, cos, sin*, itd. Składna lexpjs jest podobna do samego języka JavaScript.

4.11. Node.js

Node.js [20] to popularna platforma programistyczna działająca po stronie serwera, która umożliwia tworzenie aplikacji internetowych z wykorzystaniem języka JavaScript. Została opracowana w 2009 roku i jest oparta na silniku V8, który jest stosowany w przeglądarce internetowej Google Chrome [21].

Jedną z najważniejszą funkcjonalności Node.js jest asynchroniczność. Platforma umożliwia wykonywanie wielu operacji równocześnie, co znacznie przyspiesza przetwarzanie danych. Mechanizm asynchroniczny jest możliwy dzięki zastosowaniu tzw. wywołań zwrotnych (ang. callbacks) i obietnic (ang. promises), które umożliwiają wykonywanie operacji w tle i kontynuowanie pracy z innymi. Asynchroniczność jest szczególnie ważna w kontekście aplikacji internetowych, gdzie wymagane jest przetwarzanie wielu operacji jednocześnie.

4.12.Next.js

Next.js [22] to framework stworzony dla języka JavaScript, który umożliwia łatwe i szybkie tworzenie zaawansowanych aplikacji internetowych. Jest on oparty na platformie Node.js i działa zarówno po stronie klienta jak i serwera. Wprowadza wiele udogodnień, takich jak automatyczne renderowanie po stronie serwera, dynamiczne ładowanie danych oraz możliwość tworzenia statycznych stron internetowych. Został opracowany przez firmę Vercel i oparty jest na bibliotece React.

Renderowanie po stronie serwera oznacza, że strona jest generowana na serwerze, zanim zostanie przesłana do przeglądarki klienta. Dzięki temu czas ładowania strony jest znacznie krótszy, co poprawia doświadczenie użytkownika.

4.13.MySQL

MySQL [23] to popularny system zarządzania relacyjnymi bazami danych, który umożliwia łatwe i skuteczne zarządzanie danymi. Opiera się na języku zapytań SQL (Structured Query Language), który umożliwia łatwe dodawanie, usuwanie i modyfikowanie danych w bazie danych. MySQL obsługuje wiele różnych typów danych, w tym liczby całkowite i zmiennoprzecinkowe, daty i czasy, ciągi znaków i wiele innych.

MySQL jest platformą skalowalną, co oznacza, że może obsługiwać duże ilości danych i użytkowników bez obciążania serwera. MySQL umożliwia także replikację danych, co zapewnia wysoką dostępność i niezawodność.

4.14. PostgreSQL

PostgreSQL [24] to zaawansowany system typu open source do zarządzania relacyjnymi bazami danych. Jest darmowym oprogramowaniem i jest szeroko stosowany w różnych aplikacjach, szczególnie w tych wymagających dużej wydajności i niezawodności. Podobnie jak MySQL, opiera się na SQL, jednak rozszerza ten język o dodatkowe funkcje pozwalające na bardziej zaawansowane operacje na danych.

Zapewnia wysoki poziom bezpieczeństwa danych, co jest szczególnie ważne w przypadku aplikacji, które przechowują wrażliwe dane. Umożliwia również uwierzytelnianie użytkowników i kontrolę dostępu do danych. Możliwe jest także szyfrowanie danych przechowywanych w bazie danych.

4.15. MongoDB

MongoDB [25] to system zarządzania bazami danych NoSQL. Umożliwia przechowywanie i przetwarzanie danych w formacie dokumentów, co oznacza, że dane są przechowywane w postaci JSON [26], a nie w tabelach i wierszach, jak w przypadku baz relacyjnych.

Baza danych MongoDB składa się z kolekcji dokumentów, które przechowują dane w formacie BSON (Binary JSON). Dokumenty nie muszą mieć ściśle określonej struktury, co umożliwia przechowywanie różnych typów danych w jednej kolekcji, a powiązania między sobą dokonuje się za pomocą kluczy obcych lub przez zagnieżdżanie dokumentów w innych dokumentach.

Wykorzystuje język zapytań MongoDB Query Language (MQL), który umożliwia tworzenie zapytań i wyszukiwanie danych w bazie danych MongoDB. MQL umożliwia również zaawansowane operacje na danych, takie jak sortowanie, grupowanie i łączenie dokumentów.

4.16. Prisma.js

Prisma.js [27] to narzędzie ORM, czyli sposób zarządzania relacyjnymi bazami danych, które umożliwia pracę na bazie danych za pomocą JavaScript, czy też TypeScript. Jest pośrednikiem między aplikacją, a bazą danych dostarczającym API oraz narzędzia do tworzenia, odczytu, aktualizacji i usuwania danych.

Pozwala na zamodelowane danych za pomocą języka GraphQL [28], co umożliwia precyzyjne określenie struktury danych oraz relacji między nimi. Na podstawie stworzonej struktury, Prisma.js automatycznie generuje kod, który pozwala na sprawniejsze operacje na bazie danych poprzez dedykowane API zawierające metody domyślnie zintegrowane ze stworzonym schematem.

4.17. Lerna.js

Lerna.js [29], czyli narzędzie open source, które ułatwia zarządzanie projektami oprogramowania rozwijanych w formie monorepozytorium. Monorepozytorium to praktyka, w której cały kod źródłowy przechowywany jest w jednym miejscu. Eliminuje to potrzebę dzielenia oprogramowania na wiele mniejszych repozytoriów.

Umożliwia podział projektu na mniejsze, niezależne od siebie pakiety. Każdy pakiet może mieć swoje zależności, skrypty budowania, a także być publikowany jako osobny moduł. Przykładem takiego rozwiązania mogą być moduły, które w tej pracy posłużą jako wtyczki umożliwiające różne wizualizacje oraz integracje ze źródłami danych.

Narzędzie automatycznie zarządza zależnościami między pakietami. Dzięki temu można łatwo aktualizować i synchronizować zależności w całym projekcie. Oprogramowanie staje się łatwiejsze w utrzymaniu i rozwijaniu.

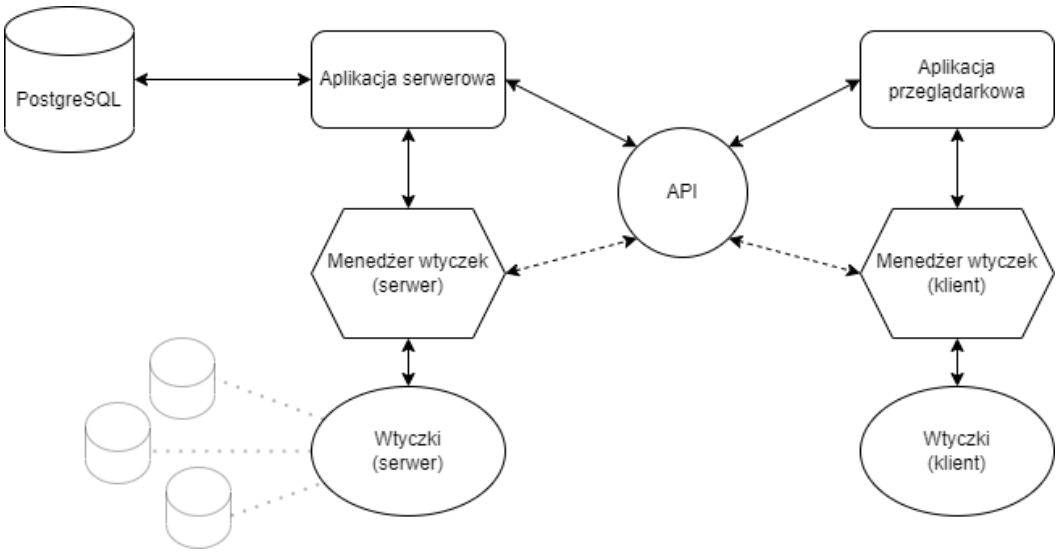
5. Projekt i implementacja

Rozdział opisuje sposób w jaki proponowane rozwiązanie zostało zaprojektowane oraz zaimplementowane. Niniejszy opis skupia się szczegółowo na złożonej architekturze oprogramowania, która umożliwia jego rozszerzanie o dodatkowe wtyczki działające zarówno po stronie serwera, jak i klienta.

5.1. Architektura rozwiązania

Na architekturę rozwiązania składają się trzy kluczowe elementy – baza danych, serwer oraz aplikacja przeglądarkowa. Zestawienie tych trzech składników pozwala użytkownikowi na podstawowe funkcjonalności – stworzenie konta, zalogowanie się oraz stworzenie projektu wizualizacji. Serwer oraz aplikacja webowa mają za zadanie udostępnić miejsca, w które można zintegrować wtyczki, które będą miały za zadanie oferować dynamiczne sposoby konfiguracji wyżej wspomnianego projektu.

Oprogramowanie przewiduje dwa rodzaje wtyczek – wizualizacyjna oraz integrująca zewnętrzne dane. Moduły wizualizacyjne nie potrzebują warstwy serwerowej, ponieważ całość kodu może korzystać z zasobów przeglądarki użytkownika. Z kolei moduły integrujące zewnętrzne dane wymagają dostępu do obu warstw architektury. W warstwie klienta użytkownik wprowadzi konfiguracje potrzebne dla danej integracji, a przy ich użyciu warstwa serwerowa połączy się z wybranym źródłem i pobierze dane.

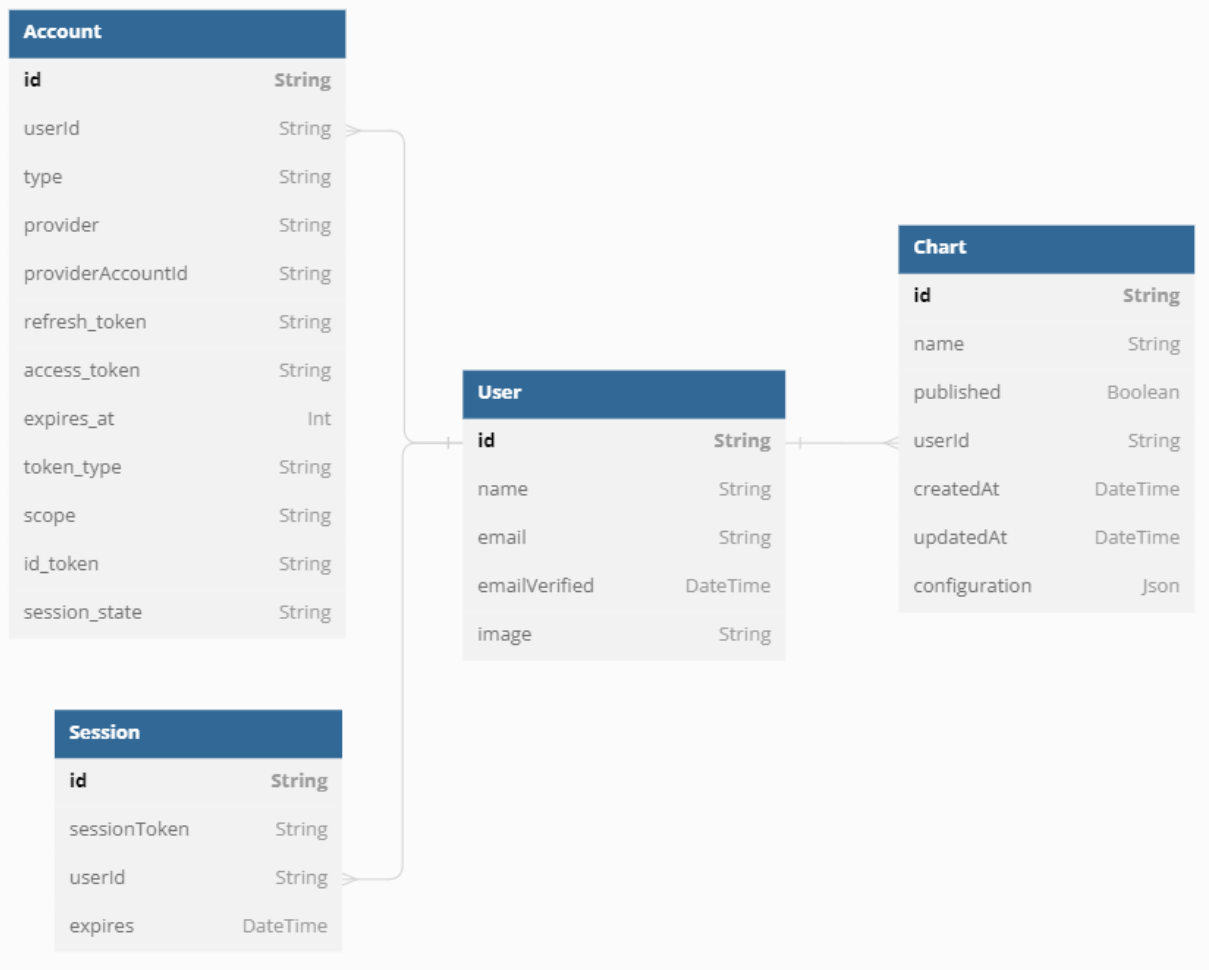


Rysunek 7. Schemat architektury rozwiązania. Źródło: opracowanie własne

Schemat widoczny na rysunku 7 ilustruje wszystkie elementy, pokazuje relacje oraz sposób komunikacji pomiędzy nimi. Całość oprogramowania jest opracowana przy użyciu frameworku Next.js uruchomionym w środowisku Node.js. Narzędziem odpowiedzialnym za rejestracje wtyczek w oprogramowaniu jest menedżer wtyczek (ang. plugin manager), który działa w obu warstwach architektury.

5.2. Baza danych

Miejscem przechowywania danych aplikacji, takich jak zarejestrowani użytkownicy oraz projekty wizualizacyjne, jest relacyjna baza danych PostgreSQL. Schemat bazy danych jest widoczny na rysunku 8.



Rysunek 8. Schemat bazy danych. Źródło: [30]

Tabele *Account*, *Session*, oraz *User* odpowiadają za przechowywanie danych o użytkowniku oraz danych o istniejącej sesji, dzięki której użytkownik otrzymuje dostęp do funkcjonalności oprogramowania. Przewidziano, że użytkownik będzie mógł się logować za pomocą kont z wielu serwisów takich jak Microsoft, GitHub itd., jednak na potrzeby prototypu zaimplementowano tylko logowanie za pomocą konta Google.

Tabela *Chart* przechowuje informacje o stworzonych projektach wizualizacji takie jak:

- *id* – unikalny identyfikator projektu,
- *name* – nazwa projektu,
- *published* – wartość logiczna mówiąca o tym, czy projekt został udostępniony i możliwy do wyświetlania bez konieczności logowania,
- *userId* – identyfikator użytkownika, do którego należy projekt,
- *createdAt* – data i czas stworzenia projektu,
- *updatedAt* – data i czas naniesienia jakichkolwiek zmian na projekt,

- *configuration* – dynamiczna struktura JSON przechowujące konfiguracje wtyczek użytych w danej wizualizacji.

Za operację na bazie danych odpowiada narzędzie Prisma.js, które zostało opisane w podrozdziale 4.16. Całość schematu została skonfigurowana z poziomu kodu oprogramowania w składni GraphQL. Konfigurację dla tabeli *Chart* przedstawiono w listingu 3. W ten sam sposób zdefiniowano resztę tabel z rysunku 8, a całość umieszczono w pliku o rozszerzeniu *.prisma*.

Listing 3. Konfiguracja schematu dla tabeli Chart. Źródło: opracowanie własne

```
model Chart {
  id            String   @id @default(cuid())
  name          String
  published     Boolean  @default(false)
  userId        String
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
  configuration Json?    @default("{}")
  user          User     @relation(fields: [userId], references:
[id], onDelete: Cascade)
}
```

Konfiguracja połączenia do bazy danych również odbywa się w pliku ze schematem. Definicje połączenia wyróżniono w listingu 4. Dane do połączenia pozyskiwane są ze zmiennej środowiskowej *DATABASE_URL*.

Listing 4. Konfiguracja połączenia do bazy danych. Źródło: opracowanie własne

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
```

Po stworzeniu odpowiedników z listingu 3 dla wszystkich pożądanych tabel oraz skonfigurowaniu połączenia wgrano opisaną strukturę pośrednio przez wiersz poleceń *prisma* do bazy danych PostgreSQL. Następnie, wygenerowano API do komunikacji z bazą danych przy użyciu modeli ze schematu kolejnym poleceniem. Obie komendy zostały kolejno wyszczególnione w listingu 5.

Listing 5. Komendy wiersza poleceń *prisma*. Źródło: opracowanie własne

```
prisma db push
prisma generate
```

Operacje na tabeli *Chart* przy pomocy wygenerowanego API odbywają się już w pliku TypeScript, gdzie po zainicjowaniu klasy *PrismaClient* możliwe jest wywoływanie zbioru metod dla danej tabeli. Dla przykładu, listing 6 pokazuje pobieranie wszystkich wpisów z tabeli *Chart*, które później wyświetlane zostaną w konsoli. Kod należy uruchamiać po stronie serwera aplikacji.

Listing 6. Przykład wyświetlenia wpisów z tabeli *Chart* w konsoli. Źródło: opracowanie własne

```
import { PrismaClient } from "@prisma/client";

const prismaClient = new PrismaClient();
const chartsPromise = prismaClient.chart.findMany();
chartsPromise.then(console.log);
```

5.3. Wtyczki

Jak opisano w podrozdziale 5.1. przewidziano dwa rodzaje wtyczek – wizualizacyjne i integrujące zewnętrzne zbiory danych. Rozszerzenia wizualizacyjne działają w warstwie klienta – przeglądarki użytkownika, z kolei rozszerzenia bazodanowe również w warstwie serwerowej. W poniższych punktach opisano przeznaczenie każdego typu wtyczki. Sposób ich tworzenia i rejestracji został przedstawiony w następnym podrozdziale 5.4.

5.3.1 Wtyczka wizualizacyjna

Wtyczka wizualizacyjna może zostać wyświetlona w dwóch miejscach w systemie. Pierwszym z nich jest przestrzeń, gdzie rozszerzenie wyświetla formularz potrzebny do skonfigurowania wizualizacji w postaci komponentu React. W drugim miejscu moduł wyświetla ówcześnie skonfigurowaną, interaktywną wizualizacje jako element HTML.

Rozszerzenie może korzystać z udostępnionego zbioru komponentów React, które wspomagają stworzenie formularza konfiguracji oraz normalizują jego wygląd, aby uniknąć rozbieżności na przestrzeni innych wtyczek.

Aby umożliwić wtyczce zapisywanie konfiguracji oraz korzystanie z poprzednio zapisanej, komponent konfiguracji React jest osadzony przez główną aplikację w *React Context*, który udostępnia metody zapisu oraz odczytu skonfigurowanych danych.

React Context to funkcja należąca do biblioteki React, która umożliwia przekazanie danych do wszystkich komponentów w hierarchii. Komponent może uzyskać dostęp do tych danych za pomocą funkcji *useContext* przekazując jako argument deklaracje kontekstu. Przykład użycia opisywanej funkcji analogicznie do funkcjonalności zaimplementowanej w prototypie został pokazany w listingu 7.

Listing 7. Przykład użycia *React Context* w celu przekazania konfiguracji do wtyczki. Źródło: opracowanie własne

```
import { createContext, useContext } from 'React';

const ConfigurationContext = createContext(); // Tworzymy kontekst
const ConfigurationProvider = ({ children }) => {
    const [configuration, setConfiguration] = useState({}); // Domyślnie
ustawiamy pustą konfiguracje

    // Dostarczamy wartość kontekstu dla swoich dzieci
    return (
      <ConfigurationContext.Provider value={{ configuration, setConfiguration
}}>
```

```
        {children}
      </ConfigurationContext.Provider>
    );
};

const VisualizationComponent = () => { // Komponent wtyczki
    const { configuration, setConfiguration } =
useContext(ConfigurationContext); // Dane z kontekstu

    return (
        <div>
            Konfiguracja: { configuration }
        </div>
    );
}

const MainApp = () => { // Komponent osadzający komponent wtyczki
    return (
        <ConfigurationProvider>
            <VisualizationComponent />
        </ConfigurationProvider>
    )
}
```

Wtyczki wizualizacyjne zaimplementowane w prototypie to:

- Słupkowy wykres 3D,
- Punktowy wykres 3D,
- Wykres powierzchni 3D,
- Kostka OLAP.

5.3.2 Wtyczka integrująca zewnętrzne źródło danych – klient

Podobnie jak wtyczka wizualizacyjna, rozszerzenie może wypełniać dwie przestrzenie w systemie. Analogicznie, pierwszym miejscem jest formularz służący do konfiguracji integracji z zewnętrznym źródłem danych. Drugim miejscem jest przestrzeń zaraz obok elementu HTML z wizualizacją. Wtyczka tam może wyświetlić dodatkową dynamiczną konfigurację pozwalającą dla przykładu przefiltrować zbiór danych za pomocą odpowiednich pól typu lista, czy suwaki, jak opisano w podrozdziale 3.2.1.

Element do dynamicznej konfiguracji jest komponentem React, który otrzymuje dwa argumenty – zapisaną konfiguracje wtyczki oraz funkcję, której wywołanie spowoduje odświeżenie danych przez wtyczkę w warstwie serwerowej z użyciem przekazanych do niej informacji. Komponent dynamicznej konfiguracji jest wyświetlany w części publicznego dostępu, dlatego przekazywana do niego konfiguracja zawiera tylko obiekt, którego klucz podczas rejestracji wtyczki opisany jest jako publiczny. Takie rozwiązanie zabezpiecza wrażliwe informacje, takie jak na przykład dane uwierzytelniające do integrowanej bazy danych.

5.3.3 Wtyczka integrująca zewnętrzne źródło danych – serwer

Część wtyczki działająca w warstwie serwerowej architektury odpowiada za komunikacje z docelowym źródłem danych, którym nie musi być koniecznie baza danych. Wtyczka uruchamia się jak każdy inny pakiet w środowisku Node.js, dlatego możliwości łączenia się z zewnętrznymi serwisami nie są poddane ograniczeniom.

Zadaniem części serwerowej rozszerzenia jest również komunikacja z częścią przeglądarkową wtyczki. Podczas rejestracji, wtyczka określa funkcje, które mogą zostać użyte w warstwie klienta. Za wywołanie tych metod i przekazanie informacji zwrotnych, odpowiada menedżer wtyczek. Diagram ukazujący sposób komunikacji pomiędzy warstwami wtyczek pokazano na rysunku 7.

Wtyczki integrujące zewnętrzne źródła danych zaimplementowane w prototypie to:

- MySQL,
- PostgreSQL,
- MongoDB.

5.4. Menedżer wtyczek

Kluczową rolę w działaniu całego systemu odgrywa menedżer wtyczek, który pozwala zarejestrować oraz uruchomić wtyczkę po stronie serwera oraz po stronie klienta aplikacji. Menedżer wtyczek jest zaimplementowany jako oddzielny pakiet w monorepozytorium, dzięki czemu wtyczki oraz główna aplikacja mają dostęp do jego wyeksportowanych pomocniczych komponentów React, klas oraz funkcji.

5.4.1 Tworzenie wtyczek

Pierwszym krokiem do utworzenia nowej wtyczki jest zadecydowanie w jakiej warstwie architektury będzie ona działać. Na potrzeby niniejszego przykładu rozpatrzymy integracje wtyczki MySQL działającej w obu warstwach, która pozwala połączyć się ze skonfigurowaną przez użytkownika bazą danych.

Kolejnym etapem jest stworzenie dwóch folderów w katalogu *plugins* nazwanych odpowiednio *mysql-server* dla warstwy serwerowej oraz *mysql-client* dla warstwy klienta. Wymagane jest, aby nazwa wtyczki składała się z trzech elementów:

- *@linkchart-plugin/* - definiuje, że konfigurowany pakiet jest wtyczką, przez co w późniejszych etapach zostanie ona poprawnie wykryta,
- *nazwa-wtyczki* – unikalna nazwa wtyczki,
- *-client* lub *-server* – wybór warstwy architektury, na której zostanie uruchomiona wtyczka.

Dlatego, w utworzonych uprzednio folderach zainicjowano projekty JavaScript i nazwano je kolejno *@linkchart-plugin/mysql-server* i *@linkchart-plugin/mysql-client*. Przykładową konfiguracje pliku *package.json* dla wtyczki serwerowej przedstawiono w listingu 8.

Listing 8. Konfiguracja pliku *package.json* dla serwerowej wtyczki MySQL. Źródło: opracowanie własne.

```
{
    "name": "@linkchart-plugin/mysql-server",
    "private": true,
    "version": "0.0.0",
```

```
        "description": "",
        "module": "./dist/index.js",
        "scripts": {
            "dev": "npm run build",
            "build:plugin": "npm run build:esm",
            "build:esm": "tsc"
        },
        "devDependencies": {
            "@linkchart/plugin-manager": "*",
            "typescript": "^4.8.2"
        },
        "peerDependencies": {
            "@linkchart/plugin-manager": "*"
        },
        "dependencies": {
            "mysql": "^2.18.1"
        }
    }
```

5.4.2 Rejestracja wtyczki

Rejestracja wtyczki odbywa się poprzez wywołanie odpowiedniej funkcji wyeksportowanej z menedżera wtyczek. Wybór funkcji zależy od przeznaczenia wtyczki. Jak opisano w podrozdziale 5.3., oprogramowanie przewiduje dwa rodzaje wtyczek – wizualizacji oraz bazy danych. Wtyczka wizualizacyjna jest uruchamiana tylko w przeglądarce klienta, z kolei wtyczka bazodanowa również na serwerze. Dlatego, dostępne są trzy funkcje pozwalające zarejestrować wtyczki:

- *defineVisualizationPlugin* – funkcja do rejestracji wtyczki wizualizacyjnej,
- *defineDatabasePlugin* – metoda pozwalająca na zarejestrowanie wtyczki do integracji zewnętrznych danych po stronie klienta,
- *defineDatabaseServerPlugin* – rejestracja wyżej opisanej wtyczki, ale po stronie serwera.

Wywołanie powyższych metod odbywa się w pliku wejściowym wtyczki, który jest definiowany we właściwości *module* w pliku *package.json* (listing 8). Pierwszym argumentem w powyższych metodach zawsze jest identyfikator w postaci ciągu znaków.

Funkcja rejestrująca wtyczkę wizualizacyjną przyjmuje argumenty, które zostały wylistowane poniżej, a jej użycie wyszczególniono w tabeli 2.

Tabela 2. Opis argumentów wymaganych do rejestracji wtyczki wizualizacji. Źródło: opracowanie własne

Nazwa argumentu	Opis argumentu
configurationComp	referencja do komponentu React służącego do konfiguracji wizualizacji

getVisualization	referencja do metody, która ma za zadanie zwrócić element HTML z wyświetloną wizualizacją
name	nazwa reprezentacyjna, która jest wyświetlana na liście wyboru wtyczki

Listing 9. Rejestracja wtyczki słupkowego wykresu 3D. Źródło: opracowanie własne

```
import { defineVisualizationPlugin } from "@linkchart/plugin-manager";
import { getVisualization } from "./visualization/getVisualization";
import Main from "./Main";

defineVisualizationPlugin("3d-point-graph", {
    configurationComp: Main,
    getVisualization: getVisualization,
    name: "3D Point Graph",
});
```

Rejestracja wtyczki bazodanowej (listing 10) w warstwie klienta wymaga danych opisanych w tabeli 3.

Tabela 3. Opis argumentów wymaganych do rejestracji wtyczki integracji z zewnętrznym źródłem danych (klient). Źródło: opracowanie własne

Nazwa argumentu	Opis argumentu
configurationComp	referencja do komponentu React służącego do konfiguracji wtyczki
dynamicConfigurationComp	komponent React z dynamiczną konfiguracją wystawiony do dostępu publicznego (szczegółowy opis w podrozdziale 5.3.2)
name	nazwa wyświetlana

Listing 10. Rejestracja wtyczki MySQL w warstwie klienta. Źródło: opracowanie własne

```
import { defineDatabasePlugin } from "@linkchart/plugin-manager";
import Dynamic from "./components/dynamic/Dynamic";
import Main from "./Main";

defineDatabasePlugin("mysql", {
    name: "MySQL",
    configurationComp: Main,
    dynamicConfigurationComp: Dynamic,
});
```

W celu zarejestrowania wtyczki po stronie serwera, wymagane są dane wypisane w tabeli 4.

Tabela 4. Opis argumentów wymaganych do rejestracji wtyczki integracji z zewnętrznym źródłem danych (serwer). Źródło: opracowanie własne

Nazwa argumentu	Opis argumentu
customMethods	zbiór udostępnionych metod serwerowych, które mogą zostać wywołane przez wtyczkę w przeglądarce
getData	referencja do funkcji pobierającej dane ze źródła
dynamicConfigurationKey	klucz obiektu w konfiguracji, który jest przeznaczony do dostępu publicznego (szczegółowy opis w podrozdziale 5.3.2)

Definicja części serwerowej wtyczki MySQL przedstawiono w listingu 11.

Listing 11. Rejestracja serwerowej części wtyczki MySQL. Źródło: opracowanie własne

```
import { defineDatabaseServerPlugin } from "@linkchart/plugin-manager";
import { getData, getSampleData } from "./getData";
import { connect, isConnectionValid } from "./methods/connect";
import { getColumns, getConfigurationColumns } from
"./methods/getColumns";
import { getTables } from "./methods/getTables";

defineDatabaseServerPlugin("mysql", {
    customMethods: {
        connect: connect,
        isConnectionValid: isConnectionValid,
        getTables: getTables,
        getColumns: getColumns,
        getConfigurationColumns: getConfigurationColumns,
        getSampleData: getSampleData,
    },
    getData: getData,
    dynamicConfigurationKey: "dynamicFilters",
});
```

Ostatnim krokiem jest wpisanie nazwy projektu wtyczki do listy, która jest udostępniona w pliku *plugins.json* w głównej aplikacji. W listingu 12 przedstawiono zawartość pliku po stworzeniu i wpisaniu wszystkich wtyczek znajdujących się w prototypie.

Listing 12. Zawartość pliku *plugins.json* z listą wtyczek. Źródło: opracowanie własne

```
[
    "postgres-client",
    "postgres-server",
    "mongodb-client",
    "mongodb-server",
    "mysql-client",
    "mysql-server",
    "olap-cube",
    "3d-bar-graph",
    "3d-point-graph",
    "3d-surface-graph"
]
```

5.4.3 Udostępnione komponenty dla wtyczek

Menedżer udostępnia szereg komponentów wylistowanych poniżej, które upraszczają budowanie wtyczek uruchamianych w przeglądarce. Wygląd każdego z nich został przystosowany do szablonu aplikacji.

- <*Collapse* /> - blok z możliwością zwijania,
- <*Input* /> - edytowalne pole formularzowe,
- <*Select* /> - lista wyboru,
- <*Button* /> - przycisk,
- <*Expression* /> - blok z możliwością wpisania i uruchomienia równania ze składnią zgodną z biblioteką opisaną w podrozdziale 4.10.

Komponent *Expression* pozwala na stworzenie dynamicznego podprogramu, który na przykładzie wizualizacji, w zależności od danych, może zmieniać wygląd wykresu i nie tylko. Przykładowy podprogram zmieniający kolor słupków na czerwony, jeśli wartość osi Z przekroczyła 20 pokazano na listingu 13.

Listing 13. Przykładowy program dynamicznego równania dla koloru słupków na wykresie. Źródło: opracowanie własne

```
if z > 20 then '#ff0000' else '#000000' endif
```

Użycie udostępnionych komponentów nie jest wymagane, aczkolwiek zaleca się ich implementacje. Dzięki temu, kod źródłowy na przestrzeni aplikacji jest ustandaryzowany i czytelny. Co za tym idzie, utrzymanie oprogramowania staje się łatwiejsze. Przykładowo, chęć zmiany wyglądu szablonu, przy nieużyciu opisywanych komponentów, będzie wymagała edycji każdej wtyczki, aby dostosować jej wygląd do nowego projektu graficznego. Z kolei w sytuacji, gdzie każdy moduł używa tego samego komponentu, zmianę wyglądu należy dokonać tylko w tym jednym komponencie, a widok zostanie zaktualizowany na przestrzeni wszystkich wtyczek.

5.5. Aplikacja

Aplikacje główną, osadzającą wtyczki i zapewniającą docelowe funkcjonalności opracowano w technologii Next.js. Prototyp umożliwia zalogowanie się za pomocą konta Google, stworzenie projektu wykresu oraz wybór wtyczek integracji z zewnętrznym źródłem danych i wizualizacji. Dodatkowo, umożliwia udostępnienie publicznego adresu URL, pod którym wyświetli się skonfigurowana wizualizacja. W tym podrozdziale został opisany sposób wyszukiwania i zaczytywania oraz osadzania wtyczek w aplikacji.

5.5.1 Wyszukiwanie i wczytywanie wtyczek

W aplikacji Next.js plikiem konfiguracyjnym jest *next.config.mjs*, który wykonuje zaimplementowany tam kod w momencie uruchamiania aplikacji. W tym miejscu został zaprogramowany mechanizm wyszukiwania wtyczek.

W listingu 14 przedstawiono kod źródłowy wyszukiwania rozszerzeń. Przy pomocy Lerna.js wyświetlono wszystkie paczki znajdujące się w monorepozytorium i wyselekcjonowano tylko te, których nazwa zaczyna się od prefiksu *@linkchart-plugin/*. Następnie, po sprawdzeniu obecności na liście z listingu 13, wtyczkę przydzielono do warstwy klienta lub serwera w zależności od sufiksu *-client*, lub *-server*. Celem tego mechanizmu jest stworzenie dwóch plików tymczasowych potrzebnych na czas działania aplikacji, celem ich zaimportowania w późniejszym etapie, co automatycznie spowoduje uruchomienie metod opisanych w podpunkcie 5.4.2.

Listing 14. Wyszukiwanie i wczytywanie wtyczek podczas uruchamiania aplikacji. Źródło: opracowanie własne

```
// Pobranie wszystkich pakietów
const list = JSON.parse(await execute("lerna ls -all --json"));
const allPluginsList = list.filter(({ name }) =>
    name.includes(PLUGIN_PREFIX)
);

// Pobranie listy wtyczek do wgrania do aplikacji
const requestedPlugins = JSON.parse(readFileSync("./plugins.json"));

for (const pluginName of requestedPlugins) {
    const fullPluginName = `${PLUGIN_PREFIX}${pluginName}`;
    const registeredPlugin = allPluginsList.find(
        ({ name }) => name === `${fullPluginName}`
    ); // Sprawdzanie obecności na liście
    if (registeredPlugin) {
        // Jeśli sufiks nie istnieje, wtyczka jest dodawana do warstwy
klienta
        if (pluginName.includes("-client"))
            importFileClientContent += `import "${fullPluginName}";\n`;
        else if (pluginName.includes("-server"))
            importFileServerContent += `import "${fullPluginName}";\n`;
        else importFileClientContent += `import "${fullPluginName}";\n`;
```

```
            log(`Installed plugin ${fullPluginName}`);
        } else log(`Unable to find plugin ${fullPluginName}`, "warn");
    }

    // Zapis plików tymaczasowych
    writeFileSync("./plugins/plugins-client.temp.ts",
    importFileClientContent);
    writeFileSync("./plugins/plugins-server.temp.ts",
    importFileServerContent);
```

5.5.2 Osadzanie wtyczek

Osadzanie wtyczek w aplikacji odbywa się za pomocą menedżera wtyczek. W zależności od miejsca, gdzie komponenty React zdefiniowane w rozszerzeniach mają być wyświetlone, pobierana jest odpowiednia lista, na przykład wtyczek wizualizacyjnych, z których selekcjonowana jest aktualnie wybrana wtyczka, finalnie wyświetlana w komponencie wizualizacyjnym. Kod źródłowy osadzenia wtyczki wizualizacyjnej pokazano w listingu 15. Analogiczne postępowanie jest przy wtyczkach integrujących zewnętrzne dane. Jedyną różnicą jest miejsce ich wyświetlania. Każdy rodzaj wtyczek jest wyświetlany w różnych zakładkach.

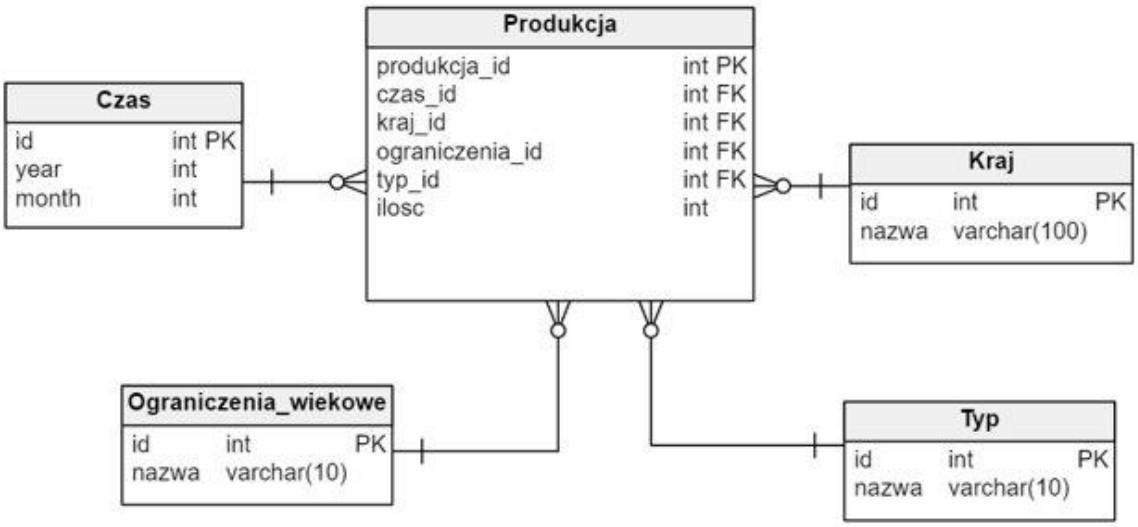
Listing 15. Uproszczony kod źródłowy osadzania wtyczki wizualizacji. Źródło: opracowanie własne

```
import { React } from 'react';
import { PluginContext, PluginManager } from "@linkchart/plugin-manager";
import ChartEditContext from "../../../../contexts/ChartEditContext";

const ChartEditVisualization: React.FC = () => {
    const { chart, setChart } = useContext(ChartEditContext);

    const [data, setData] = useState<IData>(null);

    const plugins = PluginManager.getVisualizationPlugins();
    const chartConfiguration = chart?.configuration;
    const visId = chartConfiguration?.visualization?.pluginId;
    const selectedVis = visId && plugins.find((plugin) => plugin.id ===
visId);
    const PluginComponent = selectedVis && selectedVis.configurationComp;

    const updateConfigurationBridge = async (
        pluginId: string,
        configuration: Record<string, any> | null
    ) => { /* ... */ };

    return (
        <PluginContext.Provider
            value={{
                configuration:
                    chartConfiguration?.visualization
```

```
                        ?.pluginConfiguration ?? null,
                    saveConfiguration: async (conf) => {
                        await updateConfigurationBridge(
                            selectedVis.id,
                            conf
                        );
                    },
                    callServerMethod: () =>
                        Promise.resolve(),
                }}
            >
                <PluginComponent data={data} />
            </PluginContext.Provider>
        )
    }
```

6. Zastosowanie prototypu aplikacji

W tym rozdziale opisano i przedstawiono w postaci zrzutów ekranu zastosowanie rozwiązania. Niniejsze podrozdziały pokażą proces wizualizacji integracji oraz wizualizacji danych na przykładzie udostępnionej bazie danych dotyczących produkcji na platformie Netflix [31]. Dane wcześniej zostały przetworzone i wgrane do relacyjnej bazy danych. Schemat struktury danych przedstawiono na rysunku 9.



Rysunek 9. Schemat bazy danych Netflix. Źródło: opracowanie własne.

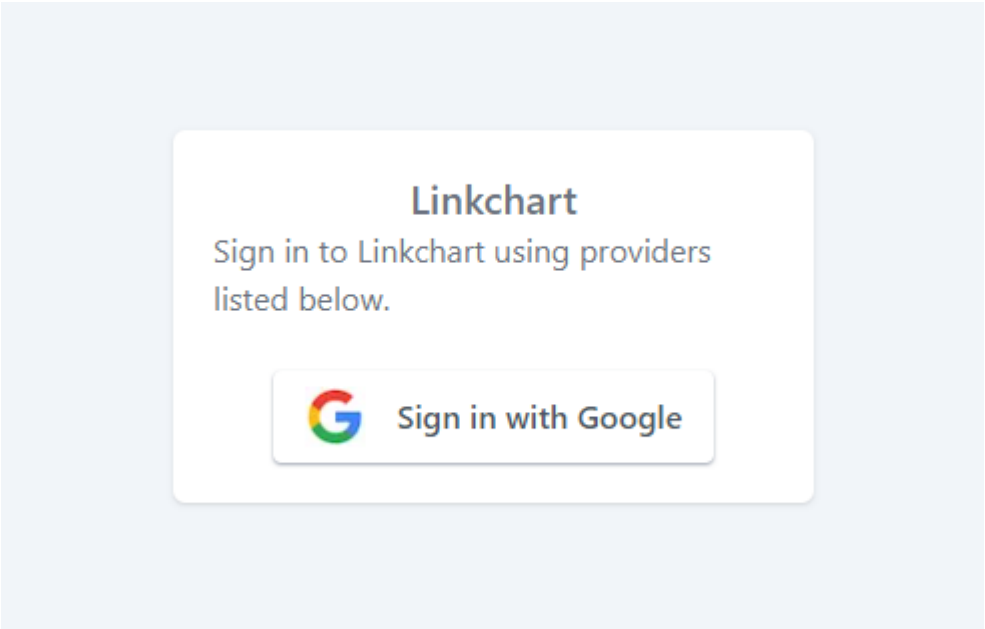
Proces obejmuje logowanie, zintegrowanie danych, konfiguracje wizualizacji oraz nałożenie dynamicznych filtrów na zaprojektowany wykres.

Rozdział przedstawia przykładowy proces wizualizacji ilości produkcji w zależności od roku jej powstania oraz kraju pochodzenia. Założenia wizualizacji:

- Produkcje z lat 2018-2022,
- Kraje pochodzenia: Stany Zjednoczone, Wielka Brytania, Indie oraz Polska.

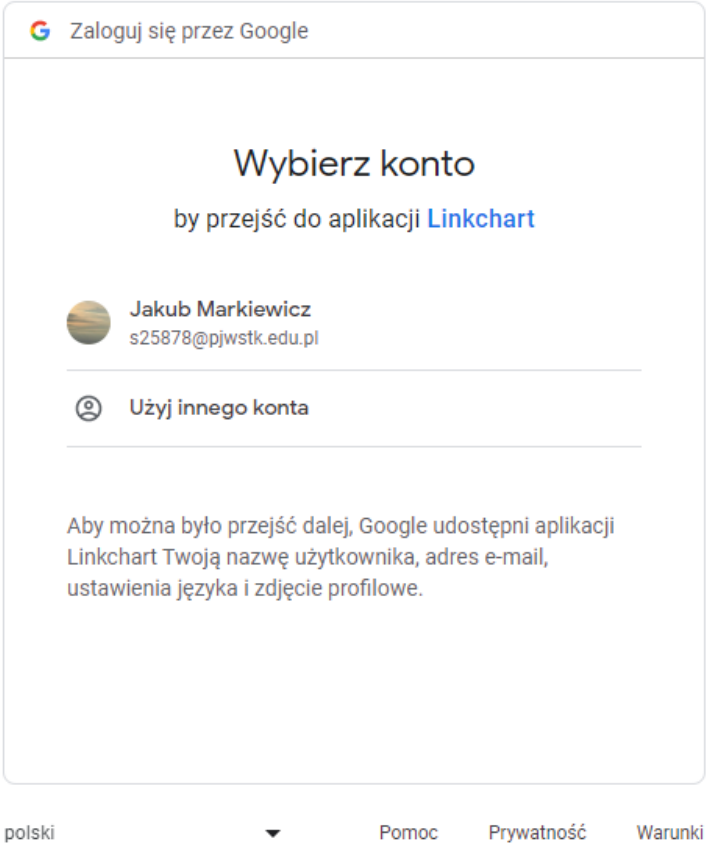
6.1. Logowanie

Po wprowadzeniu adresu, na którym uruchomiona jest aplikacja, przeglądarka wyświetli podstronę (rysunek 10), która wskazuje na potrzebę zalogowania się do systemu. W prototypie zaimplementowano tylko logowanie się za pomocą konta Google, aczkolwiek możliwe jest dodawanie większej ilości dostawców kont.



Rysunek 10. Ekran logowania do aplikacji. Źródło: opracowanie własne.

Kliknięcie w przycisk *Sign in with Google* spowoduje otwarcie okna (rysunek 11) z procesem logowania do konta Google. Wynikiem poprawnego zalogowania jest przekierowanie do głównego panelu opisanego w następnym podrozdziale.

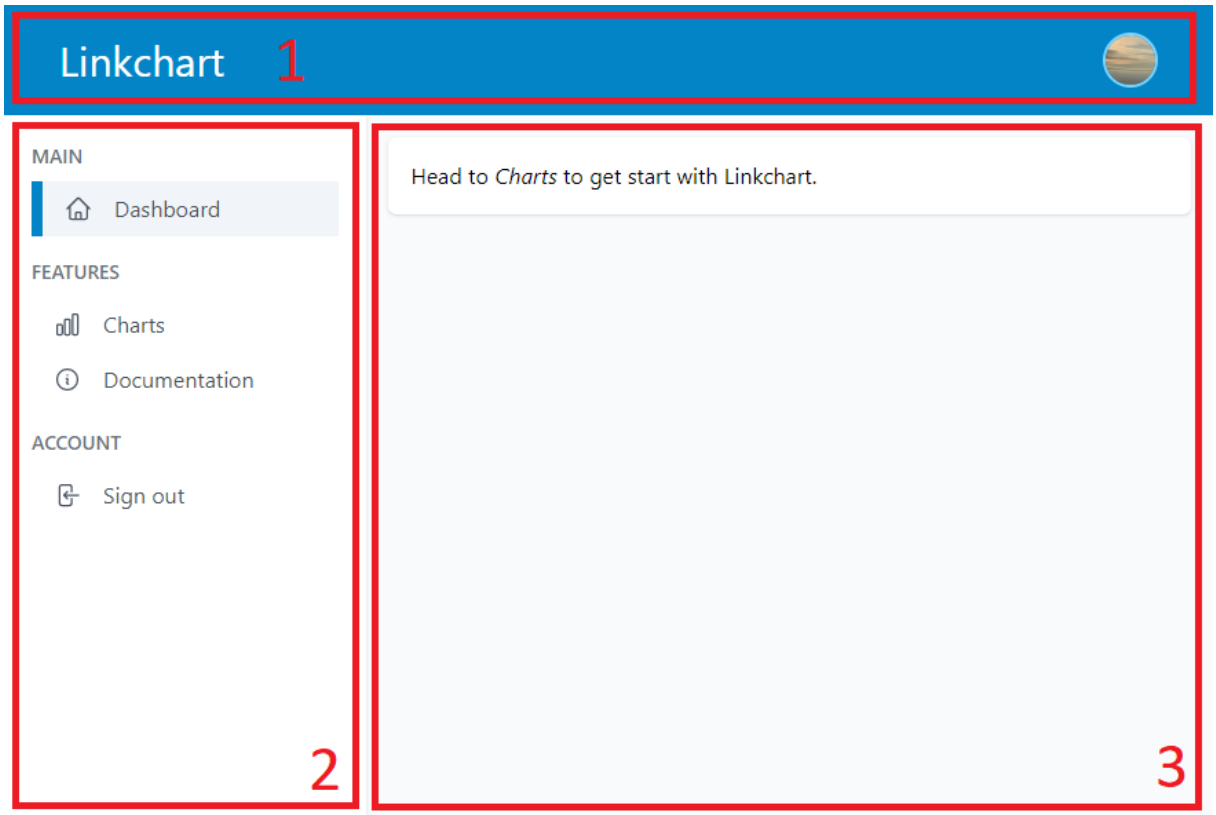


Rysunek 11. Logowanie do konta Google. Źródło: [32].

6.2. Panel główny

Panel główny aplikacji, widoczny na rysunku nr 12, został podzielony na trzy części:

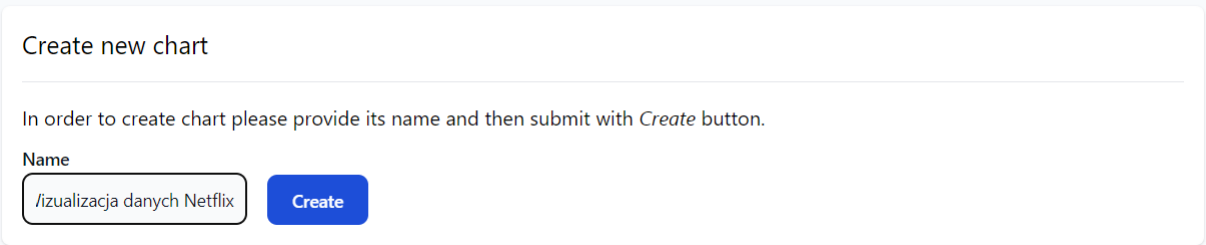
- Nagłówek (1) z nazwą oprogramowania, oraz miniaturą zdjęcia profilowego pobieranego z konta Google,
- Panel boczny (2) zawierający listę odniesień do niżej wymienionych podstron:
    - *Dashboard* – strona startowa panelu głównego,
    - *Charts* – lista projektów wizualizacji,
    - *Documentation* – dokumentacja bloku *Expression*, który został szczegółowo opisany w podrozdziale 5.4.3,
    - *Sign out* – możliwość wylogowania się z konta.
- Blok z zawartością poszczególnych podstron (3).



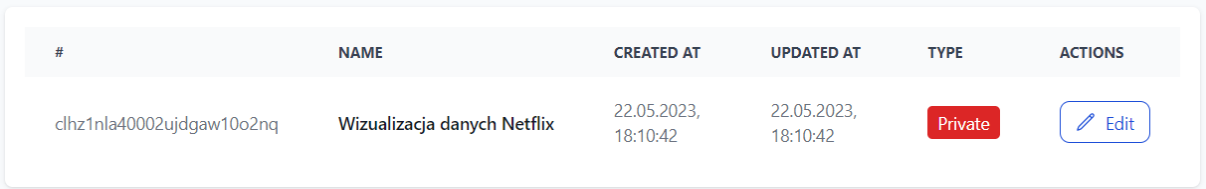
Rysunek 12. Ekran startowy panelu głównego. Źródło: opracowanie własne.

6.3. Tworzenie wizualizacji

W celu stworzenia wizualizacji wymagane jest przejście do zakładki *Charts*. Następnie, w polu *Name* (rysunek 13) należy wpisać docelową nazwę projektu. Dla tego przykładu została użyta nazwa *„Wizualizacja danych Netflix”*. Po prowadzeniu nazwy, kliknięcie przycisku *Create* spowoduje pojawienie się stworzonego projektu na liście widocznej na rysunku 14.



Rysunek 13. Tworzenie wykresu. Źródło: Opracowanie własne.



Rysunek 14. Lista zapisanych projektów wizualizacji. Źródło opracowanie własne.

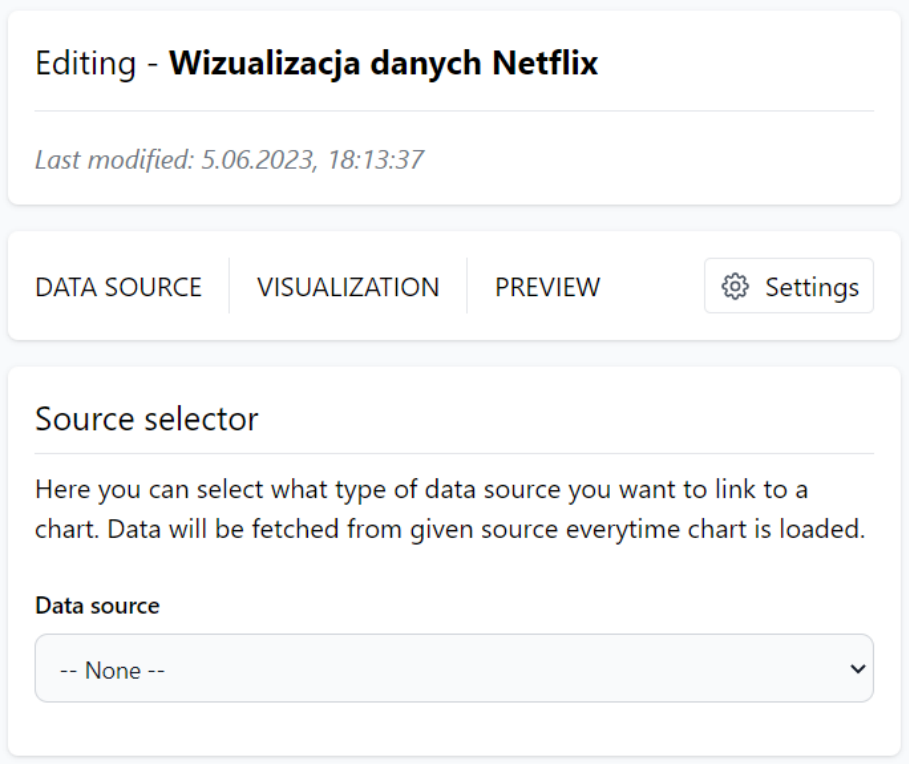
Opis znaczenia poszczególnych kolumn z widocznych na liście na rysunku 14 przedstawiono w tabeli 5.

Tabela 5. Opis kolumn listy projektów. Źródło: opracowanie własne.

Nazwa kolumny	Znaczenie kolumny
#	Identyfikator projektu w postaci ciągu znaków w formacie CUID [33]
NAME	Nazwa wizualizacji
CREATED_AT	Data i czas stworzenia
UPDATED_AT	Data i czas ostatniej aktualizacji
TYPE	Kolumna zawiera jedno z dwóch wartości: *Private* – projekt prywatny, *Public* – projekt udostępniony
ACTIONS	Akcje, które można wykonać na projekcie

6.4. Konfiguracja wizualizacji

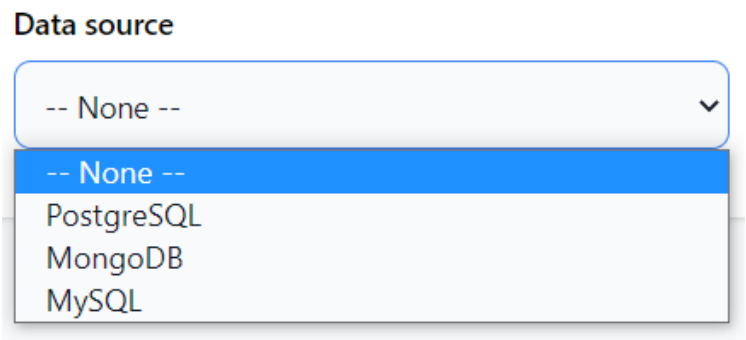
Po kliknięciu przycisku *Edit* z rysunku 14 zostanie wyświetlona podstrona widoczna na rysunku 15. Widok składa się z trzech części – nagłówka, nawigacji oraz zawartości. Proces konfiguracji należy zacząć od wybrania i dostosowania źródła danych.



Rysunek 15. Widok konfiguracji projektu. Źródło: opracowanie własne.

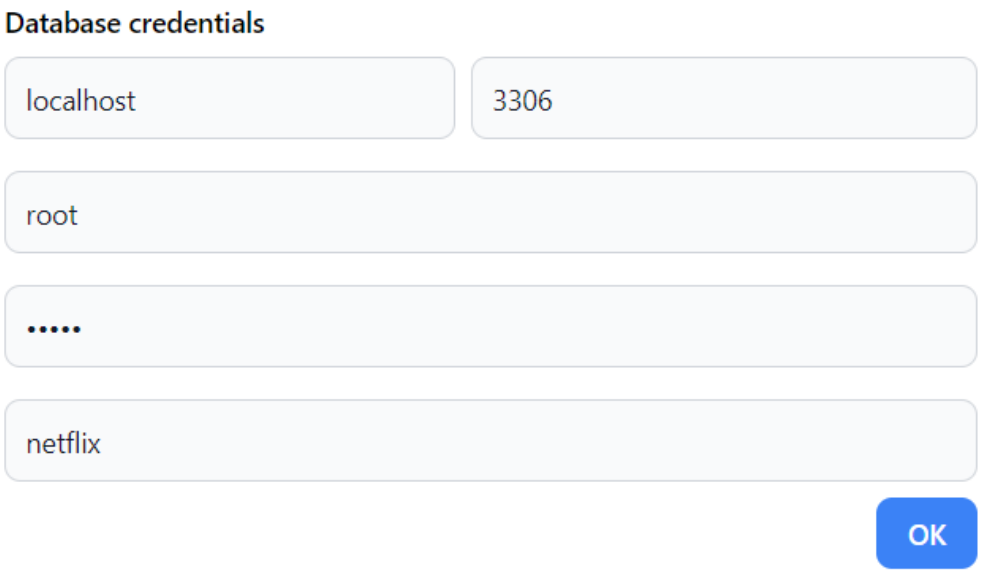
6.4.1 Integracja z zewnętrznym źródłem danych

Jak zaznaczono na początku rozdziału, wykres zostanie zintegrowany z danymi o produkcjach filmowych Netflix. Dane zostały wgrane do bazy danych MySQL, dlatego taka pozycja została wybrana na liście z rysunku 16. Pozycje na liście to nazwy zdefiniowanych wtyczek opisanych w podrozdziale 5.3.

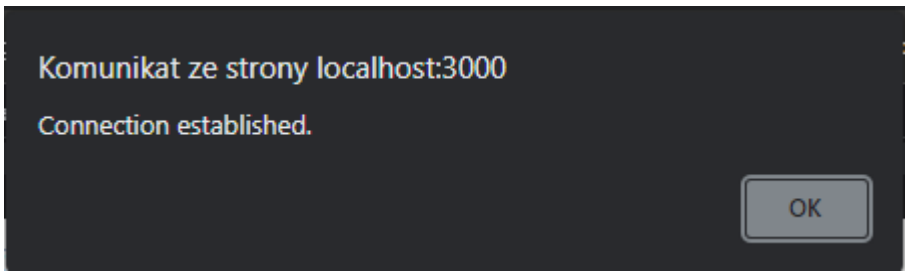


Rysunek 16. Wybór integracji z zewnętrznym źródłem danych. Źródło: opracowanie własne.

Po wybraniu wtyczki, zostanie wyświetlona jej konfiguracja. Pierwszy krok to połączenie z bazą danych poprzez wpisanie danych uwierzytelniających w pola przedstawione na rysunku 17. Kliknięcie przycisku *OK* rozpocznie próbę połączenia. Informacja zwrotna jest wyświetlana w postaci powiadomienia (rysunek 18).

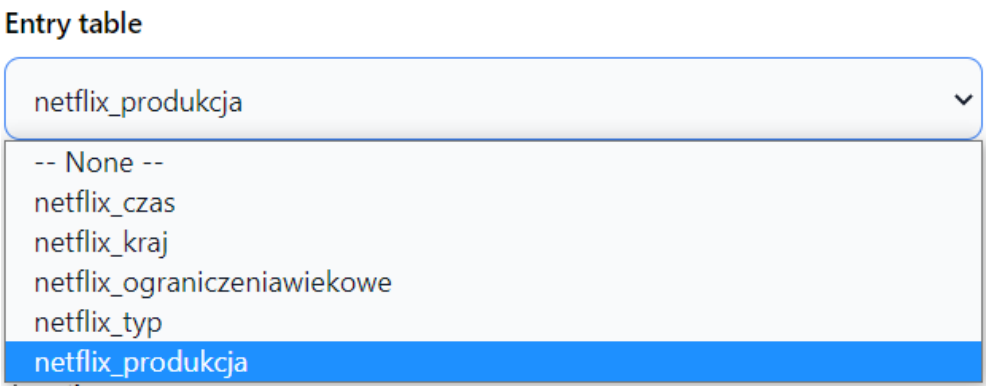


Rysunek 17. Konfiguracja danych uwierzytelniających. Źródło: opracowanie własne.



Rysunek 18. Powiadomienie z informacją zwrotną dot. połączenia. Źródło: opracowanie własne.

Następny krok to wybranie tabeli wejściowej przechowującej dane, które posłużą nam do wizualizacji. W tym przykładzie taką tabelą jest *netflix_produkcja*, która zawiera listę produkcji filmowych.



Rysunek 19. Wybór tabeli wejściowej. Źródło: opracowanie własne.

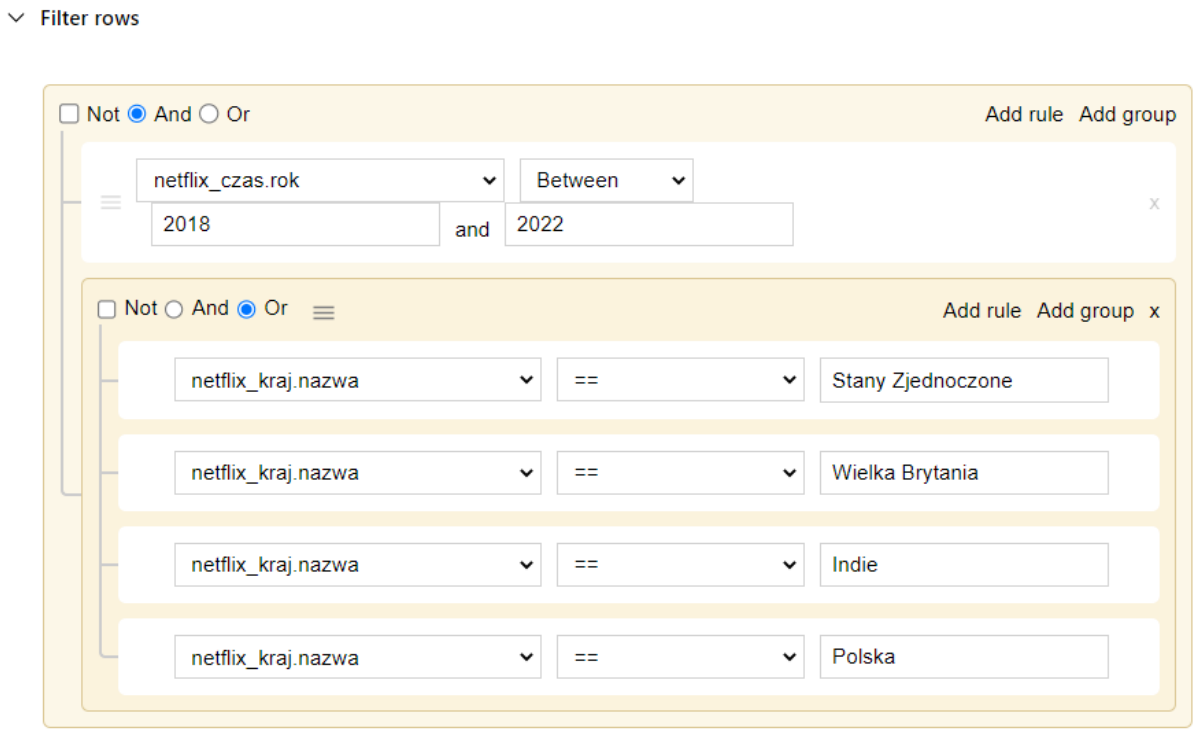
W tym przypadku, wizualizowane dane są danymi wielowymiarowymi. Znając założenie, że docelowo wizualizacja ma obejmować ilość produkcji względem roku powstania oraz kraju

pochodzenia, takie wymiary należy połączyć do tabeli wejściowej. Sposób łączenia tabel wymiarowych został przedstawiony na rysunku 20.

Rysunek 20. Łączenie tabel wymiarowych. Źródło: opracowanie własne.

Tabela *czas* z bazy danych o nazwie *netflix* została połączona kolumną *czas_id* do klucza obcego z wejściowej tabeli *produkcja* o nazwie *czas_id*. Analogicznie w przypadku tabeli *kraj*.

W następnym kroku (rysunek 21) zdefiniowano filtry, zgodnie z założeniami, jakie mają zostać nałożone na dane źródłowe.

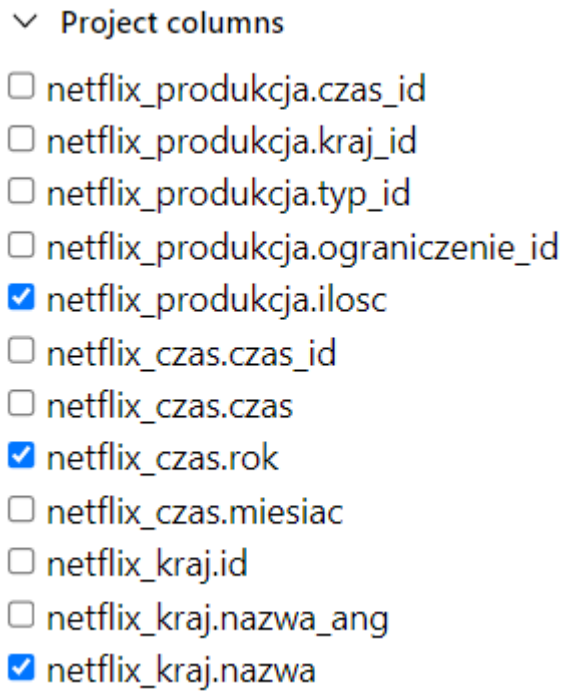


Rysunek 21. Filtrowanie danych źródłowych. Źródło: opracowanie własne.

Opis słowny zaprojektowanego drzewa filtru:
Rok pomiędzy wartością 2018, a 2022 ***oraz*** *nazwa kraju taka jak Stany Zjednoczone* ***lub*** *Wielka Brytania* ***lub*** *Indie* ***lub*** *Polska.*

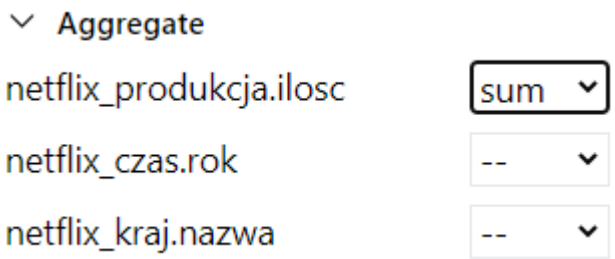
W podrozdziale 6.5. opisano użycie filtrów dynamicznych pozwalających na zintegrowanie danych z rysunku 21 z wartościami konfigurowalnymi przez użytkownika wyświetlającego wizualizacje.

Następną czynnością jest wskazanie, które kolumny mają być zawarte strukturze zwracanych danych. Odbywa się to poprzez zaznaczenie pól wyboru (rysunek 22) przy nazwach kolumn.



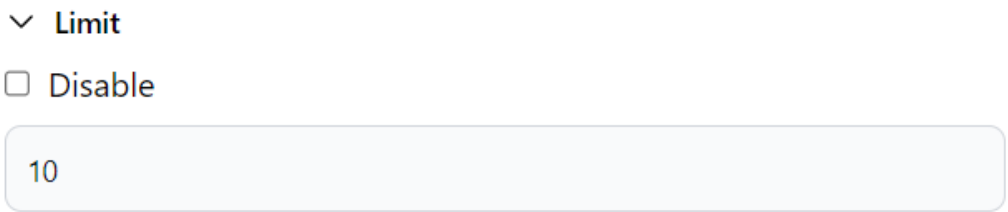
Rysunek 22. Wybór kolumn danych źródłowych. Źródło: opracowanie własne.

Wiedząc z założeń, że wizualizacja docelowo ma przedstawiać zsumowaną ilość produkcji dla danego kraju oraz czasu, w zakładce *Aggregate* należy wybrać dla kolumny *ilosc* metodę agregacji *sum* (rysunek 23).



Rysunek 23. Agregacja danych źródłowych. Źródło: opracowanie własne.

Wtyczka umożliwia również nałożenie limitu na ilość wierszy zwracanych danych (rysunek 24). Przykładowe ustawienie limitu na wartość 10 przedstawiono na rysunku 24.



Rysunek 24. Limit zwracanych danych źródłowych. Źródło: opracowanie własne.

Tak skonfigurowane dane można teraz wyświetlić w zakładce *Data preview* przedstawionej na rysunku 25. Podgląd danych umożliwia pobranie maksymalnie dziesięciu wierszy.

Data preview

showing up to 10 results

Refresh

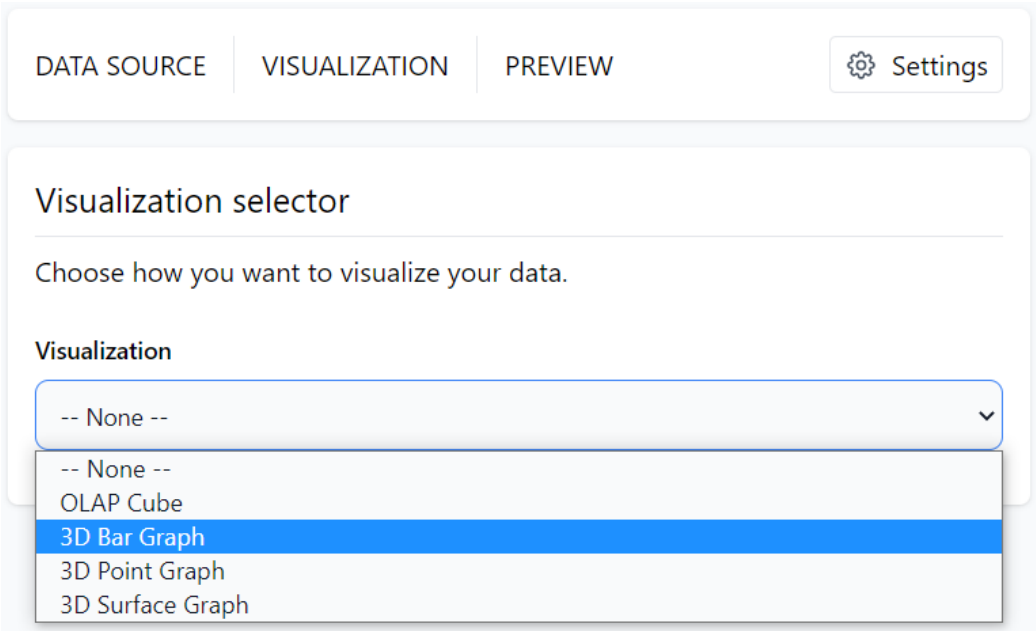
sum(netflix_produkcja.ilosc)	netflix_czas.rok	netflix_kraj.nazwa
339	2018	Indie
3	2018	Polska
502	2018	Stany Zjednoczone
116	2018	Wielka Brytania
212	2019	Indie
11	2019	Polska
768	2019	Stany Zjednoczone
155	2019	Wielka Brytania
193	2020	Indie
10	2020	Polska

Rysunek 25. Podgląd danych źródłowych po konfiguracji. Źródło: opracowanie własne.

W analogiczny sposób przebiega konfiguracja pozostałych wtyczek (MongoDB oraz PostgreSQL).

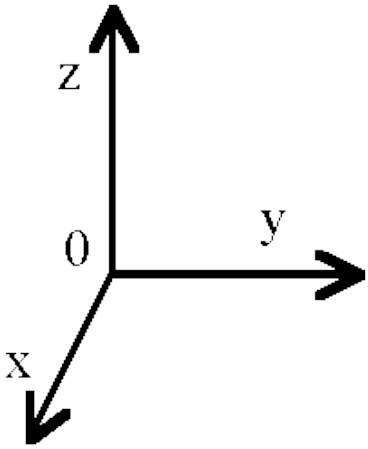
6.4.2 Konfiguracja wizualizacji

Przejście do zakładki *VISUALIZATION* rozpoczyna proces konfiguracji wizualizacji. Na liście wyboru znajdują się cztery rodzaje wizualizacji, które zostały wymienione w podpunkcie 5.3.1. Dla opisywanego przypadku zostanie wygenerowany trójwymiarowy wykres słupkowy, dlatego z listy na rysunku 26 została wybrana pozycja o nazwie *3D Bar Graph*.



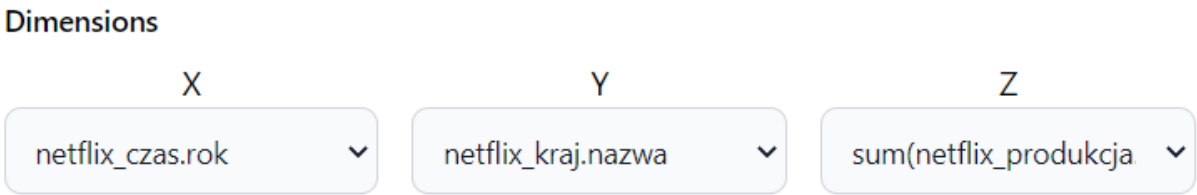
Rysunek 26. Wybór wizualizacji. Źródło: opracowanie własne.

W pierwszym kroku należy dopasować kolumny ze struktury danych do osi współrzędnych. Zaimplementowane wtyczki w prototypie używają kartezjańskiego układu współrzędnych widocznego na rysunku 27.



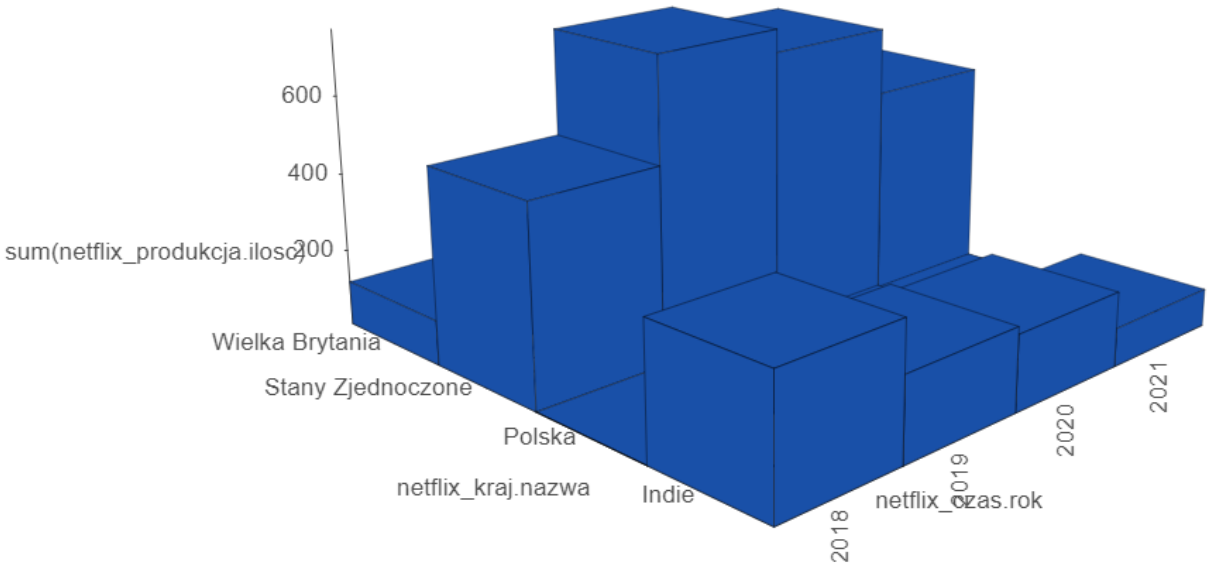
Rysunek 27. Kartezjański układ współrzędnych. Źródło: [34]

Słupki na wykresie są rysowane wzdłuż osi *Z*, dlatego do parametru *Z* z rysunku 28 wybrana została kolumna ilości produkcji. Do osi *X* przedzielono wartości dotyczące roku produkcji, a do osi *Y* nazwy krajów.



Rysunek 28. Przydzielanie kolumn do układu współrzędnych. Źródło: opracowanie własne

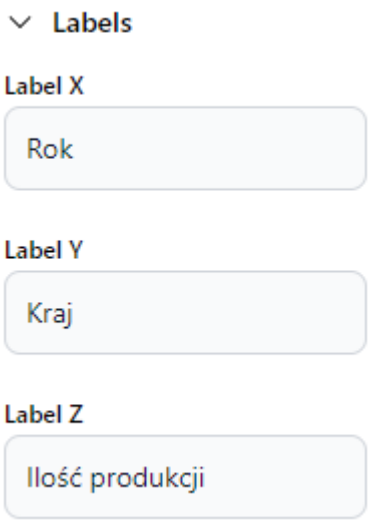
Po wykonaniu poprzedniego kroku, bez dalszej konfiguracji wygenerowano wykres i umieszczono na rysunku 29. Z racji niskiej czytelności wygenerowanej wizualizacji, dla tego przypadku, należy dostosować grubość słupków, rozmiar czcionki oraz dodać etykiety dla każdej osi.



Rysunek 29. Domyślny wygląd wykresu słupkowego 3D bez konfiguracji. Źródło: opracowanie własne.

W zakładce *Labels* (rysunek 30) istnieje możliwość nadpisania etykiety dla każdej z osi. Celem poprawy czytelności wykresu, zostały wpisane poniższe wartości:

- Oś X – Rok,
- Oś Y – Kraj,
- Os Z – Ilość produkcji.



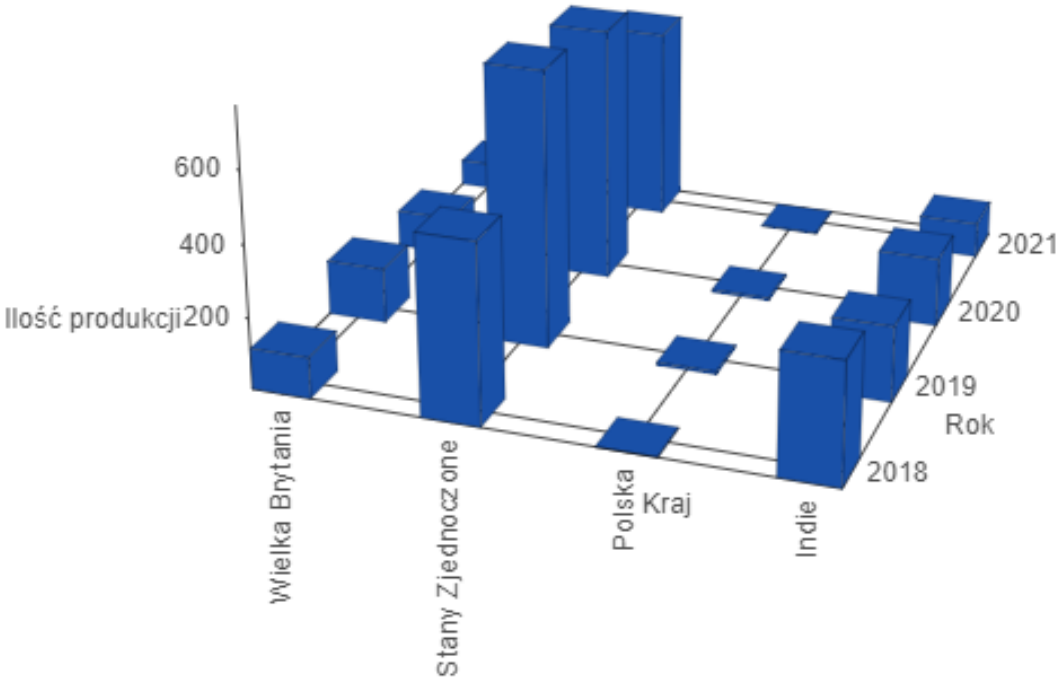
Rysunek 30. Nadpisanie etykiet przy osiach współrzędnych. Źródło: opracowanie własne.

Następna zakładka konfiguracyjna *Sizes* pozwala na zmianę rozmiaru słupków oraz wielkości czcionki. Wartości domyślne zostały przedstawione na rysunku 31. Metodą empiryczną zostały dobrane takie nowe wartości, aby wykres był czytelny po wstawieniu do niniejszej pracy:

- *Bar width* (szerokość słupka) – 0.35,
- *Bar height* (grubość słupka) – 0.35,
- *Axis text* (rozmiar tekstu na osiach) – 60.

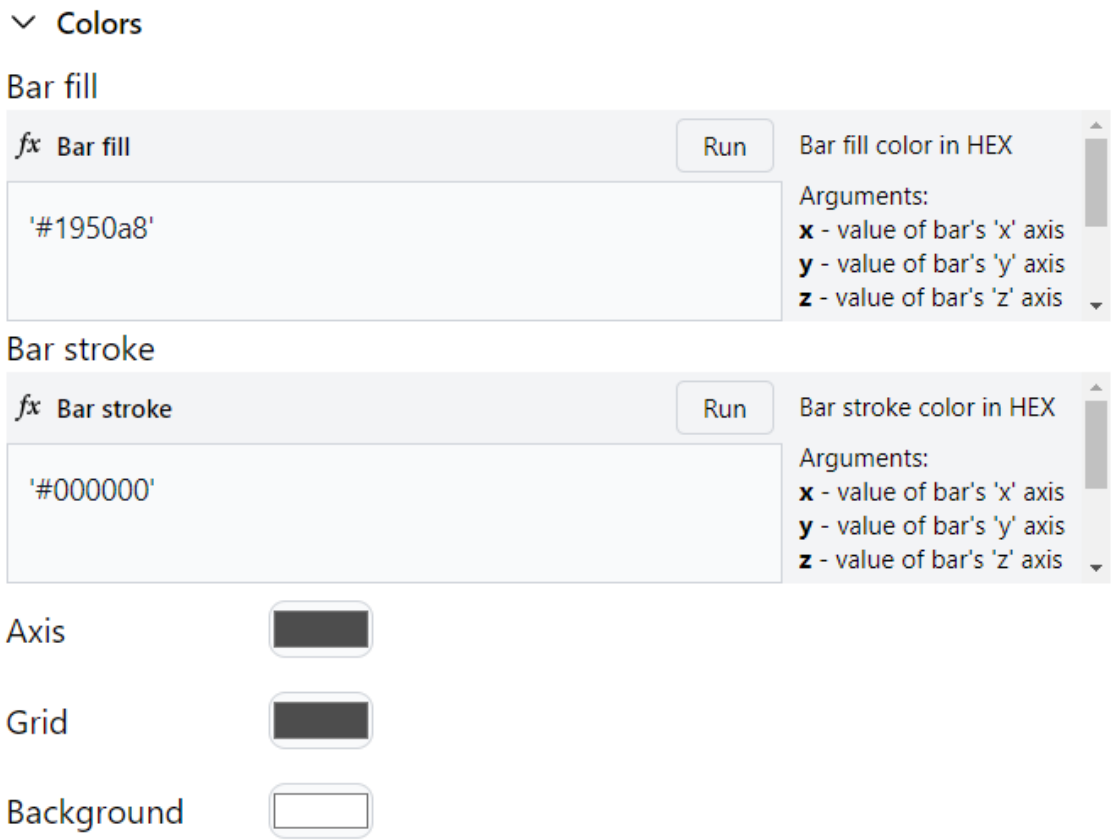
Rysunek 31. Domyślne wartości rozmiarów elementów wykresu słupkowego 3D. Źródło: opracowanie własne.

Wykres z nowymi wartościami rozmiarów pokazano na rysunku 32.



Rysunek 32. Wykres słupkowy 3D po wstępnej konfiguracji. Źródło: opracowanie własne.

Kolory elementów na wykresie również mogą zostać dostosowane. Dla przykładu, jeśli podczas analizy danych najbardziej interesujące są dane dla kraju *Stany Zjednoczone*, zmianę koloru słupków można przeprowadzić w zakładce *Colors*. Rysunek 33 przedstawia możliwe elementy, dla których można nadpisać kolor.



Rysunek 33. Konfiguracja koloru elementów na wykresie słupkowym 3D. Źródło: opracowanie własne.

Zakładka pozwala na zmianę koloru dla następujących elementów:

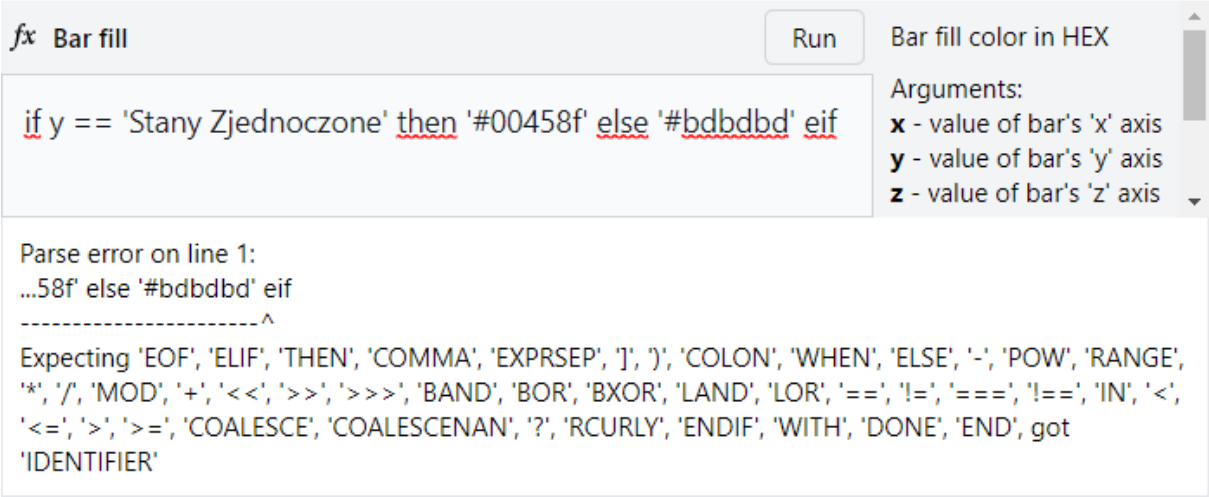
- wypełnienie słupka,
- obramowanie słupka,
- osie współrzędnych,
- siatka,
- tło.

Aby zmienić kolor wypełnienia słupka tylko dla wartości należących do *Stanów Zjednoczonych*, zgodnie z opisem zawartym w podrozdziale 5.4.3, należy napisać podprogram, który zwróci odpowiedni kolor względem wartości osi *Y* (Kraj). Kod podprogramu pokazano w listingu 16. Dla wartości *Y* = *Stany Zjednoczone* wybrano kolor ciemnoniebieski, a w każdym innym przypadku kolor jasnoszary.

Listing 16. Podprogram zmieniający kolor dla wartości Stanów Zjednoczonych. Źródło: opracowanie własne.

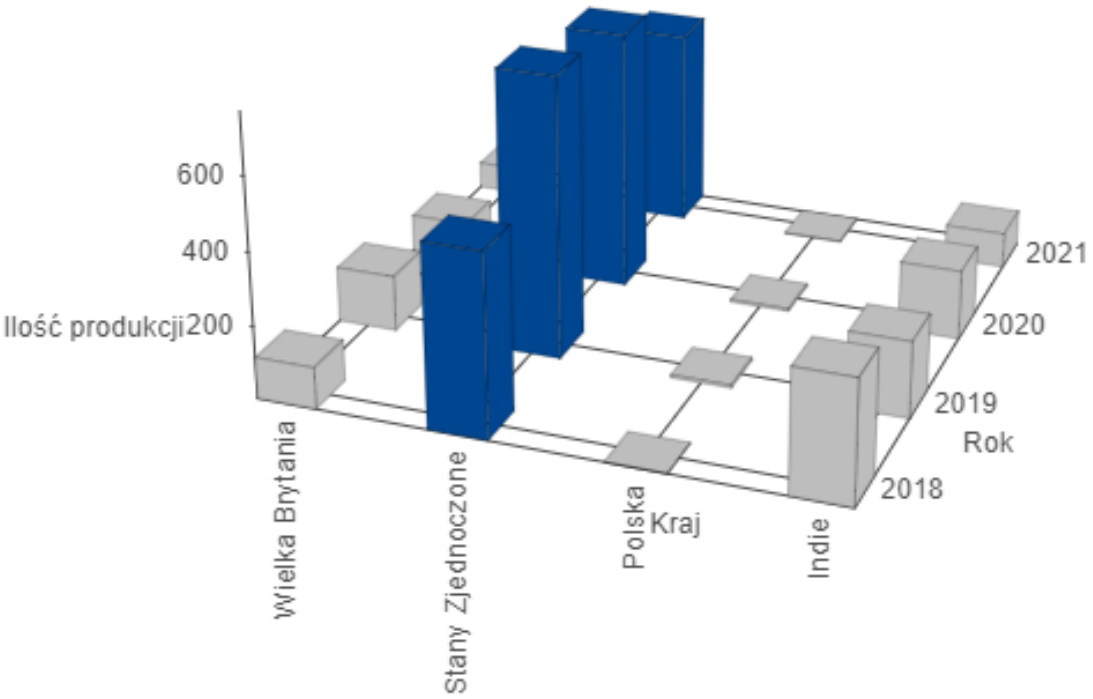
```
if y == 'Stany Zjednoczone' then '#00458f' else '#bdbdbd' endif
```

Przed wygenerowaniem wizualizacji, warto sprawdzić poprawność napisanego kodu uruchamiając go przyciskiem *Run* widocznym na rysunku 33. W przypadku błędu, system wskaże nam potencjalne miejsce pomyłki. W celu przedstawienia opisanej sytuacji, celowo dokonano literówkę – słowo zamykania warunku *endif* zamieniono na *eif*, a następnie uruchomiono podprogram. Wynik operacji pokazano na rysunku 34.



Rysunek 34. Uruchomienie błędnie napisanego podprogramu. Źródło: opracowanie własne.

Po wpisaniu podprogramu z listingu *16* ponownie wygenerowano podgląd i umieszczono na rysunku 35.

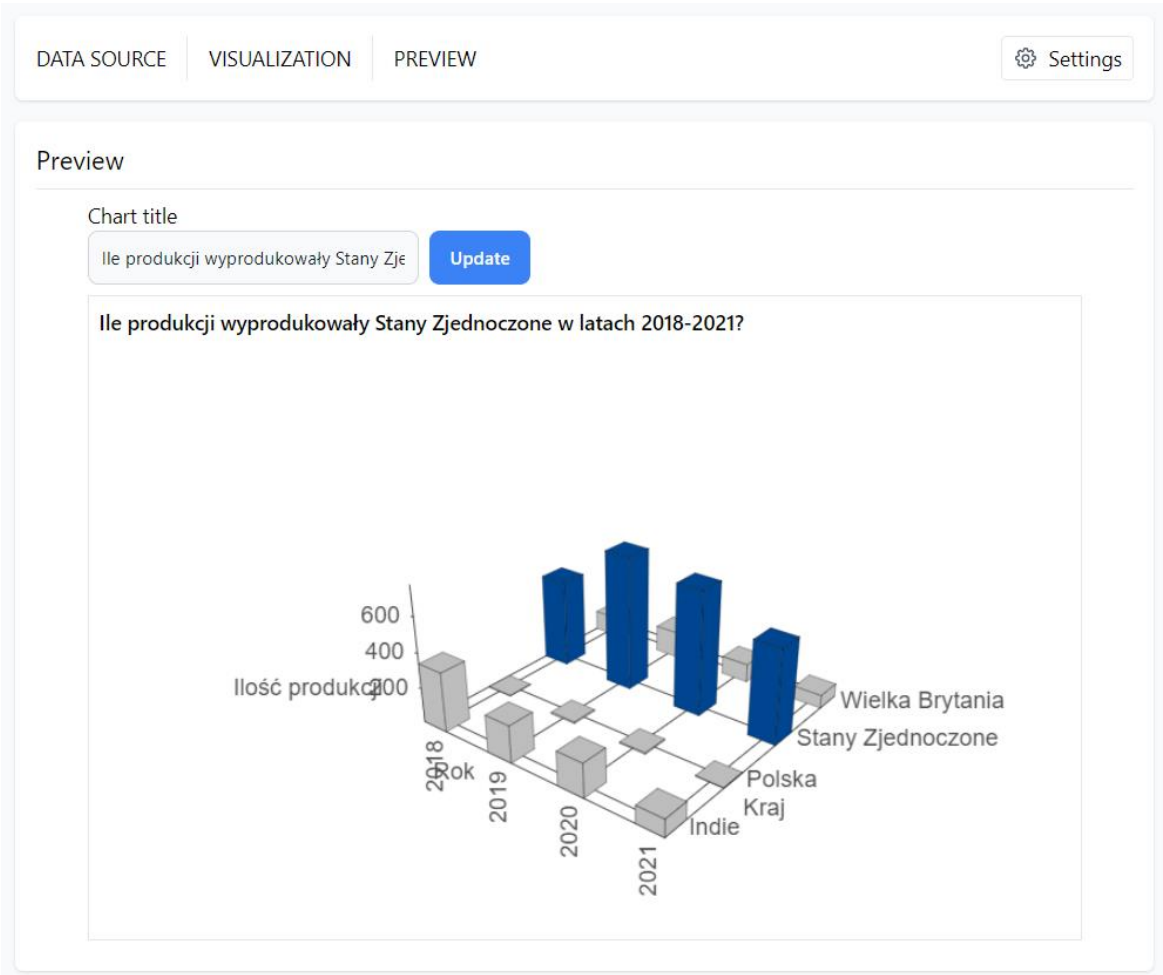


Rysunek 35. Wykres słupkowy 3D po docelowej konfiguracji. Źródło: opracowanie własne.

Aby zobaczyć dokładne wartości każdej osi poszczególnych słupków, należy najechać na badany słupek kursorem. Spowoduje to wyświetlenie się okienka (rysunek 36) z wypisanymi wartościami dla każdej osi współrzędnych.

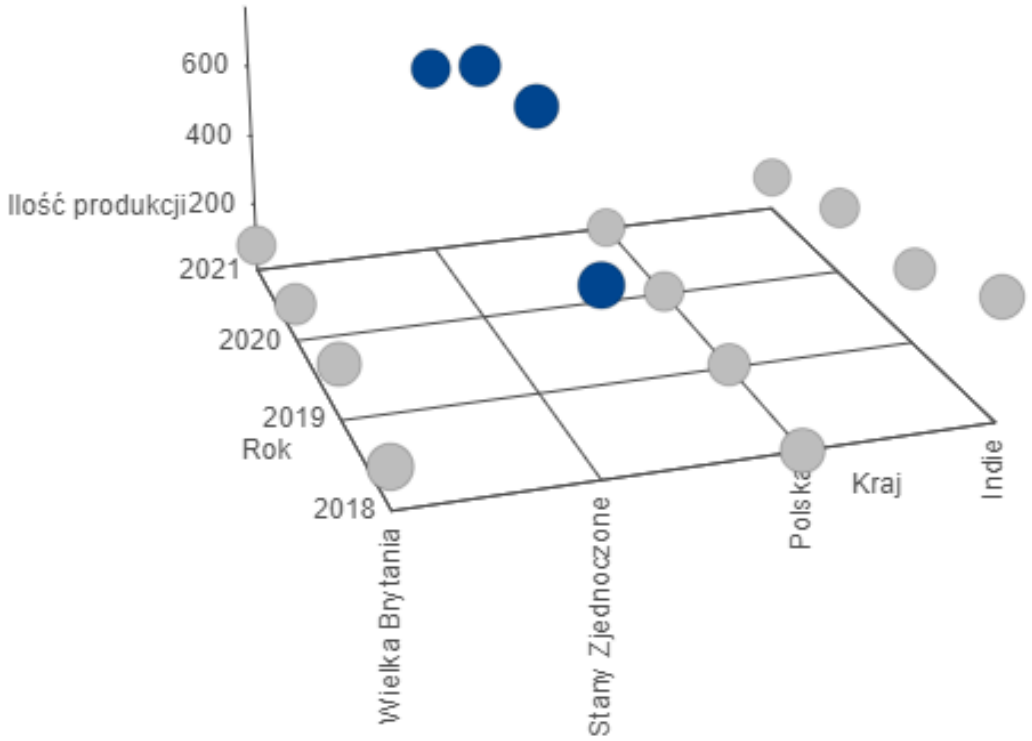
Rysunek 36. Okno informacyjne po najechaniu kursorem na słupek. Źródło: opracowanie własne.

Nadanie tytułu wykresu oraz podejrzenie jak będzie wyglądał wykres po upublicznieniu należy wykonać w zakładce *Preview* (rysunek 37).

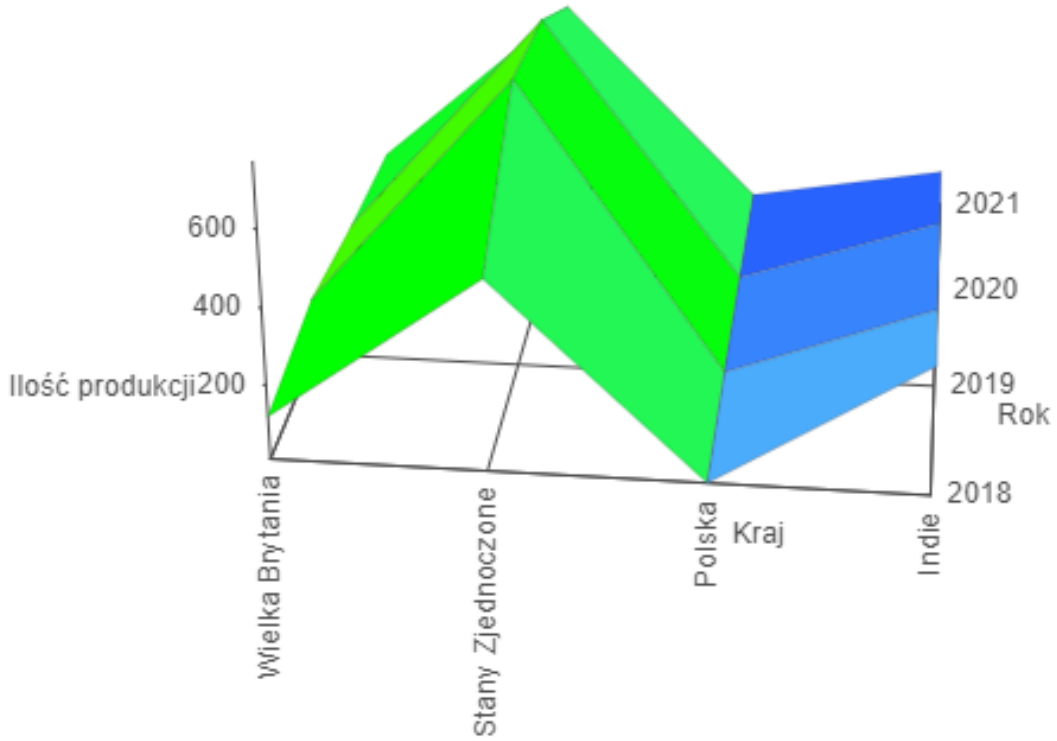


Rysunek 37. Podgląd strony wykresu po publikacji. Źródło: opracowanie własne.

Opisaną konfiguracje zastosowano również dla wtyczek *3D Point Graph* oraz *3D Surphace Graph* a wygenerowane wizualizacje zostały przedstawione odpowiednio na rysunkach 38 oraz 39.

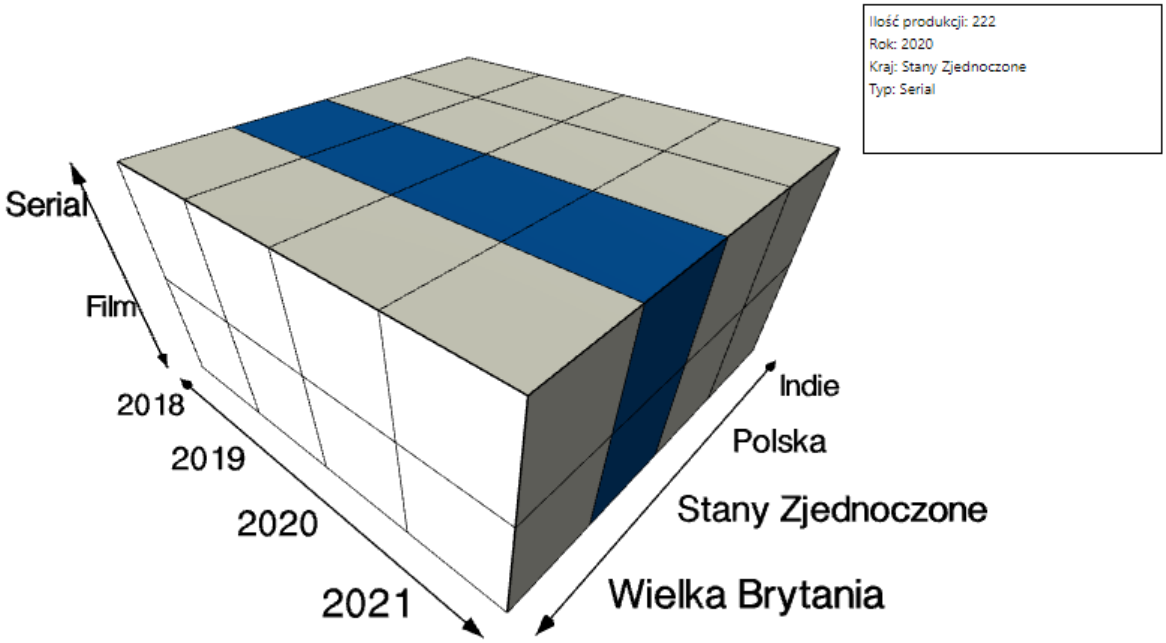


Rysunek 38. Wykres punktowy 3D. Źródło: opracowanie własne.



Rysunek 39. Wykres płaszczyzny 3D. Źródło: opracowanie własne.

Dla kostki OLAP dodano dodatkowy wymiar typu produkcji (film lub serial), a wygenerowany wykres przedstawiono na rysunku 40.

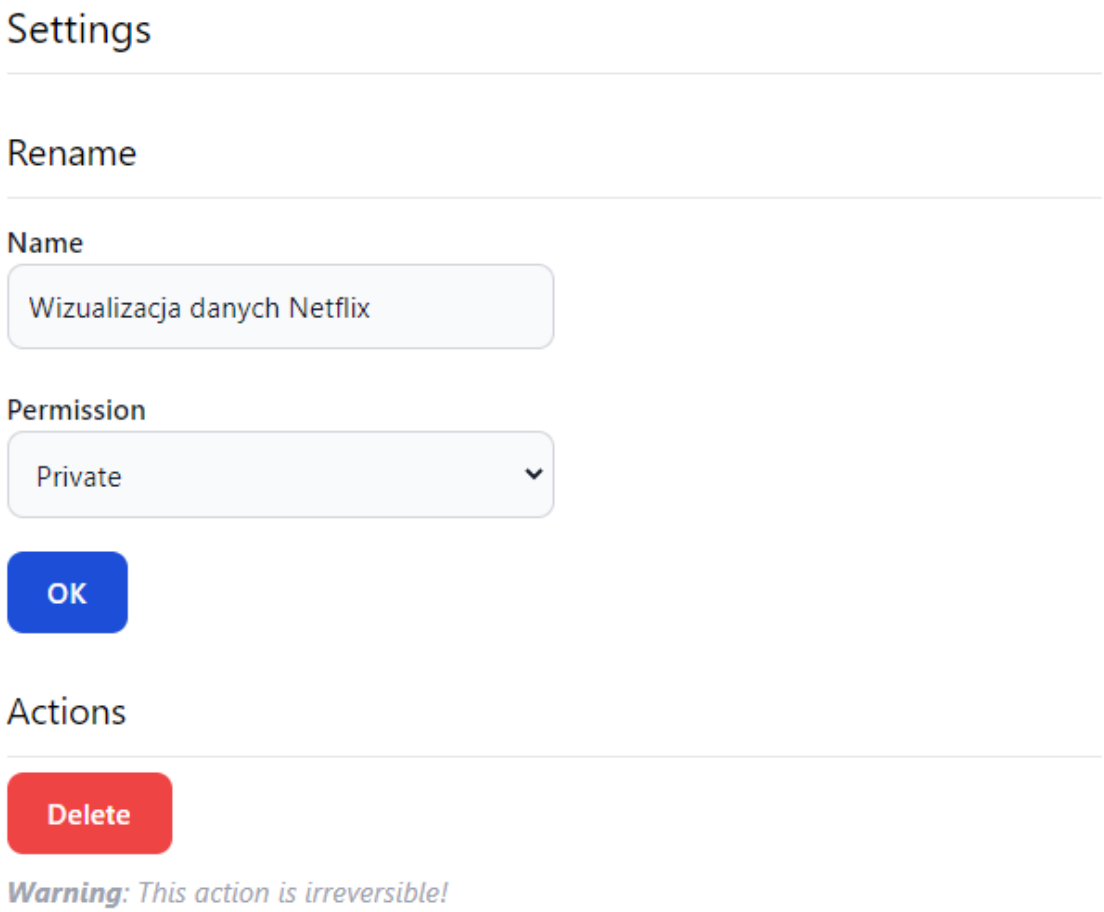


Rysunek 40. Trójwymiarowy wykres kostki OLAP. Źródło opracowanie własne.

6.4.3 Publikacja wizualizacji

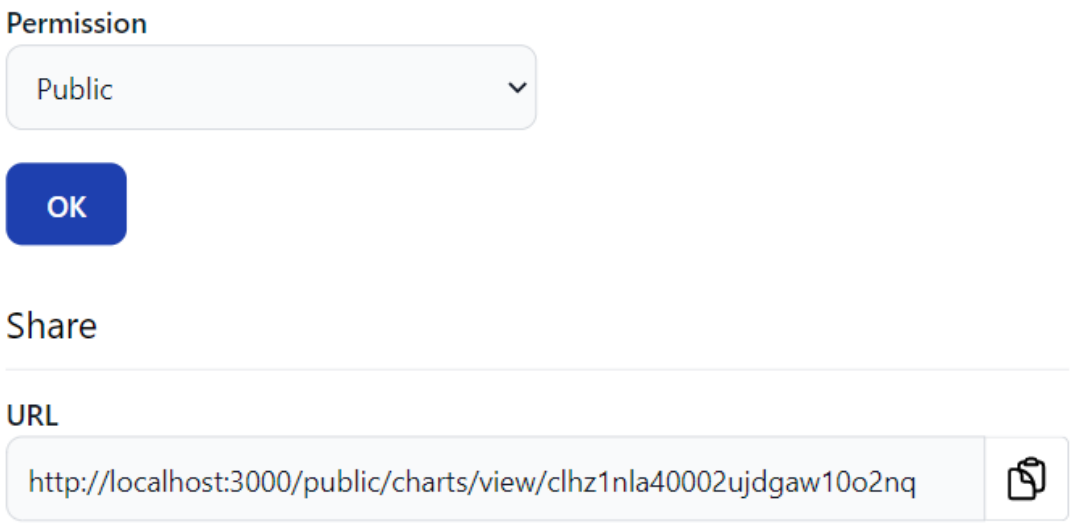
W celu publikacji wizualizacji należy przejść do ustawień projektu naciskając przycisk *Settings* z rysunku 15. Na tej podstronie (rysunek 41) możliwe jest wykonanie następujący operacji na projekcie:

- zmiana nazwy,
- usuwanie,
- publikacja.



Rysunek 41. Ustawienia projektu. Źródło: opracowanie własne.

Wybranie z listy wyboru *Permission* pozycji *Public* oraz zatwierdzenie przyciskiem *OK* spowoduje wyświetlenia się pola *Share* (rysunek 42) z publicznym adresem URL, pod którym zostanie wyświetlona strona zawierająca końcowy efekt skonfigurowanej wizualizacji.



Rysunek 42. Generowanie publicznego adresu URL wizualizacji. Źródło: opracowanie własne.

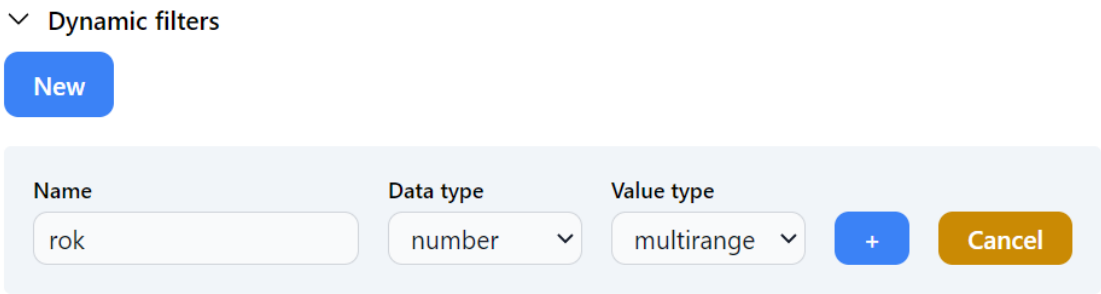
6.5. Dynamiczne filtrowanie

Przykładem w tym podrozdziale będzie stworzenie dynamicznych filtrów, które pozwolą użytkownikowi na zmianę zakresu lat oraz listy krajów z jakich rysowana jest wizualizacja.

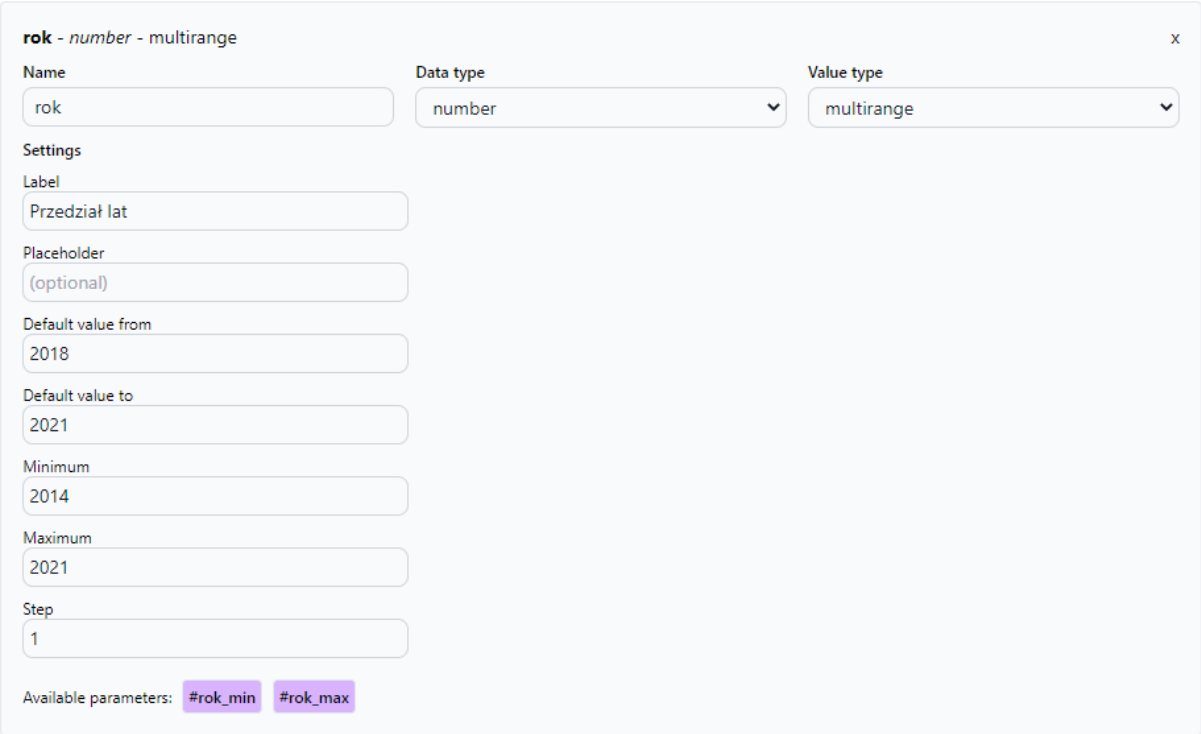
W aktualnej konfiguracji, są wpisane stałe wartości przedziału lat 2018 oraz 2021. Zadaniem dynamicznego filtru jest pozwolenie użytkownikowi na dowolną zmianę tego przedziału.

Stworzenie dynamicznego filtru odbywa się w kroku konfiguracji źródła danych (podpunkt 6.4.1). W sekcji *Dynamic filters* dodany został numeryczny filtr o nazwie *rok* o typie *multirange* (rysunek 43). Konfiguracja (rysunek 44) tego typu filtru została opisana poniżej:

- *Label* – tekst wyświetlany w wizualizacji,
- *Default value from* – początkowa wartość domyślna zakresu,
- *Default value to* – końcowa wartość domyślna zakresu,
- *Minimum* – dolna granica zakresu,
- *Maximum* – górna granica zakresu,
- *Step* – przyrost między każdą liczbą w zakresie.

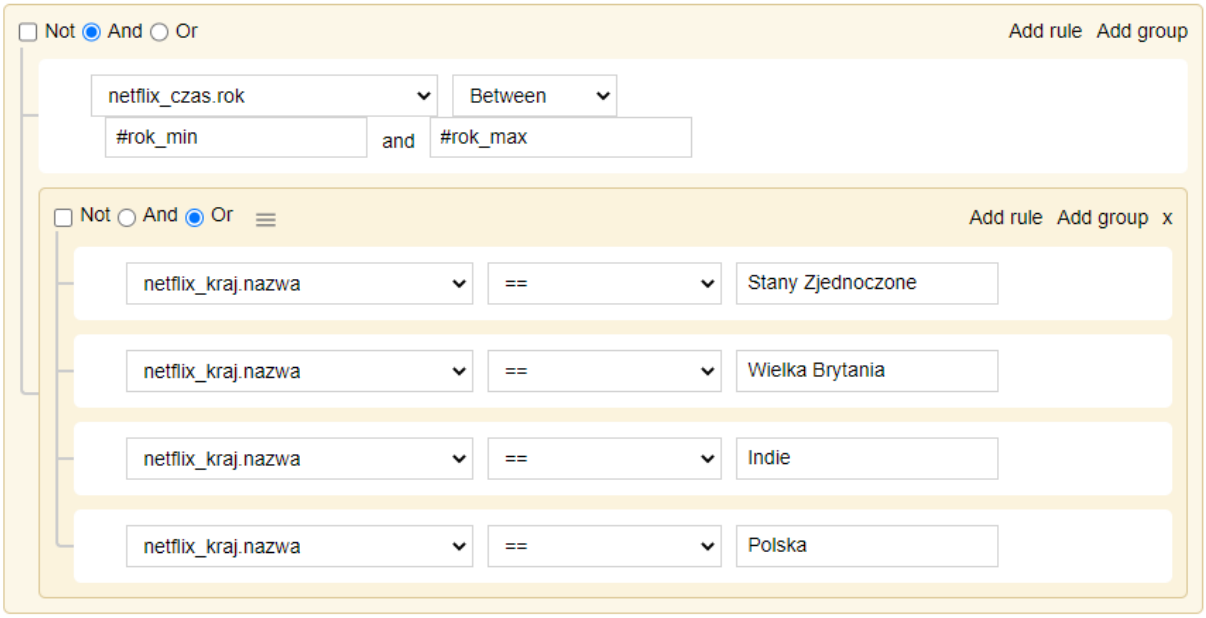


Rysunek 43. Dodawanie dynamicznego filtru. Źródło: opracowanie własne.



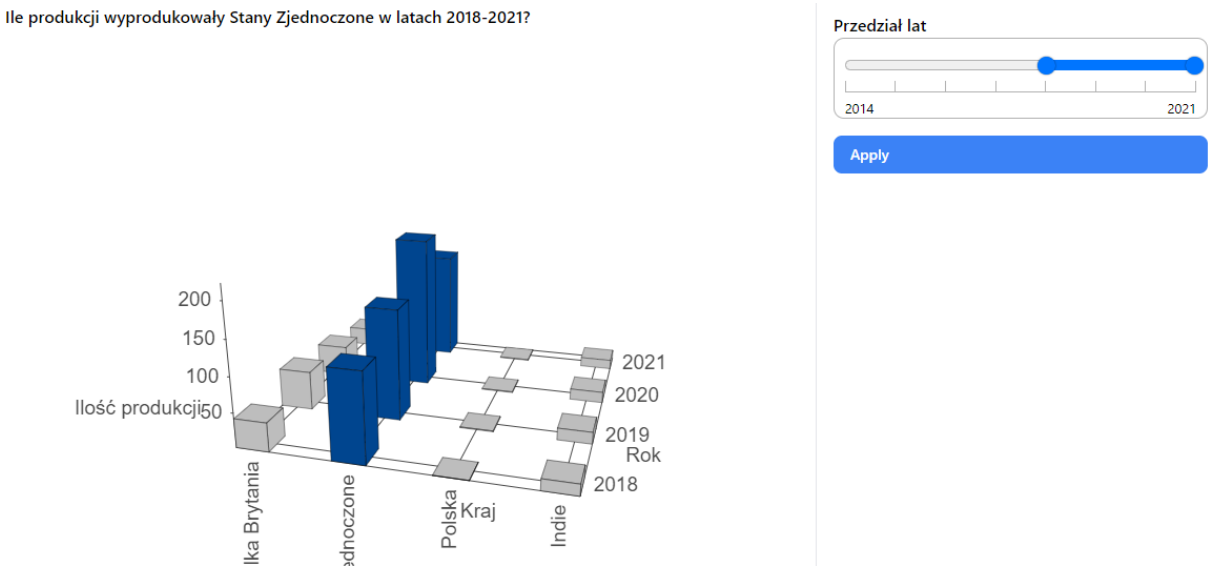
Rysunek 44. Konfiguracja dynamicznego filtru *multirange*. Źródło: opracowanie własne.

Widoczne dostępne parametry na rysunku 44 *Available parameters* wpisano w miejsce wcześniejszych wartości stałych (rysunek 45), gdzie *#rok_min* wskazuje na dolną wartość przedziału, a *#rok_max* na górną.



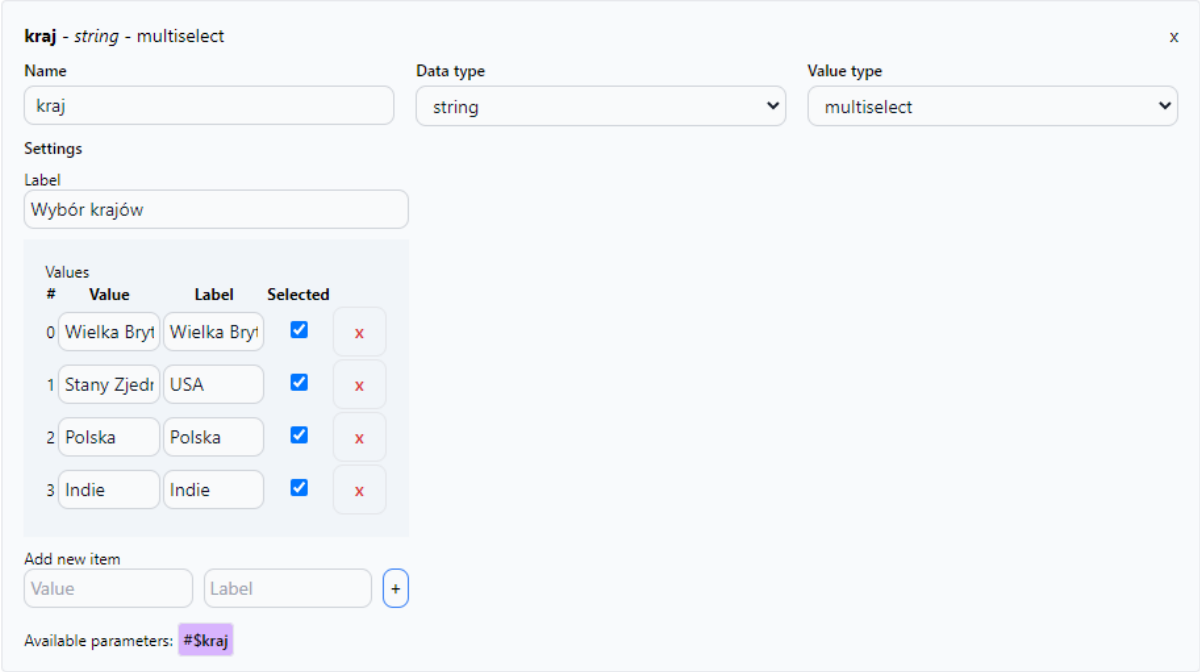
Rysunek 45. Filtrowanie danych źródłowych z użyciem dynamicznego filtru *multirange*. Źródło: opracowanie własne.

Wykonanie powyższych operacji spowoduje pojawienie się panelu z dynamicznymi filtrami na stronie wizualizacji, który został pokazany na rysunku 46. Zmiana wartości na suwaku i kliknięcie przycisku *Apply* spowoduje pobranie danych u źródła z aktualnie wybranymi przez użytkownika wartościami.

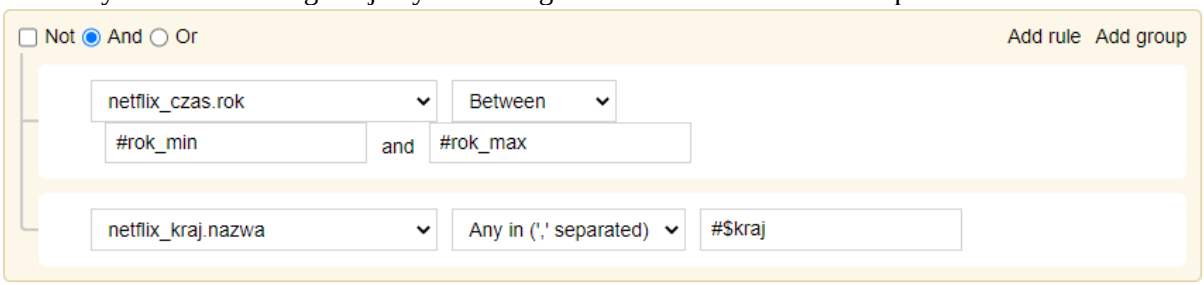


Rysunek 46. Panel dynamicznych filtrów na stronie wizualizacji. Źródło: opracowanie własne.

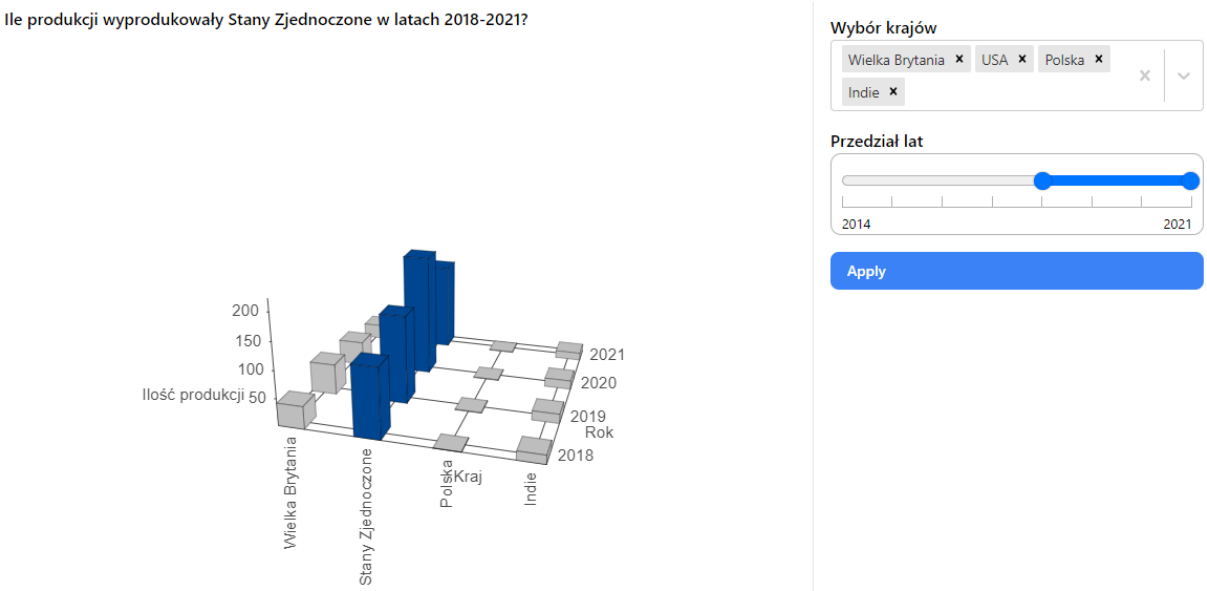
W analogiczny sposób utworzono dynamiczny filtr do wyboru krajów (rysunek 47 i 48) i tym samym zakończono konfigurację przykładu wizualizacji wielowymiarowych danych. Docelowa wizualizacja została przedstawiona na rysunku 49.



Rysunek 47. Konfiguracja dynamicznego filtru *multiselect*. Źródło: opracowanie własne.



Rysunek 48. Filtrowanie danych z użyciem dynamicznego filtru *multiselect*. Źródło: opracowanie własne.



Rysunek 49. Wygląd docelowej konfiguracji wizualizacji wielowymiarowych danych. Źródło: orpacowanie własne.

7. Podsumowanie

W niniejszym rozdziale opisane zostało podsumowanie zawierające wady, zalety oraz dalsze kierunki rozwoju zaproponowanego rozwiązania.

7.1. Wady i zalety rozwiązania

Prototyp spełnił stawiane mu wymagania, aczkolwiek etap implementacji ujawnił niektóre jego wady.

Wady:

- Wtyczki często dzielą funkcjonalność, co prowadzi do redundancji kodu spowodowanej potrzebą wdrożenia tych samych funkcji dla wielu pakietów;
- Interfejs nie posiada funkcjonalności zmiany języka – całość jest opisana w języku angielskim;
- Dostęp do systemu można uzyskać tylko poprzez zalogowanie się kontem Google;
- Interfejs nie jest zakodowany responsywnie, co powoduje utrudnioną konfigurację wizualizacji na urządzeniach mobilnych;
- Poprawna konfiguracja integracji źródła danych nie jest prosta bez wcześniejszego zapoznania się z systemem;
- Prototyp nie pozwala na zmianę wyglądu upublicznionej strony z wizualizacją;
- W systemie nie wdrożono funkcjonalności wspierających pracę na projektach, np. grupowanie, sortowanie, udostępnianie dla innych użytkowników itd.

Zalety:

- System jest generyczny i pozwala na łatwą implementację nowych wtyczek źródeł danych oraz wizualizacji;
- Wbudowany mechanizm podprogramów pozwala na wiele możliwości prezentacji wizualizowanych danych;
- Prototyp nie jest ograniczony do jednej biblioteki wizualizacji;
- Wizualizacje pozwalają na obrót, przybliżanie oraz oddalanie kamery;
- Dynamiczne filtrowanie danych w czasie rzeczywistym eliminuje potrzebę tworzenia tabel dla każdej wizualizacji z osobna u źródła;
- Brak potrzeby znajomości składni języka zapytań SQL;
- Modułowość pozwala na łatwe utrzymanie oraz rozwijanie oprogramowania;
- Aplikacja została wdrożona w oparciu o nowoczesne technologie;
- Zaprojektowany prototyp w technologiach webowych pozwala na skorzystanie z rozwiązania na każdym urządzeniu z dostępem do Internetu;

- Wtyczki posiadają dostęp do komponentów gotowych do użycia, co skraca czas tworzenia programowania;
- Prototyp udostępnia dla wtyczek wbudowane metody komunikacji pomiędzy warstwami klient – serwer.

7.2. Potencjalne kierunki rozwoju

Możliwe kierunki rozwoju zaproponowanej aplikacji opisano na liście poniżej:

- Integracja z większą ilością dostawców logowania pozwoliłaby użytkownikowi na łatwiejszy dostęp do wszystkich funkcjonalności aplikacji;
- Zaprojektowanie responsywnej wersji interfejsu użytkownika ułatwiłoby obsługę aplikacji na urządzeniach mobilnych;
- Przetłumaczenie aplikacji na wiele języków zwiększyłoby dostępność i ułatwiłoby poruszanie się po systemie;
- Możliwość grupowania oraz sortowania projektów wprowadziłoby lepszą organizację, co przełożyłoby się na swobodniejsze przeglądanie stworzonych wizualizacji;
- Udostępnianie konfiguracji wizualizacji dla innych użytkowników umożliwiłoby pracę zespołową;
- Możliwość podejrzenia zaprojektowanych zapytań bazodanowych oraz ewentualne ich ręcznie dostosowanie;
- Zwiększenie możliwości integracji z zewnętrznymi źródłami danych, na przykład, z bazami grafowymi lub wektorowymi;
- Rozszerzenie listy możliwych do wyboru wizualizacji wielowymiarowych danych;
- Możliwość dostosowania początkowej pozycji oraz ograniczeń kamery na wizualizacjach 3D;
- Funkcjonalność zmiany wyglądu udostępnionej strony wizualizacji pozwoliłaby, dla przykładu, na wpasowanie się w estetykę prezentacji marketingowych, czy analitycznych;
- Wprowadzenie historii operacji dokonanych na projekcie umożliwiłoby wykrycie ewentualnych błędów konfiguracji;
- Szyfrowanie zapewniłoby bezpieczeństwo wartości konfiguracyjnych, takich jak, wrażliwe dane logowania do bazy danych;
- Dwuetapowa weryfikacja logowania zmniejszyłaby ryzyko utraty dostępu do konta.

7.3. Wnioski

Praca przedstawia analizę istniejących rozwiązań do wizualizacji wielowymiarowych danych oraz wyszczególnia ich mocne i słabe strony. Na ich podstawie, postawiono wymagania oraz zaprojektowano aplikację, która stałaby się konkurencyjnym rozwiązaniem, a następnie

zaimplementowano działający prototyp. Aplikacja cechuje się generycznością, która pozwala na łatwy rozwój oraz utrzymanie. Największą zaletą w porównaniu do konkurencji jest możliwość dynamicznego filtrowania danych źródłowych bez znajomości bazodanowych języków zapytań. Implementacja rozwiązania w technologiach webowych eliminuje potrzebę pobierania zewnętrznych programów i maksymalnie zwiększa dostępność jej użycia.

Autor niniejszej pracy uważa, że wdrożony prototyp spełnia postawione mu wymagania i już w aktualnej formie mógłby konkurować z aktualnie dostępnymi rozwiązaniami na rynku.

Bibliografia

[1] Why do companies collect data?, Indeed Editorial Team, https://ca.indeed.com/career-advice/career-development/collect-data, dostęp 2023-06-05.

[2] Chart Studio, https://chart-studio.plotly.com/, dostęp 2023-06-05.

[3] Looker Studio, https://lookerstudio.google.com/, dostęp 2023-06-05.

[4] ChartBlocks, https://app.chartblocks.io/, dostęp 2023-06-05.

[5] Live Editor AMCharts, https://live.amcharts.com/, dostęp 2023-06-05.

[6] Longer Biography of Tim Berners Lee, https://www.w3.org/People/Berners-Lee/Longer.html, dostęp 2023-06-05.

[7] A brief history of CSS until 2016, Bert Bos, https://www.w3.org/Style/CSS20/history.html, dostęp 2023-06-05.

[8] Tailwind CSS, Dokumentacja, https://tailwindcss.com/docs, dostęp 2023-06-05.

[9] 1995: The Brith of Javascript, https://webdevelopmenthistory.com/1995-the-birth-of-javascript/, dostęp 2023-06-05.

[10] jQuery, https://jquery.com/, dostęp 2023-06-05.

[11] Angular, https://angular.io/, dostęp 2023-06-05.

[12] React, https://react.dev/, dostęp 2023-06-05.

[13] Typescript, https://www.typescriptlang.org/, dostęp 2023-06-05.

[14] JSX, https://www.typescriptlang.org/docs/handbook/jsx.html, dostęp 2023-06-05.

[15] What is React.js Used For: Top 7 Examples, https://www.monterail.com/blog/what-is-react-used-for, dostęp 2023-06-05.

[16] React Query Builder, https://react-querybuilder.js.org/, dostęp 2023-06-05.

[17] Three.js, https://threejs.org/, dostęp 2023-06-05.

[18] Vis.js, https://visjs.org/, dostęp 2023-06-05.

[19] Lexpjs, https://github.com/toggledbits/lexpjs, dostęp 2023-06-05.

[20] Node.js, https://nodejs.org/, dostęp 2023-06-05.

[21] The v8 JavaScript engine, https://nodejs.dev/en/learn/the-v8-javascript-engine/, dostęp 2023-06-05.

[22] Next.js, https://nextjs.org/, dostęp 2023-06-05.

[23] MySQL, https://www.mysql.com/, dostęp 2023-06-05.

[24] PostgreSQL, https://www.postgresql.org/, dostęp 2023-06-05.

[25] MongoDB, https://www.mongodb.com/, dostęp 2023-06-05.

[26] JSON, https://www.json.org/json-en.html, dostęp 2023-06-05.

[27] Prisma.js https://www.prisma.io/, dostęp 2023-06-05.

[28] GraphQL - https://graphql.org/, dostęp 2023-06-05.

[29] Lerna.js - https://lerna.js.org/, dostęp 2023-06-05.

[30] Dbdiagram.io, https:://dbdiagram.io, dostęp 2023-06-05.

[31] Netflix Movies and TV Shows, Shivam Bansal, https://www.kaggle.com/datasets/shivamb/netflix-shows, dostęp 2023-06-05.

[32] Google, https://accounts.google.com, dostęp 2023-06-05.

[33] CUID, https://github.com/paralleldrive/cuid2, dostęp 2023-06-05.

[34] Układ współrzędnych, http://www.fizykon.org/wlk_podstawowe/uklad_wspolrzednych_wstep.htm, dostęp 2023-06-05.

Spis tabel

Tabela 1. Zestawienie najważniejszych cech wymienionych narzędzi. Źródło: opracowanie własne. .11
Tabela 2. Opis argumentów wymaganych do rejestracji wtyczki wizualizacji. Źródło: opracowanie własne 28
Tabela 3. Opis argumentów wymaganych do rejestracji wtyczki integracji z zewnętrznym źródłem danych (klient). Źródło: opracowanie własne.....29
Tabela 4. Opis argumentów wymaganych do rejestracji wtyczki integracji z zewnętrznym źródłem danych (serwer). Źródło: opracowanie własne 30
Tabela 5. Opis kolumn listy projektów. Źródło: opracowanie własne. 38

Spis rysunków

Rysunek 1. Interfejs użytkownika Chart Studio od plotly.com. Źródło: [2] 7
Rysunek 2. Interfejs użytkownika Looker Studio od Google. Źródło: [3].....8
Rysunek 3. Interfejs użytkownika ChartBlocks. Źródło: [4] 9
Rysunek 4. Interfejs użytkownika Live Editor AMCharts. Źródło: [5] 10
Rysunek 5. Przykładowy wykres ilości sprzedaży względem lat. Źródło: opracowanie własne.....13
Rysunek 6. Przykładowa lista wyboru kraju, jako dodatkowy filtr. Źródło: opracowanie własne.....13
Rysunek 7. Schemat architektury rozwiązania. Źródło: opracowanie własne 22
Rysunek 8. Schemat bazy danych. Źródło: [30] 23
Rysunek 9. Schemat bazy danych Netflix. Źródło: opracowanie własne.....35
Rysunek 10. Ekran logowania do aplikacji. Źródło: opracowanie własne.....36
Rysunek 11. Logowanie do konta Google. Źródło: [32].....36
Rysunek 12. Ekran startowy panelu głównego. Źródło: opracowanie własne. 37
Rysunek 13. Tworzenie wykresu. Źródło: Opracowanie własne. 38
Rysunek 14. Lista zapisanych projektów wizualizacji. Źródło opracowanie własne. 38
Rysunek 15. Widok konfiguracji projektu. Źródło: opracowanie własne. 39
Rysunek 16. Wybór integracji z zewnętrznym źródłem danych. Źródło: opracowanie własne. 39
Rysunek 17. Konfiguracja danych uwierzytelniających. Źródło: opracowanie własne. 40
Rysunek 18. Powiadomienie z informacją zwrotną dot. połączenia. Źródło: opracowanie własne. 40
Rysunek 19. Wybór tabeli wejściowej. Źródło: opracowanie własne.....40
Rysunek 20. Łączenie tabel wymiarowych. Źródło: opracowanie własne.....41
Rysunek 21. Filtrowanie danych źródłowych. Źródło: opracowanie własne. 41
Rysunek 22. Wybór kolumn danych źródłowych. Źródło: opracowanie własne. 42
Rysunek 23. Agregacja danych źródłowych. Źródło: opracowanie własne. 42
Rysunek 24. Limit zwracanych danych źródłowych. Źródło: opracowanie własne.....42
Rysunek 25. Podgląd danych źródłowych po konfiguracji. Źródło: opracowanie własne. 43
Rysunek 26. Wybór wizualizacji. Źródło: opracowanie własne. 44
Rysunek 27. Kartezjański układ współrzędnych. Źródło: [34] 44
Rysunek 28. Przydzielanie kolumn do układu współrzędnych. Źródło: opracowanie własne 44
Rysunek 29. Domyślny wygląd wykresu słupkowego 3D bez konfiguracji. Źródło: opracowanie własne. 45
Rysunek 30. Nadpisanie etykiet przy osiach współrzędnych. Źródło: opracowanie własne. 45
Rysunek 31. Domyślne wartości rozmiarów elementów wykresu słupkowego 3D. Źródło: opracowanie własne. 46
Rysunek 32. Wykres słupkowy 3D po wstępnej konfiguracji. Źródło: opracowanie własne.....46
Rysunek 33. Konfiguracja koloru elementów na wykresie słupkowym 3D. Źródło: opracowanie własne. 47
Rysunek 34. Uruchomienie błędnie napisanego podprogramu. Źródło: opracowanie własne. 48
Rysunek 35. Wykres słupkowy 3D po docelowej konfiguracji. Źródło: opracowanie własne. 48
Rysunek 36. Okno informacyjne po najechaniu kursorem na słupek. Źródło: opracowanie własne.....49
Rysunek 37. Podgląd strony wykresu po publikacji. Źródło: opracowanie własne. 49
Rysunek 38. Wykres punktowy 3D. Źródło: opracowanie własne. 50
Rysunek 39. Wykres płaszczyzny 3D. Źródło: opracowanie własne.....50
Rysunek 40. Trójwymiarowy wykres kostki OLAP. Źródło opracowanie własne. 51
Rysunek 41. Ustawienia projektu. Źródło: opracowanie własne. 52
Rysunek 42. Generowanie publicznego adresu URL wizualizacji. Źródło: opracowanie własne.....52
Rysunek 43. Dodawanie dynamicznego filtru. Źródło: opracowanie własne. 53
Rysunek 44. Konfiguracja dynamicznego filtru *multirange*. Źródło: opracowanie własne. 53
Rysunek 45. Filtrowanie danych źródłowych z użyciem dynamicznego filtru *multirange*. Źródło: opracowanie własne. 54
Rysunek 46. Panel dynamicznych filtrów na stronie wizualizacji. Źródło: opracowanie własne. 54

Rysunek 47. Konfiguracja dynamicznego filtru *multiselect*. Źródło: opracowanie własne. 55
Rysunek 48. Filtrowanie danych z użyciem dynamicznego filtru *multiselect*. Źródło: opracowanie własne. 55
Rysunek 49. Wygląd docelowej konfiguracji wizualizacji wielowymiarowych danych. Źródło: orpacowanie własne. 56

Spis listingów

Listing 1. Przykładowy akapit opisany w HTML. Źródło: opracowanie własne. 15
Listing 2. Przykładowe użycie języka CSS. Źródło: opracowanie własne.....16
Listing 3. Konfiguracja schematu dla tabeli Chart. Źródło: opracowanie własne.....24
Listing 4. Konfiguracja połączenia do bazy danych. Źródło: opracowanie własne 24
Listing 5. Komendy wiersza poleceń *prisma*. Źródło: opracowanie własne.....24
Listing 6. Przykład wyświetlenia wpisów z tabeli *Chart* w konsoli. Źródło: opracowanie własne.....25
Listing 7. Przykład użycia *React Context* w celu przekazania konfiguracji do wtyczki. Źródło: opracowanie własne 25
Listing 8. Konfiguracja pliku *package.json* dla serwerowej wtyczki MySQL. Źródło: opracowanie własne. 27
Listing 9. Rejestracja wtyczki słupkowego wykresu 3D. Źródło: opracowanie własne.....29
Listing 10. Rejestracja wtyczki MySQL w warstwie klienta. Źródło: opracowanie własne 29
Listing 11. Rejestracja serwerowej części wtyczki MySQL. Źródło: opracowanie własne 30
Listing 12. Zawartość pliku *plugins.json* z listą wtyczek. Źródło: opracowanie własne.....31
Listing 13. Przykładowy program dynamicznego równania dla koloru słupków na wykresie. Źródło: opracowanie własne 31
Listing 14. Wyszukiwanie i wczytywanie wtyczek podczas uruchamiania aplikacji. Źródło: opracowanie własne 32
Listing 15. Uproszczony kod źródłowy osadzania wtyczki wizualizacji. Źródło: opracowanie własne33
Listing 16. Podprogram zmieniający kolor dla wartości Stanów Zjednoczonych. Źródło: opracowanie własne. 47