

Grzegorz Bujak
Number albumu: 088943

**Opracowanie i implementacja systemu Headless CMS z
dodatkową możliwością modelowania warstwy danych przy
pomocy natywnego SQLa**

**Praca dyplomowa
na studiach I-go stopnia
na kierunku Informatyka**

Opiekun pracy dyplomowej:
dr inż. Mariusz Jacek Wiśniewski
Katedra Systemów Informatycznych

Konsultant pracy dyplomowej:
dr inż. Karol Wieczorek

Kielce, 2021

tutaj oryginal

tutaj oświadczenie

Opracowanie i implementacja systemu Headless CMS z dodatkową możliwością modelowania warstwy danych przy pomocy natywnego SQLa

Streszczenie

Systemy CMS oraz Headless CMS mają tendencję do ograniczania możliwości operowania na danych do statycznych operacji zdefiniowanych przez system. Może to powodować trudności, gdy wymagane jest wykonanie skomplikowanych operacji na bazie danych. Ponadto, wysyłanie zapytań do bazy danych, z której korzysta system CMS jest często odradzane i niewspierane przez twórców systemu CMS. Celem pracy dyplomowej jest napisanie systemu Headless CMS, który nie ogranicza możliwości manipulacji bazą danych. Przygotowany system CMS osiąga ten cel, przez umożliwienie administratorom wykonywania natywnych zapytań SQL na bazie danych, z której korzysta system. Kolejną kluczową funkcją jest pozwolenie administratorom na definiowanie zapytań, które zostaną wykonane przy zapytaniu HTTP do systemu CMS, i których dane wynikowe zostaną zwrócone w odpowiedzi HTTP.

Słowa kluczowe: CMS, Headless CMS, bazy danych, SQL, zarządzanie treścią, warstwa danych

The development and the implementation of a Headless CMS system with support of modelling of the data layer by native SQL queries

Summary

CMS and Headless CMS systems have a tendency to limit the ability to operate on data to static operations defined by the system. It can cause difficulties when it's necessary to execute complex operations on the database. Furthermore, sending queries to the database used by the CMS system is often discouraged and not supported by the authors of the CMS system. The aim of the thesis is to prepare a Headless CMS system, which does not limit the ability of manipulating the database. The aim is achieved by allowing the administrators to execute native SQL queries on the database used by the system. Another key feature is to let administrators define queries which will be executed after an HTTP request to the CMS system and the resulting data will be returned in the HTTP response.

Keywords: CMS, Headless CMS, databases, SQL, content management, data layer

SPIS TREŚCI

1	WSTĘP	3
2	CEL I ZAKRES PRACY	3
2.1	Cel pracy	3
2.2	Zakres funkcji systemu	3
3	PRZEGLĄD ISTNIEJĄCYCH ROZWIĄZAŃ	4
3.1	Wordpress	4
3.2	Strapi	5
3.3	PostgREST	5
3.4	Podsumowanie	6
4	PROJEKT SYSTEMU	6
4.1	Projekt aplikacji frontend	6
4.2	Projekt aplikacji backend	7
5	IMPLEMENTACJA KLUCZOWYCH FUNKCJI	8
5.1	Tworzenie nowych typów danych	8
5.2	Zarządzanie danymi	8
6	test	10
6.1	test sub	10

1. WSTĘP

2. CEL I ZAKRES PRACY

2.1. Cel pracy

Celem pracy było przygotowanie systemu CMS. System powinien zawierać większość funkcji dostępnych w typowym systemie CMS, jak tworzenie nowych typów danych i manipulacja danymi znajdującymi się w systemie, jednak nie są to główne funkcje systemu. Głównym założeniem przy projektowaniu i implementacji było stworzenie systemu, który nie ogranicza możliwości operowania danymi przez administratora.

Główny cel został osiągnięty przez pozwolenie administratorowi na wykonywanie natywnych zapytań SQL na bazie danych. Zaimplementowano aplikacje frontend i backend, które razem umożliwiają administratorowi wygodne pisanie zapytań SQL i testowanie ich wpływu na bazę danych.

Nie mniej ważnym założeniem było umożliwienie administratorowi definiowania punktów końcowych, których wywołanie zapytaniem HTTP powoduje wykonanie zdefiniowanego przez administratora drzewa zapytań SQL. Przygotowany edytor pozwala na modelowanie struktury danych, które zostaną zwrócone w odpowiedzi HTTP.

2.2. Zakres funkcji systemu

Program przygotowany w ramach pracy można podzielić na dwie części: frontend i backend:

1. Frontend.

- Aplikacja SPA wykorzystująca bibliotekę react.js. Strona porozumiewa się z serwerem za pomocą API REST.
- Jest napisany w języku Typescript i wykorzystuje system typów w każdym miejscu, gdzie jest to możliwe.
- Strona jest responsywna dzięki zastosowanym nowoczesnym opcjom CSS - grid i flexbox. Strona korzysta ze zmiennych CSS. Zmienne CSS umożliwiły łatwą implementację funkcji zmiany motywu aplikacji przez użytkownika.
- Implementuje ważniejsze funkcje typowego panelu administratora systemu zarządzania treścią: tworzenie nowych tabel, zarządzanie danymi w tabeli.
- Implementuje zaawansowany edytor SQL z podświetlaniem składni, funkcją zmiany kolejności zapytań, funkcją tymczasowego wyłączenia zapytania. Wyświetla wyniki każdego zapytania pod kodem zapytania. Pozwala pisać zapytanie testowe, którego wynik zostanie pobrany przed wykonaniem listy zapytań jak i po wykonaniu. Pozwala na wygenerowanie diagramu ER przed i po wykonaniu listy zapytań. Pozwala na testowanie listy zapytań z pomocą transakcji, która zostanie wycofana przed zwrotem danych.
- Implementuje edytor punktów końcowych pozwalający na pisanie złożonych drzew zapytań, w których zapytania podrzędne mają dostęp do danych wynikających z wykonania zapytań nadrzędnych. Administrator może testować punkt końcowy przed wdrożeniem z wykorzystaniem edytora danych testowych. Administrator może ograniczać możliwość wywołania punktu końcowego do listy grup użytkowników.
- Implementuje interfejs tworzenia użytkowników.

2. Backend.

- Backend aplikacji został napisany w języku rust. Jest asynchroniczny, co pozwala na obsługiwanie wielu zapytań na mniejszej ilości wątków. Wykorzystuje asynchroniczny runtime Hyper.
- Współpracuje z bazą danych PostgreSQL. Używa tabel zawierających dane o tabelach w bazie danych, przez co nie musi sam przechowywać metadanych o tabelach.
- Korzysta z transakcji. Są używane do testowania pisanych przez administratora zapytań SQL.
- Implementuje generowanie diagramu ER w składni mermaid.js z danych pobranych z tabel zawierających metadane o tabelach w bazie danych.
- Implementuje bezpieczne wykonywanie drzewa zapytań SQL z użyciem niezaufanych danych z przychodzącego zapytania HTTP. Zapytania podrzędne mają dostęp do zapytań nadrzędnych. Każde wykorzystanie danych w zapytaniach jest zabezpieczone przed atakami SQL injection.
- Implementuje system użytkowników z wykorzystaniem technologii json web token. Bezpiecznie przechowuje hasła użytkowników za pomocą algorytmu Argon2.

3. PRZEGLĄD ISTNIEJĄCYCH ROZWIĄZAŃ

3.1. Wordpress

Pierwszym omawianym systemem zarządzania treścią jest Wordpress. Jest to najbardziej popularny CMS. Ten system został napisany z myślą o prezentowaniu treści w formie strony HTML. Wordpress jest napisany w języku PHP i do działania wymaga zainstalowania serwera HTTP. Pierwsze wydanie systemu miało miejsce w roku 2003 i od tamtej pory system jest ciągle rozwijany.

Wordpress pozwala na definiowanie własnych modeli, ale nie jest to wspierane w domyślnym panelu administratora. Do stworzenia nowego typu danych, wymagane jest wywołanie funkcji PHP udostępnionej przez Wordpress, do której nie ma domyślnie interfejsu użytkownika. Polecanym przez twórców sposobem tworzenia nowych typów danych jest zainstalowanie wtyczki, która implementuje taki interfejs [5].

Pisanie własnych zapytań SQL jest możliwe, ale podobnie jak definiowanie niestandardowych typów, domyślnie nie posiada interfejsu. W celu napisania własnego zapytania, trzeba to zrobić z poziomu PHP, lub zainstalować wtyczkę umożliwiającą pisanie własnych zapytań z panelu administratora.

W wersji 5.0 Wordpress, dodano nowy edytor “Gutenberg” pozwalający na tworzenie postów opartych o bloki. Jest to mniej skomplikowana alternatywa dla stosowanych do tej pory tzw. “online rich-text editor”. Edytory blokowe pozwalają na tworzenie wpisów składających się z bloków. Blok może zawierać tekst lub media. Bloki tekstowe mogą reprezentować nagłówek lub paragraf, a paragrafy mogą mieć fragmenty z ograniczonym stylizowaniem jak na przykład pogrubieniem czcionki. Wpisy w formie bloków są łatwiejsze w przechowywaniu i wyświetlaniu na stronie HTML. Nie jest konieczne na przykład parsowanie treści wpisów, żeby wydobyć informacje o zdjęciach, jakie zostały zamieszczone we wpisie.

Wpisy oparte o bloki mogą być wygodnie tworzone w dużo prostszych edytorach. Zmniejsza to potrzebę stosowania skomplikowanych systemów CMS do zarządzania treścią.

Podsumowując, Wordpress to oprogramowanie, które dobrze spełnia potrzeby autora bloga. Znaczną zaletą systemu Wordpress jest ekosystem wtyczek pozwalających na rozwiązanie większości problemów związanych z zarządzaniem danymi.

System Wordpress domyślnie znacznie ogranicza możliwości administratora. W celu wykonywania niestandardowych operacji na bazie danych, administrator musi polegać na wtyczkach, lub samemu napisać umożliwiające to wtyczki. Zwiększa to znacznie wymagania wiedzy wobec administratora.

Pomimo tego, możliwe jest skonfigurowanie systemu Wordpress w taki sposób, żeby spełniał wszystkie funkcje przygotowanego w ramach tej pracy systemu. Taka konfiguracja byłaby jednak mniej stabilna od systemu, który powstawał w celu spełnienia tych celów. Niezbędne byłoby poleganie na dużej ilości wtyczek, lub napisanie tych wtyczek samemu, co może być bardziej skomplikowane, niż napisanie systemu backend spełniającego potrzeby jednej strony internetowej.

(TODO: cite Wordpress bible)

3.2. Strapi

Strapi to system Headless CMS. Nie posiada on interfejsu użytkownika. Dane wprowadzane do systemu są pobierane za pomocą API REST. Domyślne API ma ograniczoną funkcjonalność. Pobieranie danych z CMS jest ograniczone do pobierania wszystkich wpisów danego typu, lub jednego wpisu, ale tylko po id.

Strapi umożliwia tworzenie własnych punktów końcowych. Administrator może ustawić ścieżkę i funkcję, która obsłuży zapytania na daną ścieżkę. Nie da się jednak zrobić tego w panelu administratora. Administrator chcąc stworzyć niestandardowy punkt końcowy musi sam napisać funkcję, które obsłużą zapytania w języku JavaScript. Jest to porównywalnie skomplikowane do napisania kontrolera w zwykłej aplikacji internetowej. Ponadto, na administratora są nałożone pewne ograniczenia. System Strapi wspiera wiele baz danych, w tym MongoDB, która nie wspiera SQL. Z tego powodu, nie jest możliwe pisanie natywnych zapytań do bazy danych. Do operacji na bazie danych Strapi udostępnia “query engine”. Jest to biblioteka do operacji na bazie danych z poziomu języka JavaScript. API udostępnione programistom przez “query engine” jest znacznie ograniczone w porównaniu do natywnego SQL. Nie można używać funkcji specyficznych do wybranej bazy danych. Operacje są ograniczone do prostych zapytań CRUD.

Strapi umożliwia operowanie na danych za pomocą GraphQL. Ta opcja pozwala na pobieranie konkretnych danych o wpisie zamiast całości informacji. W celu pobierania niestandardowych informacji, wymagane jest pisanie własnych “resolwerów”. Pisanie resolwerów przypomina pisanie funkcji obsługujących zapytania do REST API. Wymaga pisania kodu źródłowego w języku JavaScript.

Podsumowując, Strapi jest dobrym wyborem, jeśli potrzebny jest system CMS pozwalający na bardziej złożone od CRUD operacje na danych. Niezbędne będzie jednak pisanie własnych funkcji obsługujących zapytania z wykorzystaniem ograniczonego API do operacji na bazie danych.

Znacznym minusem Strapi są duże wymagania wobec sprzętu. Minimalna ilość pamięci to 2GB [1]. Nie jest to problem dla przedsiębiorstw, ale może to zwiększyć koszty osób zarządzających mniejszymi stronami internetowymi.

3.3. PostgREST

PostgREST nie jest systemem CMS. Twórcy definiują ten program, jako samodzielny serwer internetowy, który przemienia bazę danych Postgres w API REST [2]. Mimo tego, PostgREST spełnia dużą część założeń tej pracy dyplomowej.

PostgREST jest przeznaczony do pracy tylko z bazą danych PostgreSQL. Dzięki takiej specjalizacji, umożliwia on korzystanie z funkcji specyficznych dla bazy danych PostgreSQL. Umożliwia wyłuskiwanie danych z kolumn o typie `json` [4] i korzystanie z funkcji `full-text search` [3].

Minusem serwera PostgREST jest to, że zapytania SQL są kodowane w parametrach zapytania HTTP, co zmniejsza czytelność zapytań SQL. Można to zauważyć w przykładowym zapytaniu z dokumentacji PostgREST pobierającym dane z wielu tabel:

```
GET /films?select=title,actors!inner(first_name,last_name)
&actors.first_name=eq.Jehanne HTTP/1.1
```

Powinno zwrócić dane JSON:

```
[{
  "title": "The Haunted Castle",
  "actors": [{
    "first_name": "Jehanne",
    "last_name": "d'Alcy"
  }]
}]
```

Podsumowując, jeśli aplikacja będzie wymagała dużej ilości niestandardowych operacji na bazie danych, PostgREST z dodatkiem prostej aplikacji serwerowej jest dobrym wyborem, jeśli nie potrzeba panelu administratora. Jest to jedyne omawiane do tej pory narzędzie, które nie ogranicza w większym stopniu możliwości operowania na bazie danych.

3.4. Podsumowanie

Podsumowując, można zauważyć, że na rynku nie ma popularnego narzędzia spełniającego założenia pracy. Istnieje wiele narzędzi, które można dostosować do założeń pracy, ale takie zastosowanie nie jest zamierzone przez twórców, lub wymaga pisania kodu poza zapytaniami SQL.

4. PROJEKT SYSTEMU

4.1. Projekt aplikacji frontend

Aplikacja frontend została zaprojektowana w sposób typowy dla aplikacji SPA [7] z wykorzystaniem biblioteki React.js. Aplikacja nie wymaga ładowania nowej strony przez przeglądarkę podczas pracy.

Składa się z komponentów, z których niektóre są używane w wielu miejscach aplikacji. Komponenty, które implementują skomplikowaną funkcjonalność zawierają komponenty podrzędne.

Główny komponent aplikacji to widok kart (TabsView). Komponent zawiera pasek kart (TabBar) oraz komponent służący do wybierania edytorów (EditorSelector). Komponent TabBar implementuje listę kart, która pozwala użytkownikowi na przełączanie się pomiędzy edytorami. Karty, które nie są aktywne są chowane za pomocą ustawienia wartości CSS `display` na `none`. W ten sposób, nie tracą danych przechowywanych w stanie komponentu przy przełączaniu kart. Komponent EditorSelector pozwala na wybór edytora. Edytory zawierają interfejs, przy użyciu którego administrator wykonuje pracę na systemie CMS.

Komponent EditorSelector pobiera dane o dostępnych edytorach ze struktury danych. Umożliwia to łatwe dodawanie nowych edytorów do aplikacji. Strukturę danych widać na listingu 1. Struktura danych przechowuje dane o folderach, które mogą zawierać edytory. Foldery i edytory mają ikonę oraz opis, a edytory mają również komponent. Przy wyborze edytora, komponent przypisany do tego edytora zostanie wyświetlony na ekranie.

Listing 1: Struktura danych przechowująca dane o edytorach

```
const editors: EditorTree = {
  "Data management": {
    icon: faDatabase,
    editors: {
      "Table data management": { icon: faTable, component: TableDataManagement },
      "Data SQL editor": { icon: faCode, component: DataSqlEditor },
    },
  },
  "Schema editing": {
    icon: faTable,
    editors: {
      "Table creation": { icon: faPlus, component: TableCreation },
    },
  },
  "Endpoint management": {
    icon: faCode,
    editors: {
      "Endpoint management": { icon: faCogs, component: EndpointManagement },
    },
  },
  "User management": {
    icon: faUsers,
    editors: {
      "User creation": { icon: faUserPlus, component: UserCreation },
      "User management": { icon: faUsersCog, component: UserManagement },
    },
  },
};
```

Aplikacja frontend pozwala użytkownikowi na zmianę motywu. Użytkownik może wybrać motyw jasny i ciemny. Przełączenie motywu powoduje podmienienie zmiennych CSS z kodu aplikacji. Wybrany przez użytkownika motyw jest zapisywany z wykorzystaniem standardowej funkcji przeglądarki `localStorage` i ładowany przy uruchomieniu aplikacji. Domyślny motyw to motyw jasny (ciemny tekst na jasnym tle). Kod odpowiedzialny za przełączanie motywu został zamieszczony na listingu 2.

4.2. Projekt aplikacji backend

Aplikacja backend nie implementuje żadnego z wzorców architektonicznych. Cała aplikacja jest podzielona na trzy moduły:

- Moduł zawierający algorytmy.
- Moduł zawierający funkcje obsługujące zapytania HTTP.
- Moduł zawierający serwisy.

Moduł zawierający algorytmy zawiera najbardziej skomplikowane logicznie elementy programu. Znajdują się tutaj funkcje:

- generujące diagram ER w składni `mermaid.js` z tabel zawartych w systemie,
- parsujące zapytania SQL zawierające odniesienia do zmiennych,
- wykonywujące sparsowane drzewo zapytań SQL zawierających odniesienia do zmiennych.

Serwisy zawierają logikę biznesową i wywołują algorytmy. Duża część funkcji zawartych w module serwisów odpowiada za zbieranie danych o tabelach zawartych w bazie danych.

Funkcje obsługujące zapytania HTTP zawierają ograniczoną logikę. Funkcje te powinny jedynie wywoływać serwisy i przekształcać możliwe błędy na odpowiedzi HTTP.

Listing 2: Kod odpowiedzialny za przełączanie motywu

```
export function toggleTheme() {
  let current = getTheme() || defaultTheme;
  let newTheme: string;

  if (current.indexOf("light") !== -1) {
    newTheme = current.replace("light", "dark");
  } else {
    newTheme = current.replace("dark", "light");
  }

  setTheme(newTheme);
}

export function setTheme(scheme: string) {
  if (Object.keys(colorschemes).indexOf(scheme) === -1) {
    console.error(`Colorscheme ${scheme} NOT DEFINED!`);
    return;
  }

  for (let [key, value] of Object.entries(colorschemes[scheme])) {
    document.documentElement.style.setProperty(`--${key}`, value as string);
  }

  // Odwrócenie kolorów w diagramie ER mermaid.js, gdy wybrano motyw ciemny
  let mermaidFilter = "";
  if (scheme.indexOf("dark") !== -1) mermaidFilter = "invert(1)";
  document.documentElement.style.setProperty("--mermaid-filter", mermaidFilter);

  localStorage.setItem("colorscheme", scheme);
}
```

5. IMPLEMENTACJA KLUCZOWYCH FUNKCJI

5.1. Tworzenie nowych typów danych

Funkcjonalność tworzenia nowych typów danych nie wyróżnia przygotowanego systemu od systemów CMS dostępnych na rynku. Jest zaimplementowana, jako lista formularzy. Każdy formularz odpowiada jednej kolumnie w nowo utworzonej tabeli. Administrator może dodawać nowy formularz, lub kasować już istniejący.

Pojedynczy formularz zawiera następujące elementy:

- Pole tekstowe, gdzie administrator podaje nazwę kolumny.
- Menu rozwijane pozwalające na wybór typu kolumny.
- Pole wyboru, które pozwala określić administratorowi, czy kolumna może zawierać wartości null.
- Pole wyboru, które pozwala określić czy kolumna posiada wartość domyślną wraz z polem tekstowym, które pojawia się, gdy administrator oznaczy kolumnę jako posiadającą wartość domyślną, gdzie należy podać wartość domyślną.

Interfejs tworzenia nowych typów danych jest pokazany na rysunku 1.

5.2. Zarządzanie danymi

6. test

6.1. test sub

hello, world! lorem ipsum hello hello raz dwa trzy hello, world! lorem ipsum hello hello raz dwa trzy hello, world! lorem ipsum hello hello raz dwa trzy hello, world! lorem ipsum hello hello raz dwa trzy hello, world! lorem ipsum hello hello raz dwa trzy hello, world! lorem ipsum hello hello raz dwa trzy hello, world! lorem ipsum hello hello raz dwa trzy [6]

Literatura

- [1] Deployment - strapi developer docs. <https://docs.strapi.io/developer-docs/latest/setup-deployment-guides/deployment.html>. Dostęp: 2020-12-10.
- [2] Postgrest documentation. <https://postgrest.org/en/stable/index.html>. Dostęp: 2020-12-10.
- [3] Postgrest documentation - full-text search. <https://postgrest.org/en/stable/api.html#fts>. Dostęp: 2020-12-10.
- [4] Postgrest documentation - json columns. <https://postgrest.org/en/stable/api.html#json-columns>. Dostęp: 2020-12-10.
- [5] Wordpress theme handbook - post types. <https://developer.wordpress.org/themes/basics/post-types/#custom-post-types>. Dostęp: 2020-12-10.
- [6] Andreas Mauthe and Peter Thomas. *Professional Content Management Systems*. John Wiley & Sons, Ltd, jan 2004.
- [7] Michael Mikowski and Josh Powell. *Single page web applications: JavaScript end-to-end*. Simon and Schuster, 2013.