UNIWERSYTET EKONOMICZNY W KATOWICACH

KIERUNEK

INFORMATYKA

IRENEUSZ KUŚKA
140518

PROJEKT SYSTEMU WEB CMS

WEB CMS SYSTEM PROJECT

Praca licencjacka
napisana w Katedrze Informatyki
pod kierunkiem dr Artura Strzeleckiego

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem
i stwierdzam, że spełnia wymogi stawiane pracom dyplomowym

Pracę akceptuję

…… (data)

…… (podpis promotora)

KATOWICE 2021

Ireneusz Jhuska
Imię i nazwisko

Informatyka
Kierunek

140518
Nr albumu

Katowice, dnia 14.06.2021

OŚWIADCZENIE

Świadom(a) odpowiedzialności prawnej oświadczam, że złożona praca licencjacka/~~inżynierska~~/~~magisterska~~ pt.: Projekt systemu Web CMS
.....
została napisana przeze mnie samodzielnie.

Równocześnie oświadczam, że praca ta nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (tj. Dz. U. z 2018 r., poz. 1191, z późn. zm.) oraz dóbr osobistych chronionych prawem.

Ponadto praca nie zawiera informacji i danych uzyskanych w sposób niedozwolony i nie była wcześniej przedmiotem innych procedur związanych z uzyskaniem dyplomów lub tytułów zawodowych uczelni wyższej.

Wyrażam zgodę na nieodpłatne udostępnienie mojej pracy w celu oceny jej oryginalności przez Jednolity System Antyplagiatowy prowadzony przez Ministra Nauki i Szkolnictwa Wyższego oraz przechowywania jej w Ogólnopolskim Repozytorium Prac Dyplomowych oraz wewnętrznej bazie prac dyplomowych Uniwersytetu Ekonomicznego w Katowicach. Zostałem poinformowany o zasadach dotyczących oceny oryginalności pracy dyplomowej przez Jednolity System Antyplagiatowy.

Oświadczam także, że ostateczna wersja pracy przesłana przeze mnie drogą elektroniczną jest zgodna z plikiem poddanym ocenie w Jednolitym Systemie Antyplagiatowym.

Jednocześnie oświadczam, że jest mi znany przepis art. 233 § 1 Kodeksu karnego określający odpowiedzialność za składanie fałszywych zeznań.

Jhuska Ireneusz
(podpis składającego oświadczenie)

Spis treści

Wstęp

Internet jest obecnie jednym z najbardziej popularnych mediów na świecie, codziennie powstaje wiele blogów, portali informacyjnych czy sklepów internetowych dostęp do sieci. Dzisiaj każdy ma telefon komórkowy z dużym pakietem internetu, w każdej chwili może go wyciągnąć oraz przeczytać najnowsze informacje z otaczającego świata czy zamówić sobie nowe buty.

W niniejszej pracy przedstawiono najważniejsze cechy systemu zarządzania treścią, wyszczególniono rodzaje obecnych systemów zarządzania treścią oraz zdefiniowano czym jest CMS, szczegółowo opisano technologie potrzebne do utworzenia modelu oraz bardzo dokładnie opisano wszystkie punkty końcowe aplikacji oraz sposób ich obsługi.

W związku z dynamicznym rozwojem systemów oraz niewielką ilością dostępnych w języku python opartych o rest api część praktyczna została poświęcona na przygotowanie systemu napisanego w języku python wykorzystującego do tego specjalny pakiet o nazwie FastAPI, który udostępnia obsługę żądań internetowych oraz w łatwy sposób udostępnia dokumentację punktów końcowych. Do przygotowania interfejsu użytkownika oraz komunikacji z rest API wykorzystano JavaScript oraz specjalną bibliotekę React oraz pakiet narzędzi wykorzystywany do profesjonalnego tworzenia aplikacji frontendowych.

W modelu dodano obsługę użytkowników, w tym celu przygotowano bazę danych oraz specjalne tabele odpowiedzialne za przypisywanie ich do grup czy trzymanie informacji na temat utworzonego użytkownika.

Interfejs użytkownika pozwala na wyświetlenie szczegółów informacji na temat postów dodanych w systemie czy sprawdzenie i zarządzanie użytkownikami, wyposażony również został w proste intuicyjne narzędzie służące do dodawania nowych artykułów, czy zarządzania już istniejącymi.

W podsumowaniu wskazano najważniejsze informacje dotyczące projektu, podsumowano obecną wiedzę na temat systemów oraz wskazano możliwe kierunki rozwoju przygotowanego modelu, krótko podsumowano obecne ograniczenia oraz wskazano potencjalne ich rozwiązania.

1. Wprowadzenie do systemów CMS

Internet wywrócił nasze życie do góry nogami. To zrewolucjonizowało sposoby naszej komunikacji, do tego stopnia, że jest to obecnie preferowane przez nas medium codziennej komunikacji. Niemal we wszystkim co robimy, korzystamy z internetu. Zamawianie pizzy, zakupy, dzielenie się chwilami ze znajomymi, przesyłanie zdjęcia przez komunikator. Przed erą internetu, gdybyśmy chcieli być na bieżąco z wiadomościami musielibyśmy zejść do kiosku oraz kupić lokalne wydanie wiadomości opisujące co stało się dnia poprzedniego. Obecnie wystarczy jedno lub kilka kliknięć, aby przeczytać lokalną gazetę i każde źródło wiadomości z dowolnego miejsca na świecie, aktualizowane w każdej minucie. Sam internet został przekształcony, początkowo była to statyczna sieć do przesyłania małego ładunku bajtów między dwoma terminalami. Było to repozytorium informacji, w którym treści były publikowane i utrzymywane tylko przez doświadczonych programistów. Dziś jednak istnieją ogromne ilości informacji przesyłanych i pobieranych za pośrednictwem tego medium. Dlatego obecnie każdy z nas może być wydawcą własnych treści za pomocą tego medium, a pomógł w tym rozwój systemów CMS [1].

System CMS, czyli Content Management System w polskim tłumaczeniu system zarządzania treścią, zarządzanie treścią większość ludzi kojarzy z technologią. Mają oni oczywiście rację, ale to coś znacznie więcej niż tylko wybór technologii. Zarządzanie treścią ma wiele wspólnego również z ludźmi to ludzie korzystają z systemów i zarządzają treścią. Podczas wdrożenia systemu zarządzania treścią chodzi głównie o ułatwienie pracy ludziom, przezwyciężenie oporu wobec zmian i zapewnienie, że systemy wspierają w sposobie ich pracy. Brak koncentracji na ludziach może przesądzić o porażce wdrażanego systemu. Ludzie myślą o zarządzaniu treścią, jak o zarządzaniu stroną internetową, ale treść internetowa może być tylko jednym z rodzajów treści dostarczanych przez organizacje. Bardzo często w organizacjach łączy się treść papierową, internetową oraz wspólna zarządzaną różnymi mediami. Jak więc skutecznie zarządzać treścią? Treść należy efektywnie tworzyć, przechowywać oraz efektywnie odzyskiwać ponownie [2].

Systemy CMS pozwalają nam w łatwy sposób zarządzać oraz uporządkowywać treści. Wiele osób próbowało rozróżniać rozmyte pojęcia treści, danych a nawet wiedzy. Kluczowymi różnicami między treścią a danymi są:

- Treść jest tworzona inaczej.
- Treść jest wykorzystywana w inny sposób.

Treści tworzone są w procesie redakcyjnym. Ten proces jest tym, co robią ludzie, aby przygotować informacje do publikacji dla odbiorców i obejmuje modelowanie, tworzenie, edycję, przeglądanie, zatwierdzanie. Dwa zespoły redakcyjne mając te same informacje mogą opracować dwa zupełnie różne artykuły. Powoduje to, że tworzone treści zależą w dużej mierze od opinii redaktorów. Redaktorzy chcąc wydać opinię muszą udzielić odpowiedzi na poniższe pytania.

1. Jakie powinny być tematy treści?
2. Kto jest docelowym odbiorcą treści?
3. Pod jakim kątem należy podejść do tworzenia treści?
4. Jak długie powinny być treści?
5. Czy treści muszą być wspierane przez media?

Pomimo tego, że postęp w informatyce następuje bardzo szybko, decyzji o których mowa nie podejmuje za nas komputer. To niechlujne, subiektywne i niedoskonałe decyzje, które wylewają się z umysłu ludzkiego redaktora siedzącego za klawiaturą. Małe odchylenie, w którymkolwiek z nich może obrócić fragment treści w zupełnie innym kierunku. Treści są subiektywne i otwarte na ocenę oraz interpretację. Treści rzadko są tworzone raz, perfekcyjnie, a potem treści nigdy nie są dotykane. Treści raczej są wyciągane i udoskonalane, często nawet po opublikowaniu. Ponieważ to co było poprawne w danym momencie może ulec zmianie. To co jest teraz właściwe, jest po prostu teraz. Dlatego żeby w łatwy sposób można było wydawać treści potrzebne nam jest odpowiednie zarządzanie [3].

„Zarządzanie to proces obejmujący planowanie i podejmowanie decyzji, organizowanie, motywowanie i przewodzenie oraz kontrolowanie zużycia zasobów organizacji (ludzkich, finansowych, rzeczowych, organizacyjnych, relacyjnych i intelektualnych), żeby osiągnąć zamierzone cele w sposób sprawny i skuteczny.” Kiedy skorzystamy z powyższej definicji, musimy doprecyzować część zagadnień.

„Zarządzanie definiuje się jako proces” – jest to uporządkowany sposób działania, przygotowany z wielu kolejnych kroków: planowani, podejmowania decyzji, organizowania, motywowania, przewodzenia oraz kontroli. Tak naprawdę każde z wyżej wymienionych działań są od siebie zależne i ze sobą związane.

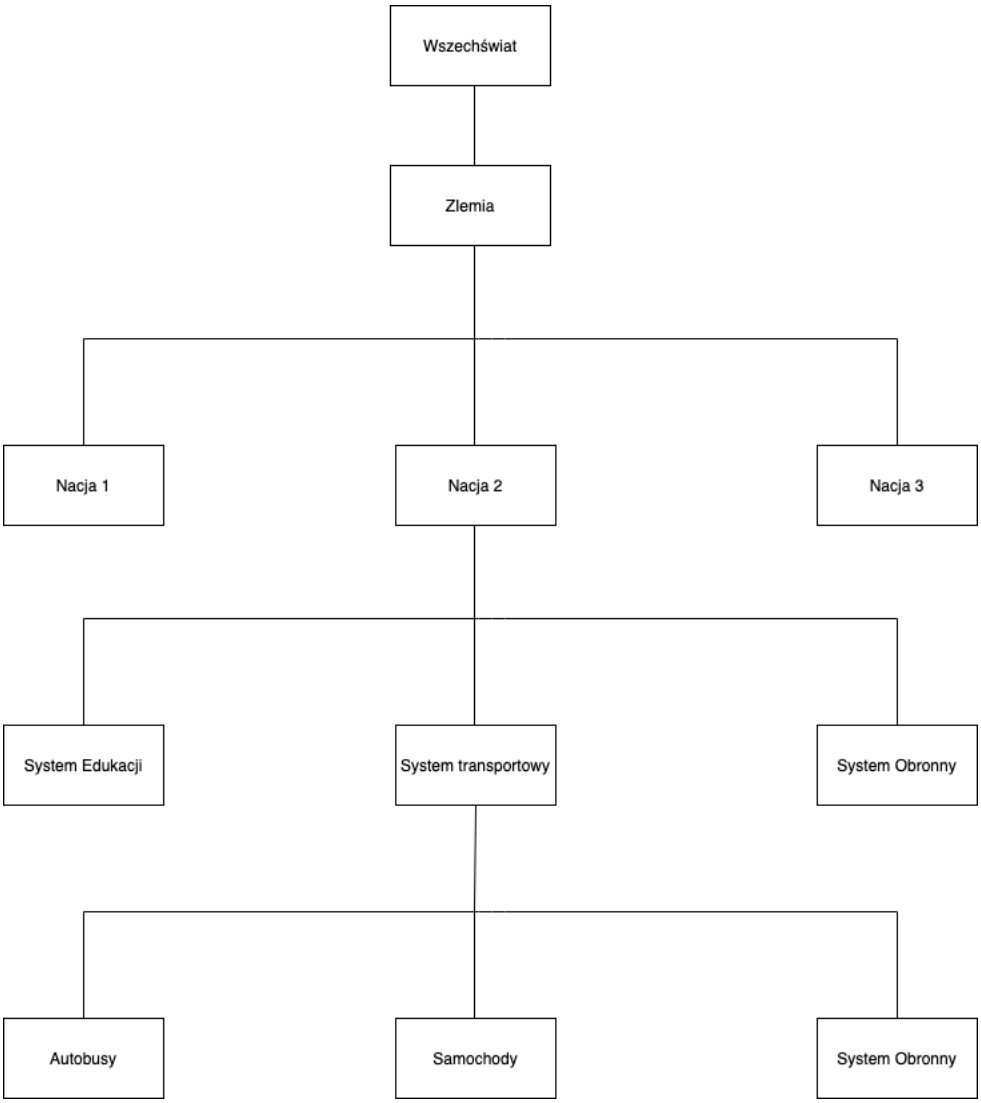
„Zarządzanie skierowane jest na wykorzystanie zasobów organizacji” – Zasadniczy podział struktur organizacyjnych można zaprezentować w ten sposób: ludzkie, finansowe, rzeczowe oraz informacyjne.

W nowocześniejszym podejściu rozróżniono sześć podstawowych kategorii: [4]

- zasoby finansowe, składają się ze wszystkich zasobów pieniężnych, które można użyć do osiągnięcia swojego planu. Np. zasoby gotówkowe, zatrzymane zyski.
- zasoby fizyczne, w skład, których wchodzą wykorzystywane technologie, lokalizacja geograficzna, budynki czy środki transportu.
- zasoby ludzkie, skupiające w sobie wszystkich ludzi, utworzone struktury pracownicze, doświadczenie czy umiejętności.
- zasoby organizacyjne, odwołują się do działań powalających na zarządzanie organizacją. Poruszają aspekty takie jak: struktura organizacyjna, procedury organizacyjne, kultura organizacyjna czy system komunikacji.
- zasoby relacyjne, do których odnoszą się więzi z otoczeniem, dotyczą one skuteczności promocji, łagodzenia i odmiennego wpływania na składniki otoczenia zewnętrznego przedsiębiorstwa, jak też współdziałania z nim. W skład tych zasobów wchodzą kontakty z agencjami biznesu.
- zasoby intelektualne, dotyczą się każdej informacji wspomagającej proces podejmowania decyzji, wiedzę technologiczną, know-how, bazy danych, prawa autorskie, patenty, potencjał rynkowy czy przewagę konkurencyjną. Zarządzanie to organizacja działań w taki sposób, by osiągnąć zamierzone plany sprawnie oraz skutecznie (Sprawność – wykonywanie działań w poprawny sposób, mądry i oszczędny nie powodujący nadmiernego zużycia zasobów przedsiębiorstwa. Skuteczność – wykonywanie odpowiednich rzeczy, które kierują do osiągnięcia planów)[5].

Mając powyższą definicję w łatwy sposób możemy dowiedzieć się czym jest zarządzanie. Wszystkie jego procesy automatyzowane są przez specjalne systemy.

Często słyszymy pojęcia takie, podejście systemowe czy system. Zazwyczaj nie zastanawiamy się nad nimi więcej, nie szukamy definicji, a próby zdefiniowania „systemu” zwykle kończą się obejmowaniem prawie wszystkiego we wszechświecie. Największym problemem jest to, że sam wszechświat jest system, a więc też jest atomem i tak jak większość rzeczy pomiędzy. W najbardziej ogólnym znaczeniu system powinien być prosty oraz zorganizowany w zestaw komponentów, które współpracują ze sobą w ściśle określony sposób [6].



Rysunek 1, Hierarchia „systemu wszechświata", źródło: opracowanie własne

Odnosząc się do wcześniejszej definicji system możemy przeglądać w kilku dziedzinach, za pomocą odpowiednich widoków, które pomogą nam w analizie oraz specyfikacji. Widoki możemy podzielić w następujący sposób:

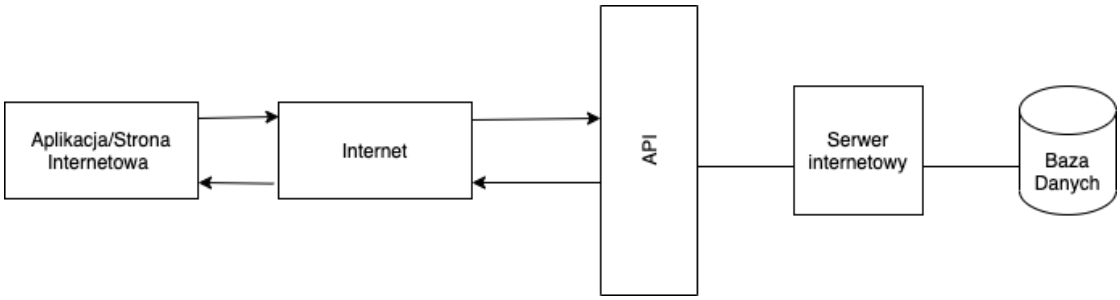
1. Widok przetwarzania – informuje nas w jaki sposób system jest podzielony na przetwarzanie informacji, materiałów i energii.
2. Widok procesora – informuje nas o tym czy przetwarzanie jest automatyczne czy ręczne.
3. Widok co/jak – kładący nacisk na oddzielenie wymagań od architektury oraz projektowania.
4. Widok inteligencji – czy każda część systemu jest prostym mechanizmem deterministycznym, czy też dostosowuje się do zmieniających się okoliczności.
5. Widok poziomu aktywności – system oddziela to co jest przechowywane statycznie od tego co robi z tymi danymi [7].

Tworzenie dobrego systemu wymaga dogłębnej analizy, takie analizy możemy podzielić na kilka obszarów. Dzieje się tak ponieważ wymagania systemu nie pochodzą tylko i wyłącznie od klienta. Generalizując, wymagania systemowe pochodzą od wszystkich interesariuszy systemu. Oraz wszystkich osób, które są nim zainteresowane, w tym użytkowników, programistów, menadżerów oraz osób biorących udział w obszarze rozwoju danego systemu [8].

Największy wpływ na wygląd oraz architekturę systemu mają programiści. Ponieważ to oni decydują, jak przenieść wymagania klientów oraz kadry zarządzającej na interfejsy oraz sposób wykorzystania dostępnej architektury. Ponieważ im system jest bardziej dojrzały tym więcej pracy skierowanej na jego rozwój niż utrzymanie już istniejących modułów. Takie podejście pozwala obecnym systemom zarządzania treści rozwinąć dwa wiodące kierunki procesu tworzenia. Tradycyjny internetowy system cms – najlepiej o nim myśleć jak o architekturze internetowej, która składa się z dwóch warstw [9]:

1. Warstwy wizualnej(frontendowej) – W której skład wchodzą:
    a. HTML – HTML to skrót od Hypertext Markup Language. Umożliwia użytkownikowi tworzenie i porządkowanie, sekcji, akapitów dla stron oraz aplikacji internetowych. HTML nie jest językiem programowania, co oznacza, że nie ma możliwości tworzenia w nim dynamicznych funkcji [10].
    b. CSS – jest językiem kaskadowych Arkuszów styli, służy do stylizacji elementów napisanych za pomocą znaczników. Oddziela on zawartość od wizualnej reprezentacji witryny. Został opracowany przez W3C (World Wide Web Consortium) w 1996 roku z powodu ograniczeń HTML, który nie zawierał tagów pomagających sformatować stronę [11].
    c. JS(JavaScript) – to lekki, interpretowany lub skompilowany na czas język programowania. Najbardziej znany jest jako język skryptowy dla stron internetowych używa go też wiele środowisk innych niż przeglądarki. JS to oparty na prototypach, wieloparadygmatyczny, jednowątkowy, dynamiczny język programowania [12].
2. Warstwy serwerowej (backendowej) – W której skład wchodzą:
    a. Serwer – Serwer to komputer, który dostarcza dane lub usługi innym komputerom za pośrednictwem sieci. Termin serwer dotyczy zarówno komputera fizycznego jak i systemu operacyjnego, na którym działa serwer fizyczny lub wirtualny [13].

b. Baza danych – Baza danych jest uporządkowanym zbiorem ustrukturyzowanych danych, aby była łatwo dostępna, łatwa w zarządzaniu i aktualizowaniu. Baza danych jest tam, gdzie znajdują się wszystkie dane naszej aplikacji [14].
c. Interfejs API – Interfejs programowania aplikacji ang. Application programming interface. API to zestaw funkcji, które pozwalają na komunikację pomiędzy frontenedem a backendem aplikacji [15].



Rysunek 2, schemat komunikacji pomiędzy API a aplikacją, źródło: opracowanie własne.

Nowsze podejście co do tworzenia systemów polega na tworzeniu bezgłowych systemów CMS, z angielskiego Headless CMS. Bezgłowy system CMS, to taki, który całkowicie odseparowuje warstwę wizualną (frontend). A do poszczególnych komponentów możemy odwołać się tylko za pomocą interfejsów API. Takie podejście jest dużo lepsze od podejścia tradycyjnego ponieważ pozwala zmniejszyć ilość punktów krytycznych, łatwiejszą obsługę błędów przy jednoczesnym zachowaniu systemu zgodnego z REST [9]. (REST - Jest to standard pisania aplikacji internetowych zapoczątkowany w 2000 roku, przez Roy Fieldinga w jego pracy doktorskiej „Representional State Transfer - Rest") [16].
Mimo różnych podejść wspólnymi cechami każdego systemu jest kontrola treści, każdy kto kiedyś tworzył rożnego rodzaju treści wie, że w łatwy sposób mogą wymknąć się z spod kontroli. CMS wie, gdzie nasze treści się znajdują. W jakim są stanie, kto mógł do nich uzyskać dostęp, jakie ma związki z innymi treściami. Poza wymienionymi podstawowymi funkcjami CMS powinien zapewnić:

1. Uprawnienia - kto może zobaczyć treści, kto może to zmienić, kto może usunąć treści.
2. Zarządzanie stanem i przepływ pracy – czy treści zostały już opublikowane, czy jest w wersji roboczej, czy został zarchiwizowany i usunięty z publicznego dostępu.
3. Wersjonowanie - Ile razy treść została zmieniana, jak wyglądały ostatnie zmiany, co się zmieniło, przywrócenie ostatnich zmian, opublikowanie poprzedniej wersji zmian.

4. Zarządzanie zależnościami – Jakie treści są używane przez jakieś inne treści co się stanie, jeśli usuną wybraną treść, jaki to będzie miało skutek dla pozostałych treści.
5. Wyszukiwanie i organizacja – Jak znaleźć konkretną treść, jak znaleźć wszystkie treści, które odnoszą się do wybranej, jak łatwo zarządzać [17].

Obecne najbardziej popularne tradycyjne systemy CMS:

WordPress – to jeden z najpopularniejszych dostępnych systemów CMS typu open source(Oprogramowanie open source to oprogramowanie z kodem źródłowym, który każdy może sprawdzić, zmodyfikować oraz ulepszyć.)[18] z globalnymi aktywami społecznościami użytkowników, programistów i wsparcia. WordPress wyróżnia się szeroką gamą opcji hostingu, rozszerzeniami funkcjonalnymi(wtyczkami) oraz estetycznymi projektami i elementami(motywami). WordPress zaczął działać podobnie jak wiele innych pakietów oprogramowania typu open source ponieważ część programistów dostrzegła potrzebę stworzenia potężnego, prostego narzędzia w oparciu o istniejący już projekt licencjonowany na licencji GPL b2/cafelog Michaela Valdrighiego [19].

Drupal – to darmowy system CMS o otwartym kodzie źródłowym napisany w PHP, rozpowszechniony na podstawie Powszechnej Licencji Publicznej GNU. Drupal wywodzi się z projektu holenderskiego studenta Dries Buytaerta. Celem projektu było zapewnienie twórcy oraz jego znajomym mechanizmu do dzielenia się wiadomościami i wydarzeniami. Buytaert przekształcił Drupala w open source w 2001 roku, a społeczność chętnie przyjęła tę koncepcję tworząc obecnie jedną z najbardziej funkcjonalnych platform CMS w sieci. Standardowa wersja Drupala znana jako Drupal Core, zawiera podstawowe funkcje, które można wykorzystać do stworzenia klasycznej witryny z broszurami, bloga, forum lub witryny społecznościowej z zawartością generowaną przez użytkowników. Funkcje zawarte w rdzeniu obejmują możliwość tworzenia i publikowania treści, tworzenie i zarządzanie użytkownikami, menu, forami i ankietami oraz do zarządzania witryną za pośrednictwem interfejsu administracyjnego. Drupal został zaprojektowany w ten sposób, żeby w łatwy sposób można było go rozszerzać o nowe funkcjonalności oraz moduły [20].

Joomla – to jeden z najpotężniejszych systemów zarządzania treścią, jest to oprogramowanie darmowe typu open source, które służy do publikowania treści w intrenecie. Jest też system wydawniczym, przeznaczonym do tworzenia wysoce interaktywnych wielojęzycznych witryn internetowych. Za jego pomocą możemy stworzyć portale społecznościowe, media, portale czy blogi. Joomla jest system uniwersalnym. Uniwersalność oznacza, że można dostosować go sobie do swoich potrzeb. Służą temu szablony, szablony możemy wybierać spośród tysięcy darmowych szablonów, kiedy jednak potrzebujemy

skorzystać z czegoś bardziej profesjonalnego możemy skorzystać z projektu płatnego. Dodatkowo funkcje naszego systemu możemy rozszerzać za pomocą dodatków, które możemy wybrać spośród wielu darmowych, lub też istnieje opcja wybrania płatnych dodatków. Jeśli jednak żaden z dodatków nie spełnia naszych wymagań możemy przygotować oraz opublikować swój własny. Podstawowe funkcje udostępniane przez system Joomla:

- Zarządzanie użytkownikami
- Menadżer mediów
- Zarządzanie banerami
- Zarządzanie kontaktami
- Ankiety
- Wyszukiwanie
- Zarządzanie linkami
- Zarządzanie treścią
- Zarządzanie kanałami dostępów
- Zarządzanie szablonami
- Zintegrowany system pomocy
- Zapewnia cechy systemu operacyjnego
- Usługi internetowe
- Wiele możliwości rozszerzenia systemu [21]

Django CMS – to dobrze przetestowana platforma CMS, która obsługuje zarówno duże jak i małe witryny internetowe. Oto najważniejsze funkcje zawarte w pakiecie Django CMS:

- Wsparcie internacjonalizacji z pomocą narzędzia i18n, która wspiera i ułatwia tworzenie stron wielojęzykowych.
- Możliwość dostosowania własnego frontendu, dzięki niemu mamy możliwość szybkiego dostępu do interfejsu zarządzania treścią.
- Wsparcie różnych edytorów z zaawansowanymi funkcjami edycji tekstu.
- Elastyczny system wtyczek, który pozwala programistom umieścić potężne narzędzia na wyciągnięcie ręki każdego redaktora korzystającego z Django CMS, bez przytłaczania trudnym skomplikowanym interfejsem.

Dlaczego warto korzystać z Django CMS:

- Dokładną dokumentację
- Łatwą i wszechstronną integrację z istniejącymi projektami
- Nie jest aplikacją monolityczną.

- Posiada dużą i aktywną społeczność programistów
- Programiści podczas tworzenia kładą nacisk na pisanie wysokiej jakości kodu oraz automatyczne testowanie [22].

Magento – w skrócie, Magento jest platformą open source używającą języka PHP (PHP to skryptowy język programowania zaprojektowany w 1994r przez Rasmusa Lerdorfa[23]) dodatkowo wykorzystujący Zend Framework (Zend Framwork jest to zbiór profesjonalnych pakietów php) [24]. Po raz pierwszy wprowadzony przez Varien w 2007 roku. Magento stale rozwija się w różnych produktach i wersjach. Obecnie staje się jedną z wiodących platform open source, która oferuje prawie wszystkie funkcje i narzędzie do budowy witryn sklepów internetowych [25]. W coraz większym stopniu osoby prowadzące sklepy internetowe oraz zespoły odpowiedzialne za content marketing (opisać content marketiong) dążą do tego, aby zapewnić dodatkową wartość, wykorzystują informacje, aby edukować, informować lub inspirować. W tym celu potrzebują już coś więcej niż tylko kreatywności, potrzebują rozwiązania technicznego, aby przełożyć swoje pomysły na blogi, historie i inne wybrane przez nich formaty treści [26].Dlatego w jednym z największych narzędzi do tworzenia sklepów internetowych dodano moduł odpowiedzialny za tworzenie i zarządzanie treścią oraz udostępniono go użytkownikom.

Obecne najbardziej popularne bezgłowe systemy CMS:

Strapi – jest wiodącym system headless opublikowany na licencji open source. Do stworzenia go wykorzystano język JavaScript. Początki pracy nad systemem rozpoczęły się w 2013 roku, kiedy Pierre Burgy, Aurélien Georget, Jim Laurie spotkali się na uniwersytecie informatycznym, od samego początku byli pasjonatami tworzenia stron internetowych. Swoją przygodę z tworzeniem stron internetowych zaczęli od tradycyjnych systemów CMS, takich jak WordPress, Joomla czy Drupal. Jednak dostępne tradycyjne systemy CMS nie były wystarczająco konfigurowalne, a jedynym sposobem było ich nadpisanie a rezultat zawsze nie był zadowalający kod zawsze był wolny i trudny do wykorzystania. Trójka założycieli jako fani open source opublikowali projekt w 2015 roku [27]. Skrót Strapi oznacza „Bootstrap your API" co oznacza „Uruchom swój interfejs API".

Ghost – Powstał w 2013 roku po bardzo udanej kampanii na Kikckstarterze(dopisać kilka słów) mającej na celu stworzenie nowej platformy nastawionej wyłącznie na profesjonalne publikowanie. Misją Ghost jest tworzenie najlepszych narzędzi typu open source dla niezależnych dziennikarzy czy pisarzy z całego świata, które mają realny wpływ na przyszłość mediów internetowych. Obecnie Ghost obsługuje niesamowitą gamę witryn internetowych, od indywidualnych blogerów, którzy dopiero stawiają dopiero pierwsze kroki.

Po całe zespoły dziennikarzy oraz redaktorów w największych organizacjach na świecie. Ghost powstał jako bezpłatna aplikacja, z płatną funkcją specjalną udostępniającą platformę na, której można uruchomić cały serwis [28].

Netlify CMS – jest to system zarządzania treścią typu open source, który udostępnia przyjazny interfejs użytkownika i intuicyjny przepływ pracy. Można używać go z dowolnym generatorem stron statycznych, w celu tworzenia szybszych i bardziej elastycznych projektów internetowych. Treść przechowywana jest w repozytorium git wraz z kodem w celu ułatwienia wersjonowania, publikowania wielokanałowego i opcji aktualizacji bezpośrednio z git. Zasadniczo Netlify CMS jest otwartą aplikacją React, która działa jako kontener dla przepływu pracy GIT, używając GitHub, GitLab lub Bitbucket API. Zapewnia to wiele korzyści:

1. Szybki internetowy interfejs użytkownika – z zaawansowanym edytorem tekstu, podglądem czasu rzeczywistego oraz obsługą przeciągania dla zewnętrznych obrazków.
2. Niezależność od platformy – działa z większością statycznych generatorów stron www.
3. Łatwa instalacja – w celu instalacji systemu wystarczy umieścić dwa pliki na serwerze, oraz odpowiednio je ze sobą skomunikować, lun alternatywnym sposobem jest skorzystanie z CDN (Content Delivery Network, czyli sieć dystrybucji treści to rozproszona sieć serwerów i ich centrów danych.) Netlify.
4. Nowoczesne uwierzytelnianie użytkowników za pomocą tokenów internetowych generowanych w jednym z hostów git.
5. Elastyczne typy treści – pozwala tworzyć nieograniczoną ilość typów treści za pomocą pól niestandardowych.
6. Całkowicie rozszerzalny – pozwala tworzyć własne podglądy, elementy interfejsu użytkownika i wtyczki do edytorów [29].

1.1 Podsumowanie

Systemy CMS możemy podzielić na dwa główne kierunki rozwoju, stare podejście jakie reprezentowane jest przez systemy takie jak WordPress czy Joomla a nowe podejście dużo chętniej wybierane przez programistów Headless, jednak nieważne na jakie podejście się zdecydujemy, ponieważ każdy system oferuje nam to samo tylko przedstawia to w nieco innej formie oraz korzysta z innych sposobów udostępniania przygotowanych treści. Przy wyborze odpowiedniego systemu powinniśmy się zastanowić, jakie koszty jesteśmy wstanie ponieść, ponieważ wykorzystanie najnowszych technologii jest dość kosztowne.

2. Projekt systemu wraz z opisem technologii

W tym rozdziale szeroko omówiono technologie, strukturę katalogów, sposoby komunikacji oraz pokazano przykłady, w jaki sposób działa aplikacja oraz jakie rodzaje żądań obsługuje. Skupiono się na opisaniu zasady działania poszczególnych endpointów(Punktów końcowych aplikacji), pokazaniu, jak aplikacja wysyła żądania oraz w jaki sposób żądania procesowane po każdej stronie. Udokumentowane również zostały dane początkowe, jakie zobaczymy instalując system na serwerze podczas rozpoczynania z nim pracy oraz wskazano punkty na przyszły rozwój systemu.

2.1 Opis technologii potrzebnych do stworzenia modelu

Python – jest językiem interpretowanym wysokiego poziomu przeznaczenia ogólnego. Filozofia projektowania Pythona kładzie nacisk na czytelność kodu dzięki znacznemu zastosowaniu znacznych wcięć. Jego konstrukcje językowe, a także podejście obiektowe mają na celu pomóc programistom w pisaniu jasnego logicznego kodu dla małych i dużych projektów [30]. Python jest językiem typowanym dynamicznie oraz wykorzystującym garbage-collector w celu dynamicznego zwalniania pamięci. Obsługuje wiele paradygmatów programowania, w tym programowania strukturalne, obiektowe oraz funkcjonalne. Python jest często opisywany jako język „z bateriami" ze względu na jego obszerną bibliotekę standardową [31]. Guido van Rossum rozpoczął pracę na Pythonem pod koniec lat 80. XX wieku jako następcę języka programowania ABC, a po raz pierwszy wydał go w 1991 roku jako Python 0.9.0 [32]. Python 2.0 został wydany w 2000 roku i wprowadził nowe funkcje, taki jak „list comprehensions" oraz mechanizm garbage-collector korzystających z licznych odwołań i został wycofany z wersją 2.7.18 w 2020 roku [33].
Python 3.0 został wydany w 2008 roku i był główną wersją języka, która nie jest w pełni kompatybilna z poprzednimi wersjami, a większość kodu pythona 2.0 nie działa bez modyfikacji w pythonie 3 [34].

Docker – to platforma typu open source, która uruchamia aplikacje u ułatwia rozwój, dystrybucję procesu. Aplikacje, które są wbudowane w dockera są dostarczane z pakietem wszystkich potrzebnych zależności w standardowej nazwie „kontener". Te kontenery działają w izolacji wysoko nad jądrem systemu operacyjnego. Dodatkowa warstwa abstrakcji może mieć wpływ na wydajność. Docker zapewnia narzędzie do automatyzacji aplikacji, kiedy są wdrażane w kontenerach. W kontenerze środowiskowym, w którym aplikacje są

zwirtualizowane i wykonywane, docker dodaje dodatkową warstwę wdrożenia silnika na wierzchu. Sposób w jaki docker został zaprojektowany zapewnia szybkie i lekkie środowisko, w którym można programować, działać wydajnie a ponadto zapewnia udogodnienia w procesie publikacji wersji produkcyjnych aplikacji [35].

PostgreSQL – to obiektowo-relacyjny system zarządzania bazami danych (ORDBMS), który był rozwijany w różnych formach od 1977 roku. Zaczął się jako projekt o nazwie Ingres na Uniwersytecie Kalifornijskim w Berkeley. Sam Ingres został później opracowany komercyjnie przez Relational Technologies / Ingress Corporation. W 1986 roku inny zespół kierowany przez Michaela Stonebrakera z Barkley kontynuował rozwój kodu Ingress w celu stworzenia systemu obiektowo-relacyjnej bazy danych o nazwie Postgres. W 1996 roku, z powodu nowych wysiłków na rzecz otwartego programowania i ulepszonej funkcjonalności oprogramowania nazwa Postgres została zmieniona na PostgreSQL, po krótkim okresie pracy jako Postgres95. Projekt PostgreSQL jest nadal bardzo aktywnie rozwijany na całym świecie przez zespoły programistów i współpracowników open source. PostgreSQL jest powszechnie uważany za najbardziej zaawansowany system baz danych typu open source na świecie [36].

Redis – jest to oprogramowanie typu open source, struktura pamięci buforowanie i broker wiadomości w pamięci (na licencji BSD). Obsługuje struktury danych, takie jak ciągi znaków, haszowanie, listy, zbiory, posortowane zestawy z zapytaniami o zakres, mapy bitowe, zapytania o promień i indeksy geoprzetrzenne strumienia. Redis jest bazą NoSQL, co oznacza, że nie wymaga strukturalnego języka zapytań, aby uzyskać dostęp do danych [37].

JavaScript – często w skrócie JS, jest językiem programowania zgodnym ze specyfikacjom ECMAScript.[38] JS jest wysokopoziomowym i wieloparadygmatycznym językiem programowania. Składnię opartą na nawiasach klamrowych, dynamiczne typowanie oraz opartą na prototypach orientację obiektową. JavaScript jest jedną z podstawowych technologii w sieci WWW [39].

Node.js – Na European JSConf w 2009 roku Rayan Dahl, młody programista przedstawił projekt, nad którym pracował. Ten projekt był platformą, która łączyła silnik JavaScript V8 firmy Google, pętlę zdarzeń i niskopoziomowy interfejs I/O API. Ten projekt nie był podobny do innych platform JavaScript po stronie serwera, w którym wszystkie metody były sterowane za pomocą zdarzeń i nie można było tego obejść. Wykorzystując moc praz prostotę języka JavaScript, projekt ten zmienił trudne zadanie pisania aplikacji sterowanych zdarzeniami po stronie serwera w łatwe. Projekt ten otrzymał owację na stojąco i od tego czasu spotyka się z bezprecedensowym wzrostem, popularnością oraz przyjęciem. Projekt nosił nazwę node.js ale teraz jest znany programistom po prostu jako Node [40].

Npm – To największy na świecie rejestr oprogramowania. Deweloperzy open source ze wszystkich kontynentów używają npm'a do udostępniania i publikowania pakietów, a wiele organizacji używa npm do zarządzania oprogramowaniem używanym wewnątrz organizacji. Npm składa się z trzech odrębnych komponentów strony internetowej, interfejsu poleceń oraz rejestru. Strona internetowa pozwala konfigurować profile oraz odkrywać pakiety. Interfejs poleceń działa w terminalu z niego korzysta większość programistów. W rejestrze znajduje się duża publiczna baza oprogramowania JavaScript [41].

Git – to rozproszony system kontroli wersji dostępny na wszystkich popularnych platformach programistycznych w ramach bezpłatnej licencji na oprogramowanie. Ważną różnicą między gitem a jego poprzednikami jest to, że podnosi on wersje oprogramowania do najwyższych. Programiści bardzo dbają o wersje oprogramowania, a git wspiera to, udostępniając każdemu deweloperowi pełną prywatną kopię repozytorium oprogramowania i liczne sposoby zarządzania wersjami w jego kontekście. Możliwość powiązania repozytorium lokalnego z wieloma repozytoriami zdalnymi umożliwia programistom i ich liderom tworzenie różnych interesujących rozproszonych przepływów pracy, z których większość nie jest możliwa do uruchomienia w tradycyjnym scentralizowanym systemie kontroli wersji. Lokalne repozytorium sprawia, że git jest responsywny, łatwy w konfiguracji i może działać bez połączenia z Internetem [42].

React – to framework JavaScript. React został pierwotnie stworzony przez inżynierów z Facebooka, aby sprostać wyzwaniom związanym z opracowywaniem złożonych interfejsów użytkownika z zestawami danych, które zmieniają się w czasie. Nie jest to banalne przedsięwzięcie i musi być nie tylko możliwe do utrzymania, ale także skalowalne, aby działało w skali Facebooka. React narodził się w organizacji reklamowej Facebooka, gdzie wykorzystywał tradycyjne podejście Model-View-Controller po stronie klienta. Aplikacje takie jak te zwykle składają się z dwukierunkowego wiązania danych wraz z szablonem generowania. React zmienił sposób tworzenia tych aplikacji, dokonując odważnych postępów w tworzeniu stron internetowych. Kiedy React został wydany w 2013 roku, społeczność twórców stron internetowych była zarówno zainteresowana, jak i zdegustowana tym, co robi React. React rzuca wyzwanie konwencjom, które stały się faktycznymi standardami najlepszych praktyk JavaScript. React robi to, wprowadzając wiele nowych paradygmatów i zmieniając status quo w zakresie tworzenia skalowalnych i łatwych w utrzymaniu aplikacji JavaScript oraz interfejsów użytkownika. Wraz ze zmianą mentalności programowania front-end, React oferuje bogaty zestaw funkcji, które sprawiają, że tworzenie aplikacji jednostronicowej lub interfejsu użytkownika jest dostępne dla programistów o wielu

poziomach umiejętności - od tych, którzy dopiero poznali JavaScript, po doświadczonych weteranów sieci [43].

2.2 Struktura plików oraz katalogów projektu

Katalog główny – katalog główny, nazwany CMS katalog ten zawiera w sobie całą strukturę poszczególnych katalogów oraz konfigurację dockera. Z obrazu dockera korzysta baza danych, backend aplikacji oraz redis. Frontend aplikacji nie korzysta z obrazu dockera.

Katalog Backend – katalog zawiera informacje na temat backendu aplikacji, strukturę bazy danych, poszczególne definicje każdej ścieżki, każdego endpointu (endpoint – jest to ścieżka w aplikacji internetowej, która zwraca odpowiednie dane bądź też usuwa lub edytuje dane w bazie danych).

Katalog Frontend – katalog zawiera pliki html, js oraz wszystkie pozostałe potrzebne do utworzenia wyglądu aplikacji. Katalog jest podzielony na 3 dodatkowe foldery, public, src oraz automatycznie tworzony katalog node_modules, katalog ten zawiera wszystkie potrzebne zewnętrzne biblioteki wykorzystywana w projekcie. W katalogu public znajdują się wszystkie skompilowane pliki js, które tworzą cały frontend aplikacji. Katalog src zawiera nieskompilowane pliki całej aplikacji, które pozwalają edytować poszczególne funkcje oraz zawierają kod, łatwy do edycji oraz zrozumienia przez programistę.

2.3 Baza danych oraz struktura rekordów aplikacji

Baza danych całej aplikacji zawiera osiem tabel, każda z tabel zawiera informacje potrzebne do przechowywania danych na temat użytkowników, artykułów, statusu artykułu, ustawień użytkownika, kategorii każdego artykuły, tagów oraz uprawnień, jakie są dostępne. Nazwy wszystkich tabel prezentują się w następujący sposób: user, group, privilages, settings, posts, status, category, privilage. Opis każdej tabeli:

a. User: id (klucz główny tabeli), name, email, password_hash, id_group,(klucz obcy), setting_id(klucz obcy), is_active.
b. Group: id (klucz główny tabeli), name, id_ privilage (klucz obcy).
c. Privilage: id (klucz główny tabeli), can_edit, can_save, can_add_user, can_delete_user, can_publish_article, can_create_admin_role
d. Post: id (klucz główny tabeli), name, img(text filed), created_at, last_edit, content, created_by, edited_by, status_id, category_id.
e. Status: id (klucz główny tabeli), name.

f. Settings: id (klucz główny tabeli), color, display_name.

g. Category: id (klucz główny tabeli), name, description.

Dane początkowe aplikacji – Każda tabela zawiera dane początkowe potrzebne do działania z aplikacją, które prezentują się w następujący sposób:

a. Category:
   - (id: 1, name: „Aktualności”, description: „Najważniejsze aktualności”)
   - (id: 2, name: „Nowości”, description: „Z ostatniej chwili…”)
   - (id: 3, name: „Świat”, description: „Najważniejsze wydarzenia ze świata”)
   - (id: 4, name: „Kraj”, description: „Najważniejsze wydarzenia z kraju”)
   - (id: 5, name: „Tech”, description: „Najważniejsze wydarzenia technologiczne”)

   Obecna wersja systemu nie pozwala też na dodanie żadnych nowych kategorii.

b. Groups:
   - (id: 1, name: „super_admin”, id_privilage_id: 1)
   - (id: 1, name: „admin”, id_privilage_id: 2)
   - (id: 1, name: „editor_in_chief”, id_privilage_id: 3)
   - (id: 1, name: „editor”, id_privilage_id: 4)

c. Post: ta tabela nie zawiera żadnych rekordów.

d. Privilage:
   - (id: 1, can_edit: true, can_save: true, can_add_user: true, can_delete_user: true, can_publish_article: true, can_create_admin_role: true)
   - (id: 2, can_edit: true, can_save: true, can_add_user: true, can_delete_user: true, can_publish_article: true, can_create_admin_role: false)
   - (id: 3, can_edit: true, can_save: true, can_add_user: true, can_delete_user: false, can_publish_article: true, can_create_admin_role: false)
   - (id: 4, can_edit: true, can_save: true, can_add_user: false, can_delete_user: truefalse can_publish_article: false, can_create_admin_role: false)

e. Settings:
   - (id: 1, color: „”, display_name: „”)
   - (id: 2, color: „”, display_name: „”)
   - (id: 3, color: „”, display_name: „”)
   - (id: 4, color: „”, display_name: „”)

Tabela zawiera cztery puste rekordy potrzebne do utworzenia czterech kont użytkowników.

f. Status:
   - (id: 1, name: „saved”)
   - (id: 2, name: „published”)
   - (id: 3, name: „unpublished”)
   - (id: 4, name: „draft”)

g. Tags: Ta tabela nie zawiera żadnych rekordów.

h. Users:
   - (id: 1, name: „admin”, email: „admin@admin.pl”,, password_hash: „admin”, group_id: 1, setting_id: 1, is_active: true)
   - (id: 2, name: „admin2”, email: „admin2@admin.pl”,, password_hash: „admin2”, group_id: 2, setting_id: 2, is_active: false)
   - (id: 3, name: „redaktor”, email: „ redaktor@ redaktor.pl”,, ,password_hash: „ redaktor”, group_id: 3, setting_id: 3, is_active: true)
   - (id: 4, name: „user”, email: „user@user.pl”,, ,password_hash: „user”, group_id: 4, setting_id: 4, is_active: true)

2.4 Punkty Końcowe aplikacji (endpointy)

Aplikacja posiada cztery główne ścieżki za pomocą, których możemy się komunikować z bazą danych. Ścieżki z jakich możemy skorzystać. Każda ścieżka odpowiada za konkretne tabele w bazie danych, w zależności od rodzaju żądania, jakie wyślemy oraz adresu możemy odpowiednio pobrać zawartość z tabel, zaktualizować dane w tabeli, usunąć dane czy dodać nowe rekordy:

Ścieżka tags/:

- tags/: podstawowy adres żądanie GET zwraca wszystkie dostępne tagi w aplikacji.
- tags/{id}: skorzystanie z tego adresu za pomocą żądania GET w formie tags/1: zwróci jeden wybrany rekord w przykładzie tag o id 1.
- tags/add: skorzystanie z tego adresu za pomocą żądania POST wymaga podania dodatkowego parametru „name”, który odpowiada za nazwę wyświetlanego tagu. Przykład takiego żądania:

```
7     const onSubmit = (data) => {
8       const elements = {
9         name: data.name,
10      };
11      fetch('http://127.0.0.1:8000/tags/add', {
12        method: 'POST',
13        body: JSON.stringify(elements),
14      }).then((data2) => data2.json())
15        .then((data2) => alert(`dodano tag ${data2.name}`));
16    };
17
```

Rysunek 3, przykład żądania post, źródło: opracowanie własne.

- tags/update{id}: skorzystanie z tego adresu za pomocą żądania PUT spowoduje aktualizację wybranego tagu o wskazanym id. Przykład żądania PUT:

```
7     const onSubmit = (data) => {
8       const elements = {
9         name: data.name,
10      };
11      fetch(`http://127.0.0.1:8000/tags/update/${data.id}`, {
12        method: 'PUT',
13        body: JSON.stringify(elements),
14      }).then((data2) => data2.json())
15        .then((data2) => alert(`dodano tag ${data2.name}`));
16    };
17
```

Rysunek 4, przykład żądania put, źródło: opracowanie własne.

- tags/delete{id}: skorzystanie z tego adresu za pomocą żądania DELETE spowoduje usunięcie tagu o wskazanym id. Przykład żądania:

```
7   const onSubmit = (data) => {
8     const elements = {
9       name: data.name,
10    };
11    fetch(`http://127.0.0.1:8000/tags/delete/%7Bid%7D?tag_id=${data.id}`, {
12      method: 'delete',
13      body: JSON.stringify(elements),
14    }).then((data2) => data2.json())
15      .then(() => alert(`usunięto tag ${data.id}`));
16  };
17
```

Rysunek 5, przykład żądania delete, źródło: opracowanie własne

Ścieżka /settings:

- /settings{id}: w tym modelu podstawowa ścieżka pozwalająca pobrać ustawienia o wskazanym id.
- /settings/add: ścieżka pozwalająca dodać ustawienie o wskazanym id, wykorzystywane, żeby umożliwić odpowiednim użytkownikom dodawanie własnych ustawień.

- /settings/update{id}: ścieżka odpowiada za aktualizacje wybranych ustawień dla poszczególnych użytkowników. Przykład żądania:

```
9   const onSubmit = (data) => {
10    const elements = {
11      color: data.color,
12      display_name: data.name,
13    };
14    fetch(`http://127.0.0.1:8000/settings/update/${user.setting.id}`, {
15      method: 'PUT',
16      body: JSON.stringify(elements),
17    }).then((data2) => data2.json())
18      .then((data2) => alert(`dodano tag ${data2.name}`));
19  };
20
```

Rysunek 6, przykład żądania PUT dla ustawień, źródło: opracowanie własne

Ścieżka /users:

- users/all: podstawowy endpoint ścieżki users, zwraca on listę z wszystkimi użytkownikami. Przykładowe żądanie typu GET:

```
fetch('http://127.0.0.1:8000/users/all')
  .then((resp) => resp.json())
  .then((resp) => setUsers(resp));
```

Rysunek 7, przykładowe żądanie typu get dla wszystkich użytkowników, źródło: opracowanie własne

- users/{id}: endpoint zwracający użytkownika o wskazanym id przykładowe żądanie GET

```
fetch(`http://127.0.0.1:8000/users/${resp.id}`)
  .then((item) => item.json())
  .then((item) => setCurrentUser(item));
}
```

Rysunek 8, przykładowe żądanie typu get dla wybranego użytkownika, źródło: opracowanie własne

- users/me: endpoint zwracający informacje o bieżącym użytkowniku. W obecnej wersji aplikacji endpoint ten nie jest wykorzystywany. Został przygotowany do przyszłej rozbudowy aplikacji.

- users/: wysłanie żądania pod ten endpoint wymaga podania dodatkowych parametrów, jest to żądanie typu POST, przykładowe żądanie:

```
8   const onSubmit = (data) => {
9     const elements = {
10      name: data.name,
11      email: data.Email,
12      password_hash: data.password,
13      is_active: data.Activity,
14      group_id: data.Title,
15      setting_id: 1,
16    };
17    fetch('http://127.0.0.1:8000/users/', {
18      method: 'POST',
19      body: JSON.stringify(elements),
20    }).then((data2) => data2.json())
21      .then((data2) => alert(`dodano użytkownika ${data2.name}`));
22  };
```

Rysunek 9, przykładowe żądanie typu post dla dodania nowego użytkownika, źródło: opracowanie własne

- /users/update/{id}: wysłanie żądania pod ten adres spowoduje aktualizację danych w rekordzie użytkownika o wskazanym id. W tym momencie, nie jest on używany nigdzie w aplikacji, ponieważ został on przygotowany pod przyszłą rozbudowę oraz rozwój systemu.
- /users/token – żądanie typu post, służy do autoryzacji logowania użytkownika w momencie, kiedy użytkownik jest aktywny zwraca odpowiednie dane. Kod odpowiedzialny za autoryzację

```
38  @users_router.post('/token')
39  async def generate_token(form_data: OAuth2PasswordRequestForm = Depends()):
40      user = await auth_user(form_data.username, form_data.password)
41      if not user:
42          return {'error': 'invalid cred'}
43
44      user_obj = await User_Pydantic.from_tortoise_orm(user)
45
46      if(user_obj.is_active):
47          token = jwt.encode(user_obj.dict(), JWT_SECRET)
48
49          return {
50              'id': user_obj.id,
51              'access_token': token,
52              'token_type': 'bearer'
53          }
54
55      raise HTTPException(
56          status_code=status.HTTP_401_UNAUTHORIZED,
57          detail='Account is not active'
58      )
59
```

Rysunek 10, obsługa żądania typu post odpowiedzialnego za autentykację użytkownika, źródło: opracowanie własne

Przykładowe żądanie odpowiedzi z serwera

```
55    const handleLogin = (e) => {
56      if (e.code === 'Enter' || e.type === 'click') {
57        const url = 'http://127.0.0.1:8000/users/token';
58        const formData = new FormData();
59        formData.append('username', loginValue);
60        formData.append('password', password);
61        formData.append('grant_type', 'password');
62        fetch(url, {
63          method: 'POST',
64          body: formData,
65        }).then((resp) => resp.json())
66          .then((resp) => {
67            setLogin(resp);
68            setCredentials(resp);
```

Rysunek 11, przykładowe żądanie odpowiedzi autentykacji, źródło: opracowanie własne

Polem dodatkowo wymaganym przez to żądanie jest pole „grant_type" pole jest wymagane w celu rozróżnienia typu autoryzacji. W tym momencie aplikacja obsługuje tylko i wyłącznie jeden typ autoryzacji, jakim jest podanie hasła.

Ścieżka /posts/:

- posts/: podstawowy endpoint ścieżki posts, zwraca on listę z wszystkimi Artykułami. Kod odpowiedzialny po stronie serwera za realizację żądania:

```
14
15  @posts_router.get('/', response_model=List[Post_Pydantic])
16  async def get_posts():
17      return await Post_Pydantic.from_queryset(Post.all())
18
```

Rysunek 12, kod odpowiedzialny za obsługę żądania typu get po stronie serwera, źródło: opracowanie własne

Przykładowe żądanie po wszystkie posty:

```
21    useEffect(() => {
22      fetch('http://127.0.0.1:8000/posts/')
23        .then((resp) => resp.json())
24        .then((resp) => setPosts(resp));
25    }, []);
```

Rysunek 13, przykład żądania odpowiedzialnego za pobranie wszystkich postów, źródło: opracowanie własne

- posts/add: endpoint odpowiedzialny za dodawanie nowych artykułów, jest to żądanie typu POST, które wymaga podania dodatkowych parametrów, które powinny zostać zapisane w bazie danych. Kod odpowiedzialny za realizację po stronie serwera:

```
19  @posts_router.post('/add', response_model=Post_Pydantic)
20  async def add_post(item: PostIn_Pydantic):
21      post_obj = Post(
22          name=item.name,
23          img=item.img,
24          created_at=item.created_at,
25          last_edit=item.last_edit,
26          category_id_id=item.category_id_id,
27          created_by_id=item.created_by_id,
28          edited_by_id=item.edited_by_id,
29          status_id_id=item.status_id_id,
30          content=item.content,
31      )
32      await post_obj.save()
33      return await Post_Pydantic.from_tortoise_orm(post_obj)
```

Rysunek 14, kod odpowiedzialny za realizację żądania typu post dla artykułów, źródło: opracowanie własne

Przykładowe żądanie z danymi:

```
11    const onSubmit = (data) => {
12      const elements = {
13        name: data.name,
14        img: 'obrazek',
15        created_at: Date.now(),
16        last_edit: Date.now(),
17        content: data.content,
18        status_id_id: data.status,
19        category_id_id: data.category,
20        edited_by_id: user.id,
21        created_by_id: user.id,
22      };
23      fetch('http://127.0.0.1:8000/posts/add', {
24        method: 'POST',
25        body: JSON.stringify(elements),
26      }).then((data2) => data2.json())
```

Rysunek 15, przykład żądania typu post dla artykułów, źródło: opracowanie własne

- posts/update/{id}: endpoint odpowiedzialny za aktualizowanie poszczególnych postów. Kod odpowiedzialny postronnie serwera:

```python
@posts_router.post('/update/{id}', response_model=Post_Pydantic, responses={404: {"model": HTTPNotFoundError}})
async def update_post(id: int, item: PostIn_Pydantic):
    await Post.filter(id=id).update(**item.dict(exclude_unset=True))
    return await Post_Pydantic.from_queryset_single(Post.get(id=id))
```

Rysunek 16, kod po stronie serwera odpowiedzialny za aktualizację danych, źródło: opracowanie własne

- posts/{id}: endpoint odpowiedzialny za pobieranie wybranego postu. Kod odpowiedzialny za zwrócenie danych po stronie serwera:

```python
@posts_router.get('/{id}', response_model=Post_Pydantic)
async def get_article(id: int):
    return await Post_Pydantic.from_queryset_single(Post.get(id=id))
```

Rysunek 17 kod po stronie serwera odpowiedzialny za pobranie wybranego, źródło: opracowanie własne

- posts/delete/{id}: endpoint odpowiedzialny za usuwanie wybranego artykułu. Usuwanie odbywa się za pomocą wybranego id.

  Kod po stronie serwera

```python
@posts_router.delete('/delete/{id}', response_model=Status, responses={404: {"model": HTTPNotFoundError}})
async def delete_post(post_id: int):
    remove_item = await Post.filter(id=post_id).delete()
    if not remove_item:
        raise HTTPException(status_code=404, detail={f"Tag {post_id} not found"})
    return Status(message=f"Deleted post {post_id}")
```

Rysunek 18 kod po stronie serwera odpowiedzialny za usuwanie artykułu, źródło: opracowanie własne

Przykładowe żądanie po stronie aplikacji.

```javascript
const removeArticle = (id) => {
  fetch(`http://0.0.0.0:8000/posts/delete/%7Bid%7D?post_id=${id}`, {
    method: 'delete',
  });
};
```

Rysunek 19, żądanie usunięcia wybranego artykułu, źródło: opracowanie własne

2.5 Podsumowanie

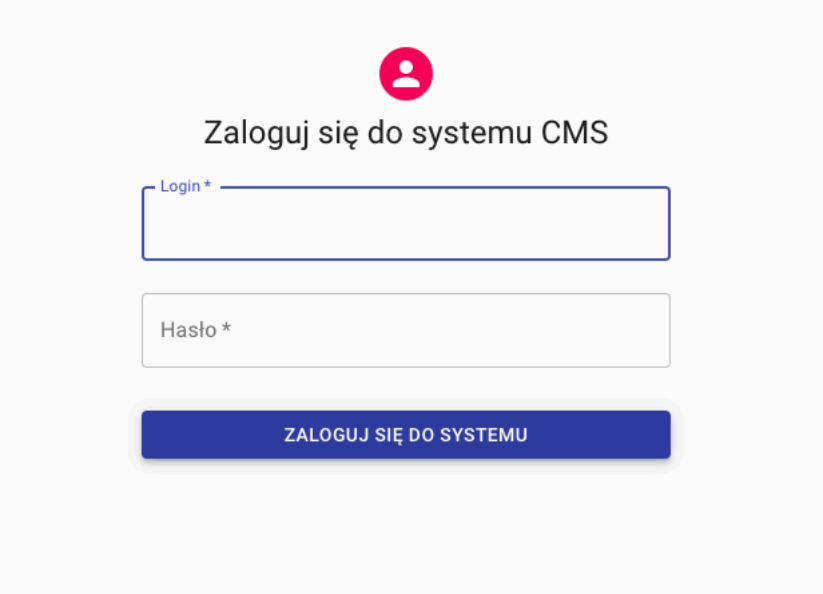
W tym rozdziale skupiono się na opisie technologii, jakie wykorzystywane są w całej aplikacji. Opisano punkty końcowe z jakimi aplikacja może się komunikować podczas codziennego użytkowania oraz brakach w początkowej wersji systemu, przedstawiono schemat bazy danych oraz wszystkie tabele wraz z danymi początkowymi, jakie trafią do użytkownika końcowego systemu.

3. Prezentacja systemu wraz z opisem funkcjonalności

Rozdział trzeci przedstawia działający system CMS. Pokazuje w jaki, sposób działają poszczególne funkcje oraz dokładnie opisuje, czego się spodziewać w każdym miejscu aplikacji. Pokazuje co się dzieje z każdym wysłanym żądaniem oraz jak zachowa się aplikacja, kiedy natrafi na błąd.

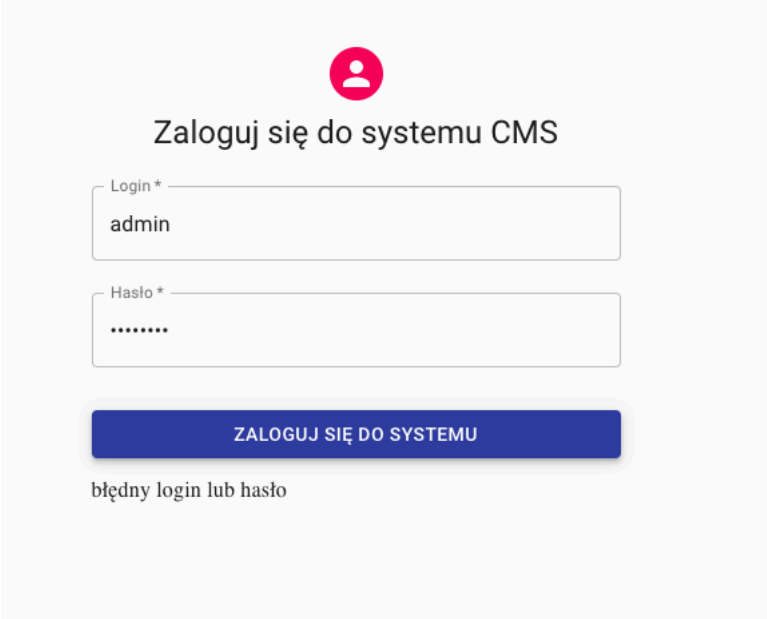
3.1 Widok logowania

Podstawowym widokiem podczas uruchamiania aplikacji jest widok logowania. Po przejściu na odpowiedni adres system poprosi o dane potrzebne do zalogowania do systemu.



Rysunek 20, widok logowania do systemu, źródło: opracowanie własne

Widok logowania składa się z dwóch pól tekstowych oraz przycisku odpowiedzialnego za logowanie. Żeby zalogować się do systemu trzeba podać wcześniej utworzony login oraz hasło. Z miejsca logowania nie ma możliwości utworzenia nowego konta. Jest to podstawowe zabezpieczenie, które chroni system przed nieautoryzowanym zakładaniem kont osób spoza firmy/domeny. W systemie konto może utworzyć użytkownik z odpowiednimi prawami np. administrator, super administrator lub redaktor naczelny. Podczas próby nieautoryzowanego logowania do sytemu zostaje zwrócony błąd logowania.

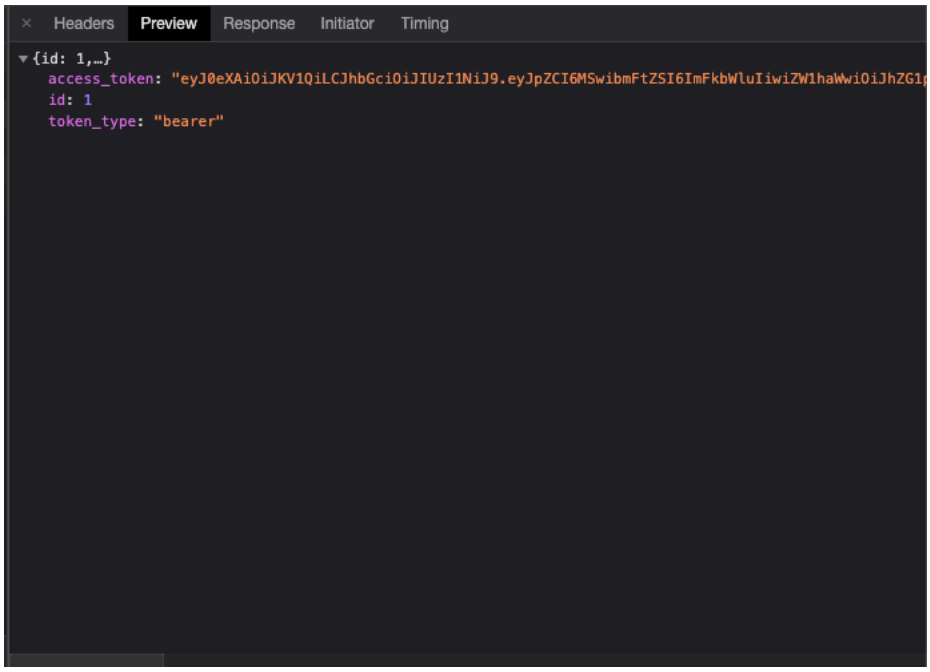


Rysunek 21, błędne logowanie do systemu, źródło: opracowanie własne

Dzieje się tak ponieważ użytkownik wysłał żądanie pod adres http://127.0.0.1:8000/users/token z błędnymi danymi, dlatego serwer po odpytaniu bazy danych oraz porównaniu pól, które zostały wysłane w formularzu zwraca status 200, ponieważ żądanie zostało przetworzone, ale nie zwraca danych pozwalających się zalogować do systemu. Zwraca jedynie informacje z błędem.

Rysunek 22, kod błędu w żądaniu, źródło: opracowanie własne

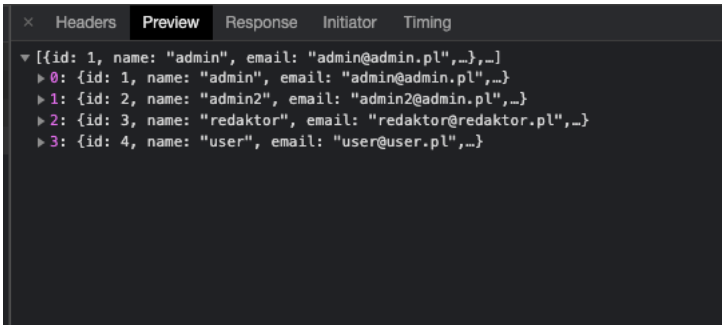
Użytkownik, który poda poprawne dane do logowania, zostanie przekierowany do głównej tablicy, stanie się tak ponieważ podał poprawne dane do logowania oraz serwer zwrócił odpowiedni żeton, który potrzebny jest do zalogowania oraz id użytkownika, dodatkowo żądanie zwróciło informacje o typie logowania.



Rysunek 23, przykładowe dane zwracane w żądaniu autoryzacji, źródło: opracowanie własne

3.2 Widok tablicy głównej

Tablica główna widok, który zawiera podsumowanie najważniejszych informacji zebranych w systemie. Pokazuje uproszczoną listę ostatnich pięciu ostatnio dodanych artykułów oraz listę pięciu najnowszych użytkowników. Mechanizm odpowiedzialny za wyświetlanie listy użytkowników wysyła żądanie pod adres http://127.0.0.1:8000/users/all oraz po otrzymaniu odpowiedzi odpowiednio wyświetla otrzymane dane.

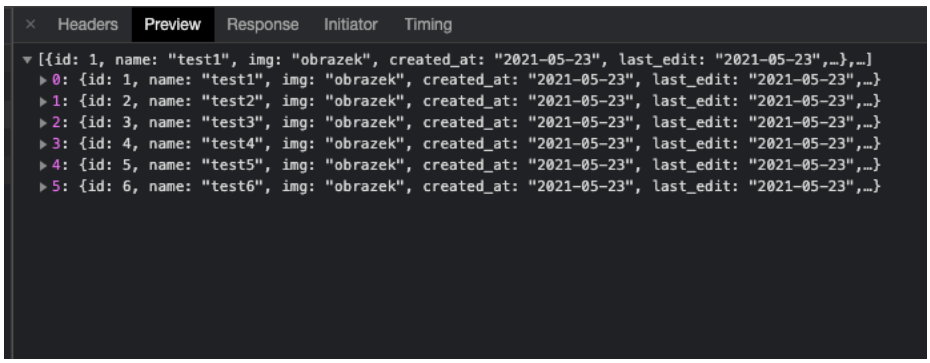


Rysunek 24, przykładowe dane zwracane w żądaniu wszystkich użytkowników, źródło: opracowanie własne

```
      <div style={{
        width: '70%', display: 'flex', flexDirection: 'column', marginTop: '5%',
      }}
      >
        <Typography component="h1" variant="h4" color="inherit" align="center">
          Użytkownicy
        </Typography>
        <Table size="xs" width="90%">
          <TableHead>
            <TableRow>
              <TableCell>Email</TableCell>
              <TableCell>Name</TableCell>
              <TableCell>Grupa</TableCell>
              <TableCell>Status</TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {users.length && users.sort((a, b) => b.id - a.id).slice(0, userCount).map((item) => (
              <TableRow key={item.id}>
                <TableCell>{item.email}</TableCell>
                <TableCell>{item.name}</TableCell>
                <TableCell>{item.group.name}</TableCell>
                <TableCell>{item.is_active ? 'Aktywny' : 'Nieaktywny'}</TableCell>
              </TableRow>
            ))}
          </TableBody>
        </Table>
      </div>
    </div>
  );
```

Rysunek 25, kod odpowiedzialny za wygląd użytkowników w liście, źródło: opracowanie własne

Żądanie po artykuły http://127.0.0.1:8000/posts/ zwraca informacje na temat postów, które są dostępne w systemie. Zadaniem aplikacji jest wyświetlenie 5 ostatnio dodanych, nie biorąc pod uwagę statusu w jakim, się znajdują.



Rysunek 26, przykładowa odpowiedź żądania wszystkich artykułów, źródło: opracowanie własne

```jsx
<div style={{ width: '70%', display: 'flex', flexDirection: 'column' }}>
  <Typography component="h1" variant="h4" color="inherit" align="center">
    Najnowsze artykuły
  </Typography>
  <Table size="xs" width="90%">
    <TableHead>
      <TableRow>
        <TableCell>nazwa</TableCell>
        <TableCell>Autor</TableCell>
        <TableCell>Kategoria</TableCell>
        <TableCell>Utworzono</TableCell>
        <TableCell>Status</TableCell>
      </TableRow>
    </TableHead>
    <TableBody>
      {post.length && post.sort((a, b) => b.id - a.id).slice(0, postsCount).map((item) => (
        <TableRow key={item.id}>
          <TableCell>{item.name}</TableCell>
          <TableCell>{item.created_by.name}</TableCell>
          <TableCell>{item.category_id.name}</TableCell>
          <TableCell>{item.created_at}</TableCell>
          <TableCell>{item.status_id.name}</TableCell>
        </TableRow>
      )) || ''}
    </TableBody>
  </Table>
</div>
```

Rysunek 27, kod odpowiedzialny za wygląd artykułów w liście, źródło: opracowanie własne

3.3 Widok artykułów

Widok artykułów przedstawia wszystkie artykuły jakie znajdują się w systemie. Pozwala on też skasować wybrany artykuł lub przejść do edycji wybranego artykułu.

Artykuły > Dodaj nowy Artykuł >

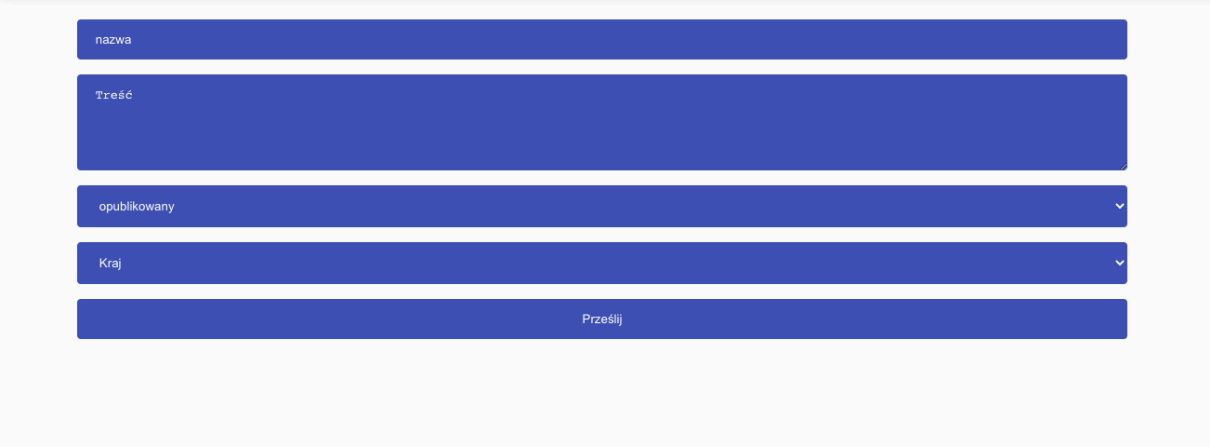
id	nazwa	Utworzony	Kategoria	Data utworzenia	Status		
1	test1	admin	Aktualności	2021-05-23	unpublished	Edytuj	Usuń
2	test2	admin	Kraj	2021-05-23	published	Edytuj	Usuń
3	test3	admin	Tech	2021-05-23	published	Edytuj	Usuń
4	test4	admin	Nowości	2021-05-23	draft	Edytuj	Usuń
5	test5	admin	Świat	2021-05-23	saved	Edytuj	Usuń
6	test6	admin	Kraj	2021-05-23	published	Edytuj	Usuń

Rysunek 28, wygląd listy artykułów, źródło: opracowanie własne

Dodatkowo widok zawiera informacje o identyfikatorze każdego artykułu, informacje o autorze, kategorii, dacie jego utworzenia oraz statusie. Aby utworzyć listę aplikacja wysyła

żądanie pod adres http://127.0.0.1:8000/posts/, który zwraca szczegółowe informacje z artykułami.

Edycja artykułu prowadzi nas do kolejnej podstrony, która jest podobna do widoku dodawania artykułu.

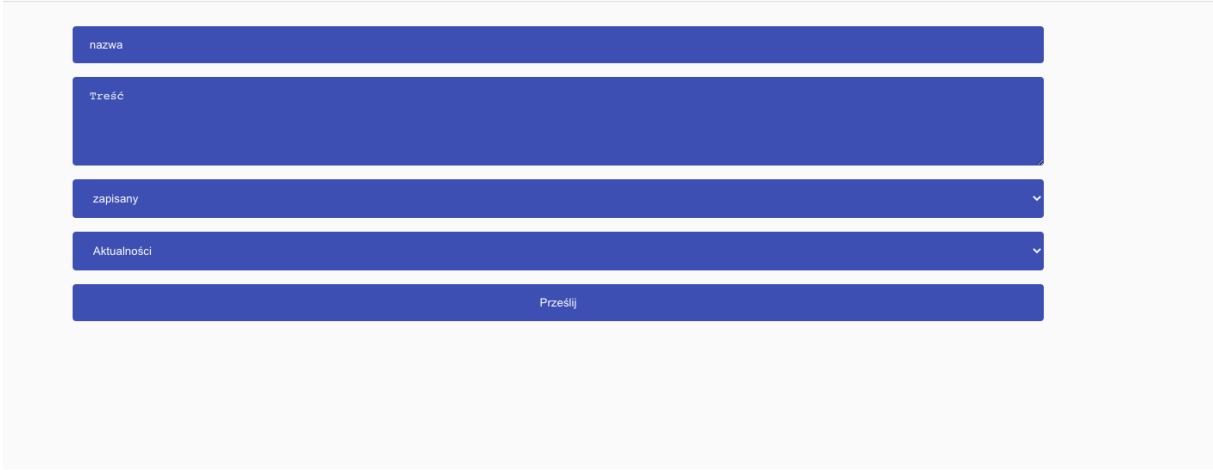


Rysunek 29, formularz edycji artykułu, źródło: opracowanie własne

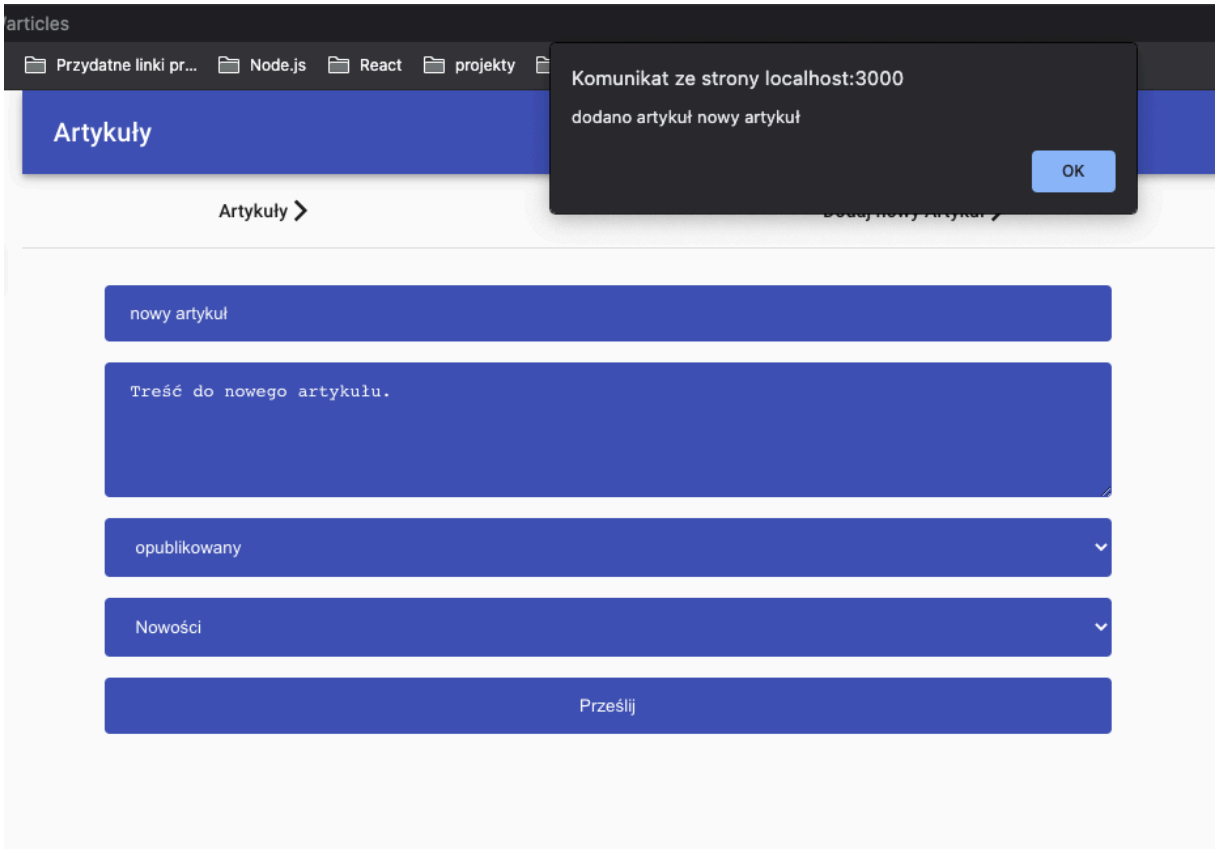
Różnica pomiędzy dodawaniem artykułu a edycją polega na tym, że edytowany artykuł zawiera informacje na temat kategorii oraz statusu jaki wcześniej został mu przyporządkowany. Nazwa oraz treść są kopiowane, ale nie pokazywana użytkownikowi, dopiero zmiana w jednym z tych pól wymusi aktualizację danych po stronie serwera.

3.4 Widok dodawania artykułu

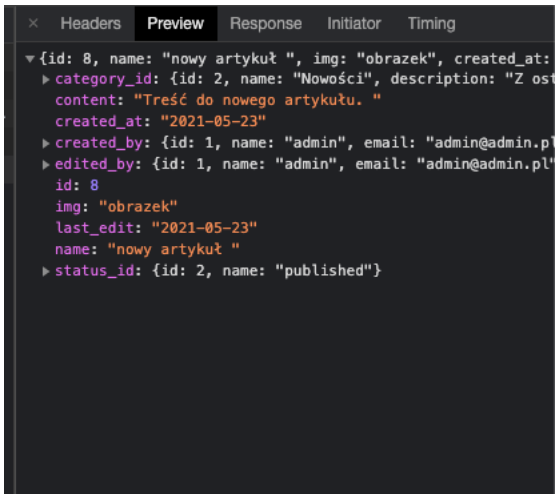
Strona z widokiem artykuły zawiera bardzo prosty formularz służący do wpisania odpowiednich danych oraz wysłania żądania do serwera oraz zwróci informację, jaki artykuł właśnie został dodany.



Rysunek 30, formularz dodawania nowego artykułu, źródło: opracowanie własne



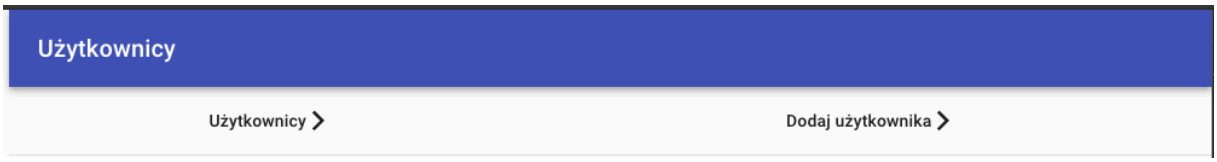
Rysunek 31, widok dodanego artykułu, źródło: opracowanie własne



Rysunek 32, przykładowa odpowiedź serwera po dodaniu artykułu, źródło: opracowanie własne

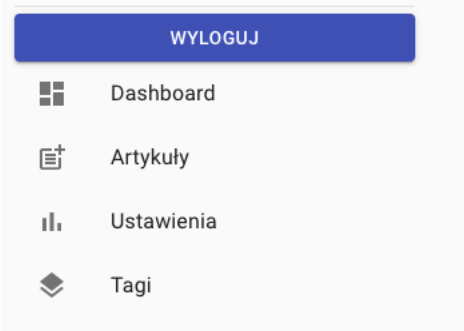
3.5 Widok użytkowników

Widok użytkowników składa się z dwóch zakładek „Użytkownicy” oraz „Dodaj użytkownika”. Widok dodawania użytkownika jest dostępny tylko i wyłącznie dla użytkownika z odpowiednimi uprawnieniami.



Rysunek 33, widok użytkowników, źródło: opracowanie własne

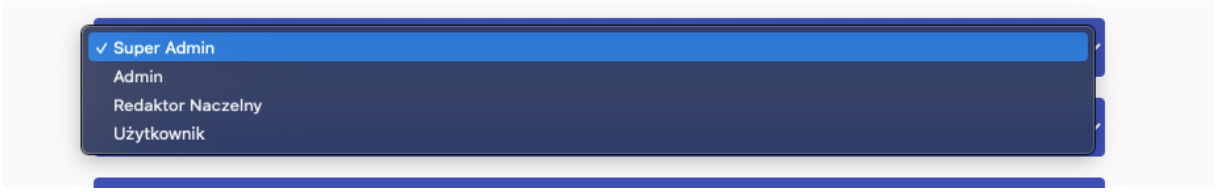
Kiedy użytkownik nie ma odpowiednich uprawnień nie jest w stanie zobaczyć zakładki z dodaniem użytkowników oraz całej listy użytkowników, ponieważ nie wyświetla się ona w menu.



Rysunek 34, widok braku uprawnień do użytkowników, źródło: opracowanie własne

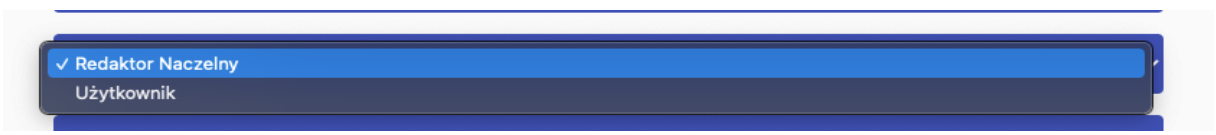
3.6 Widok dodawania użytkownika

Widok dodawania użytkowników zawiera prosty formularz, który zbiera dane potrzebne do utworzenia użytkownika. Widok użytkownika, który zawiera maksymalne uprawnienia pozwala dodać użytkownika do wszystkich grup.



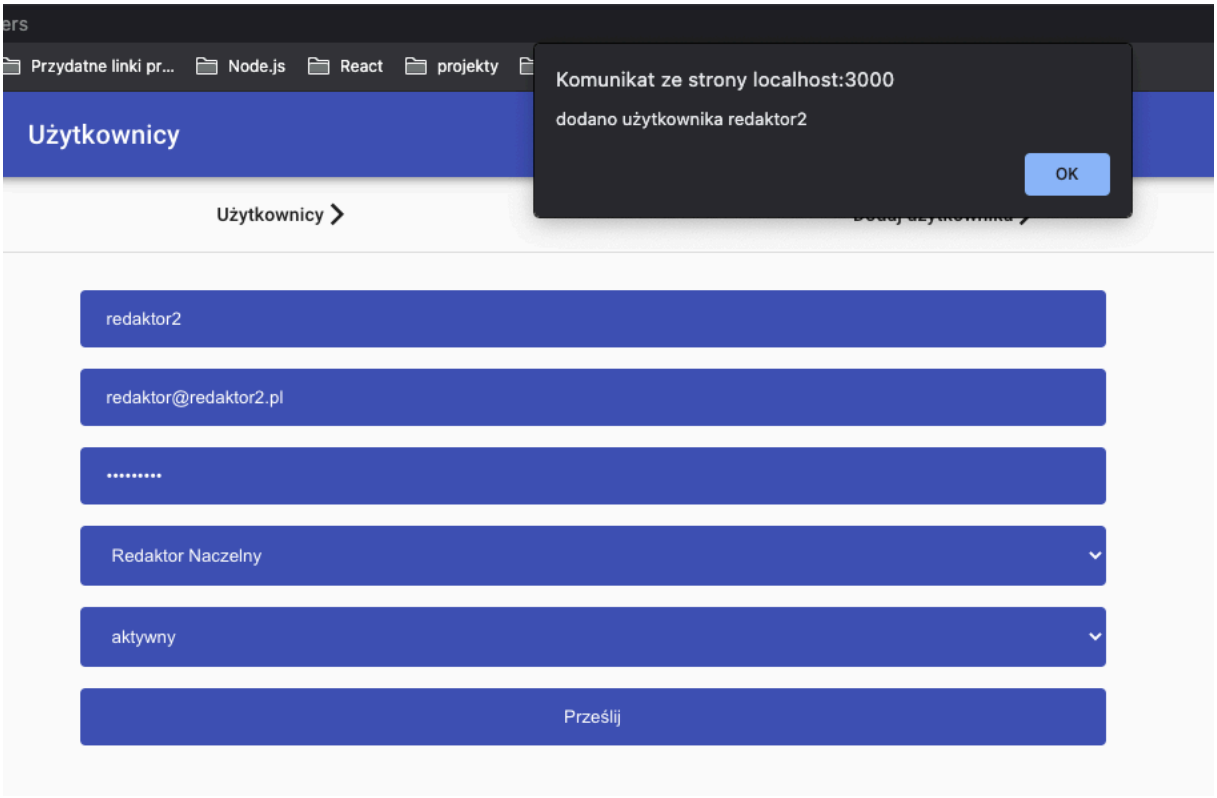
Rysunek 35, Grupy użytkowników widziane z poziomu użytkownika ze wszystkimi uprawnieniami, źródło: opracowanie własne

Użytkownicy z ograniczonymi prawami dodawania są pozbawieni dodawania użytkowników z grupy admin oraz super admin. Ma to na celu zapewnienie bezpieczeństwa w aplikacji oraz uchronić ją od szkód, jakie mogą spowodować użytkownicy ze zbyt wysokimi prawami dostępowymi do systemu.



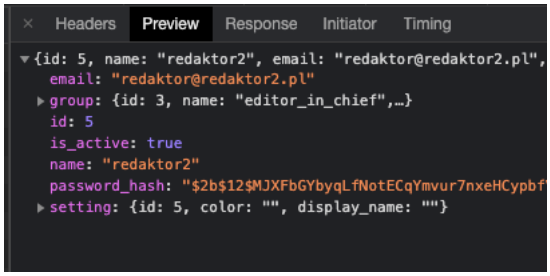
Rysunek 36, Grupy użytkowników widziane z poziomu użytkownika z ograniczonymi uprawnieniami, źródło: opracowanie własne

Poprawne dodanie użytkownika wyświetla nam komunikat o sukcesie.



Rysunek 37, widok poprawnie dodanego użytkownika, źródło: opracowanie własne

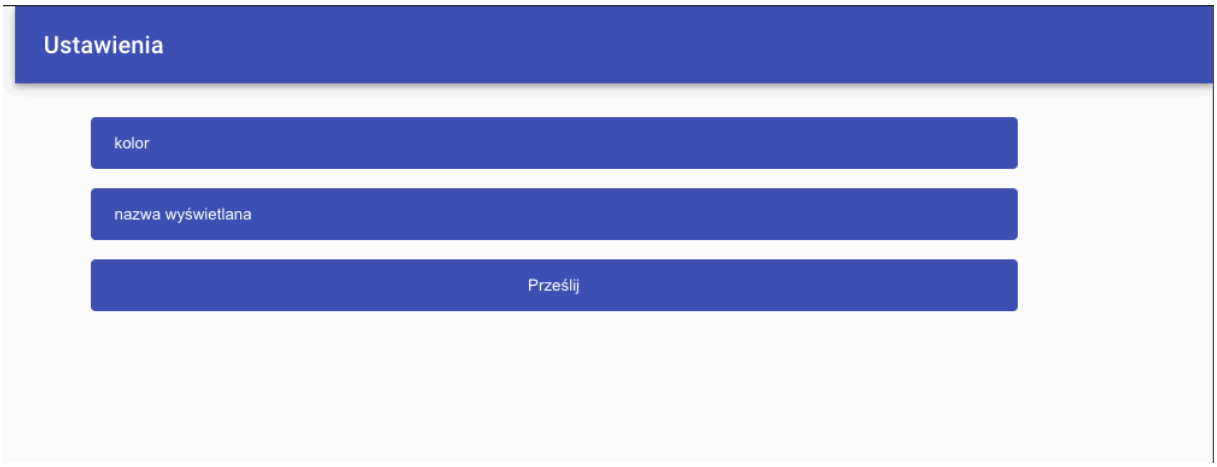
Komunikat o sukcesie zostaje wyświetlony po obsłużeniu żądania wysłanego do serwera pod odpowiedni adres http://127.0.0.1:8000/users/, przykładowa odpowiedź serwera:



Rysunek 38, przykładowa odpowiedź serwera po dodaniu użytkownika, źródło: opracowanie własne

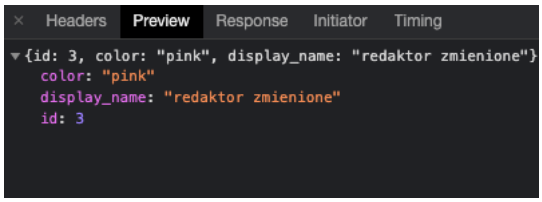
3.7 Widok ustawień

Widok ustawień pozwala zarządzać nazwą wyświetlaną oraz tłem całego systemu. Składa się z prostego formularza zawierającego dwa pola.



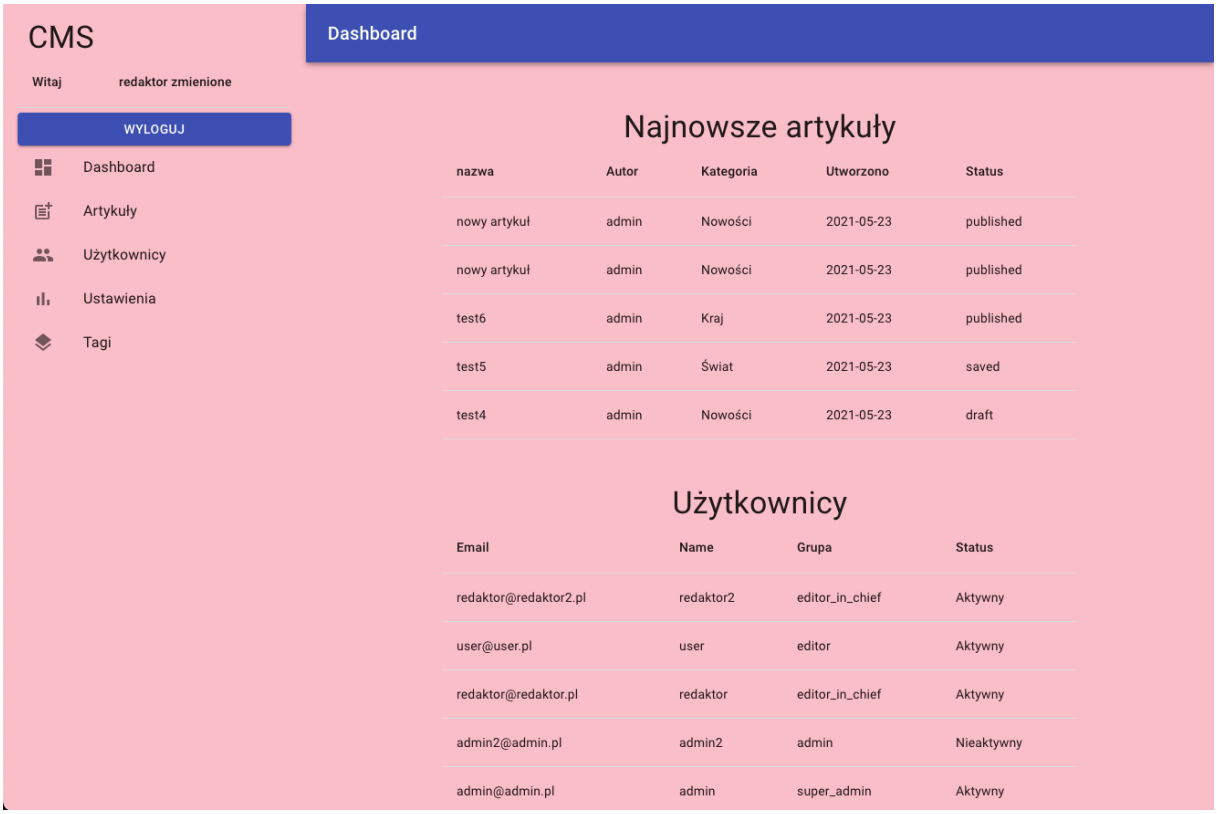
Rysunek 39, widok ustawień, źródło: opracowanie własne

Aktualizacja ustawień wymaga podania obu parametrów, po wysłaniu parametrów pod adres http://127.0.0.1:8000/settings/update/3, otrzymujemy informacje o sukcesie.



Rysunek 40, przykładowa odpowiedź serwera po dodaniu ustawień, źródło: opracowanie własne

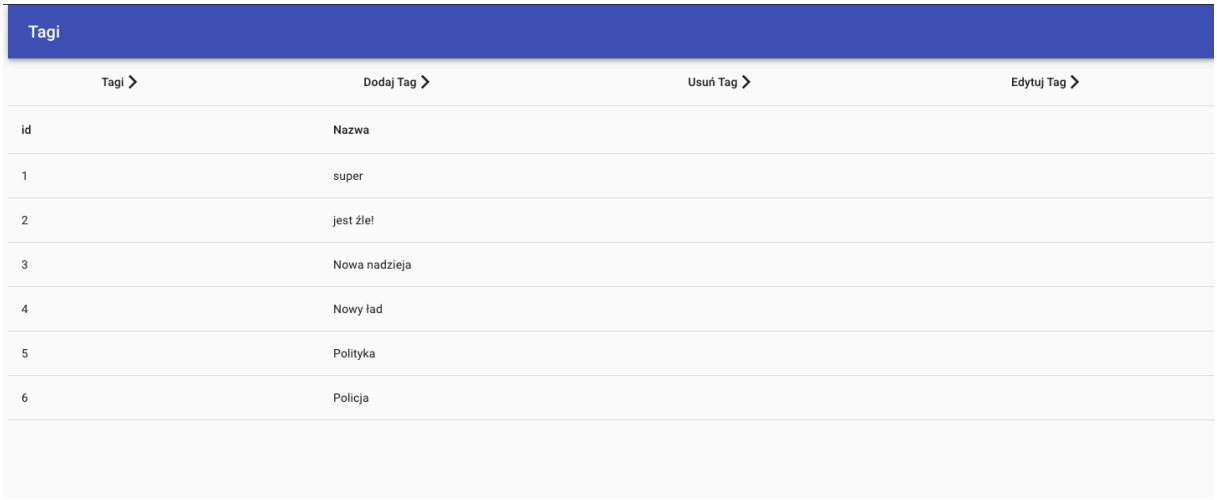
Żeby zmiany były widoczne trzeba ponownie zalogować się na konto użytkownika.



Rysunek 41, widok systemu po zmianie ustawień, źródło: opracowanie własne

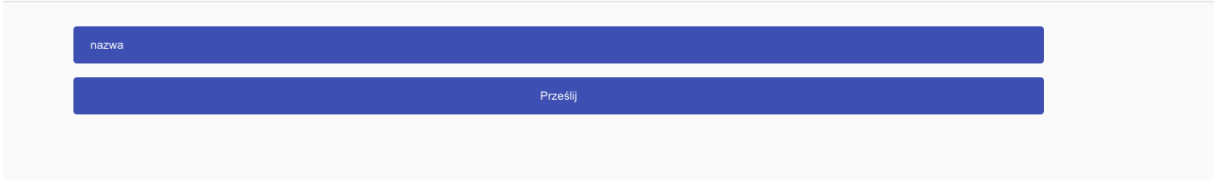
3.8 Widok tagów

Podstawowym widokiem dla tagów jest lista wszystkich tagów, widok ten zawiera informacje na temat wszystkich tagów, jakie zostały dodane do systemu.



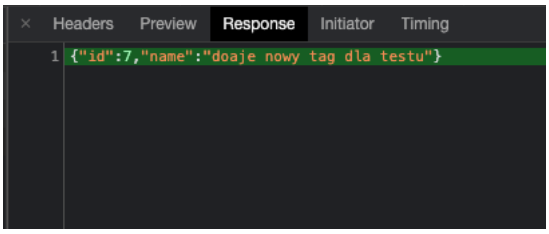
Rysunek 42, główny widok tagów, źródło: opracowanie własne

Po wybraniu zakładki „Dodaj Tag" wyświetlony zostanie formularz, w którym możemy dodać nowy tag, zawiera on jedno pole wymagające podania nazwy.



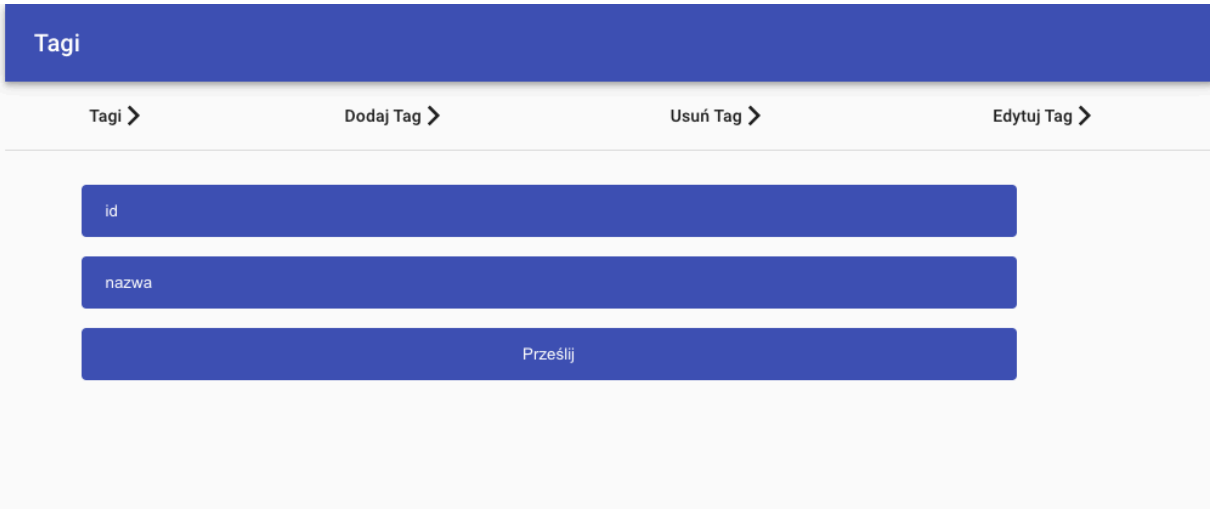
Rysunek 43, widok dodawania tagu, źródło: opracowanie własne

Po wypełnieniu formularza zostaje wysłane żądanie pod adres http://127.0.0.1:8000/tags/add, który po wysłaniu zwraca informacje o id nowego tagu oraz jego nazwie.



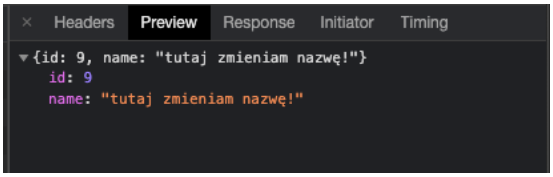
Rysunek 44, przykładowa odpowiedź serwera po dodaniu tagu, źródło: opracowanie własne

Zakładka edytuj tag wymaga podania dwóch parametrów, pierwszy parametr to identyfikator „id" wybranego tagu oraz nowa nazwa jaka ma zostać dodana w miejsce wypełnionego tagu:



Rysunek 45, widok modyfikacji tagu, źródło: opracowanie własne

Po uzupełnieniu wymaganych pól oraz kliknięciu w przycisk prześlij zostaje wysłane żądanie do serwera pod adres http://127.0.0.1:8000/tags/update/9, które informuje nas o tym, że zmieniamy nazwę dla tagu o id 9, żądanie zakończone sukcesem zwraca nam nową nazwę tagu.



Rysunek 46, przykładowa odpowiedź serwera po modyfikacji tagu, źródło: opracowanie własne

Zakładka usuń tag zawiera formularz z jednym polem, które wymaga podania identyfikatora tagu jaki chcemy usunąć.

Rysunek 47, widok usuwania tagu, źródło: opracowanie własne

Po wybraniu odpowiedniego identyfikatora oraz kliknięciu przycisku wyślij aplikacja wyśle żądanie do serwera pod adres http://127.0.0.1:8000/tags/delete/%7Bid%7D?tag_id=9, kiedy żądanie pomyślnie serwer zwróci nam informacje, jaki tag usunął

Rysunek 48, odpowiedź serwera po usunięciu tagu, źródło: opracowanie własne

Po ponownym wybraniu zakładki z listą tagów nie będzie już informacji o tagu numer dziewięć.

id	Nazwa
1	super
2	jest źle!
3	Nowa nadzieja
4	Nowy ład
5	Polityka
6	Policja
7	doaje nowy tag dla testu

Rysunek 49, widok listy tagów po usunięciu tagu, źródło: opracowanie własne

3.9 Funkcja wylogowania

Jedną z najważniejszych funkcji w systemie jest funkcja wyloguj. Kończy ona sesję aktualnego użytkownika oraz wymaga jego ponownego zalogowania się do systemu. Pozwala to uniknąć nieautoryzowanych zmian osobą z zewnątrz np. po utracie komputera czy innego urządzenia, na którym byliśmy zalogowani aktualnie do systemu.

Funkcja działa w ten sposób, że aktualnego użytkownika ustawia, jako pusty obiekt a dane jakie były ustawione po zalogowaniu, również są czyszczone, po wykonaniu tej akcji nie możemy już przeglądać żadnej innej strony niż strony logowania.

3.10 Podsumowanie

Aplikacja umożliwia dostarczanie treści w bardzo prosty sposób, jej interfejs przedstawia wszystkie kategorie dostępne w systemie, umożliwia dodawanie oraz zarządzanie użytkownikami. Nie udostępnia możliwości usunięcia użytkownika tylko jego dezaktywację, żeby w łatwy sposób można było go przywrócić do działania w systemie. Jest przygotowana do przyszłej rozbudowy, żeby z każdą kolejną aktualizacją udostępniać użytkownikom oraz twórcą jak najlepsze doświadczenia z korzystania.

Zakończenie

Webowe systemy zarządzania treścią rozwijają się bardzo dynamicznie, w ostatnim czasie powstało wiele nowych projektów oferujących zastosowanie najnowszych technologii, część z systemów do pracy wykorzystuje system wersjonowania git. Systemy obecnie bardzo ułatwiają pracę redaktorom oraz firmom zajmującym się tworzeniem treści, w łatwy sposób pozwalają dotrzeć do nowych użytkowników oraz stałych odbiorców za pomocą internetu.

Projekt systemu CMS pozwolił przyjrzeć się działaniu systemu, stawianym mu wymaganiom oraz sposobie przetwarzania oraz zarządzania informacjami. Umożliwił odpowiednie ustawianie statusu artykułu, dzięki któremu można było zachować odpowiedni przepływ informacji oraz nie uszkodzić pracy innych użytkowników. Dzięki zastosowaniu grup, użytkownicy z najniższymi poziomami uprawnień nie mają możliwości, żeby swój artykuł opublikować. Jest to zabezpieczenie przed tym, żeby artykuły niskiej jakości nie trafiły do szerszego grona odbiorców.

Wykorzystane technologie podczas tworzenia systemu zapewniają jego bezpieczeństwo. Przy projektowaniu systemu użyto języków takich jak Python, czy JavaScript ich popularność oraz ciągły rozwój zapewni to, że ich bezpieczeństwo oraz możliwość dostarczania najnowszych funkcji ciągle pozostanie na najwyższym poziomie. Wykorzystanie w projekcie Dockera w łatwy sposób pozwala uruchomienie wersji rozwojowej systemu innym programistom oraz wspólną pracę nad nowymi funkcjonalnościami.

Zaprojektowany system posiada kilka ograniczeń, nie ma możliwości dodawania własnych kategorii czy oznaczania więcej niż jednym tagiem, zaprezentowana wersja systemu nie posiada poprawnej obsługi edycji artykułów. Wykorzystane technologie podczas tworzenia pozwalają w przyszłości w łatwy sposób rozszerzyć funkcjonalność systemu.
Dzięki zastosowaniu języka Python oraz JavaScript zaplanowano rozwój systemu, w kolejnych wersjach zaplanowano dodanie opcji zarządzania istniejącymi użytkownikami czy też możliwość obsługi własnych kategorii.

Istniejące systemy oferują wiele funkcji od zarządzania całym przebiegiem treści po zarządzanie użytkownikami ich rozwój oraz wszechstronność coraz większa ich popularność stanowi też o coraz większej popularności internetu oraz chęć rozszerzania wpływów przez wydawców.

4. Bibliografia:

[1] Z. Dentzel, “How the Internet Has Changed Everyday Life,” p. 7.

[2] B. Boiko, “Content Management Bible, 2nd Edition,” p. 67.

[3] D. Barker, “Web content management: Systems, features, and best practices,” “ O’Reilly Media, Inc.,” 2016, pp. 3–4.

[4] Z.-B. Agnieszka, “Podstawy zarządzania Teoria i ćwiczenia,” 2012, p. 25.

[5] H. S. Georg Schreyogg, “Zarządzanie podstawy kierowania przedsiębiorstwem koncepcje funkcje przykłady,” Georg Schreyogg, Hotdt Steinnann, 2001, pp. 54–55.

[6] D. Hatley, P. Hruschka, and I. Pirbhai, “Process for system architecture and requirements engineering,” Addison-Wesley, 2013, p. 9.

[7] D. Hatley, P. Hruschka, and I. Pirbhai, “Process for system architecture and requirements engineering,” Addison-Wesley, 2013, pp. 18–19.

[8] D. Hatley, P. Hruschka, and I. Pirbhai, “Process for system architecture and requirements engineering,” Addison-Wesley, 2013, p. 26.

[9] G. Nuschese, “What is a CMS.” https://medium.com/swlh/what-is-a-cms-39bfdcb7201a.

[10] D. G., “What is HTML? The Basics of Hypertext Markup Language Explained,” 2019. https://www.hostinger.com/tutorials/what-is-html.

[11] A. B., “What is CSS,” 2021. https://www.hostinger.com/tutorials/what-is-css.

[12] Arpitgoyalgg *et al.*, “JavaScript,” 2021. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[13] Femi, “Understanding servers.,” 2018. https://medium.com/@theoluwafemi/understanding-servers-cbc61f910b9b.

[14] V. M. R, “What is a Database? Definition, Types and Components,” 2020. .

[15] Camille SiegelArun Dorairajan, “What is an API?,” 2020. https://apifriends.com/api-management/what-is-an-api/.

[16] M. Masse, “REST API Design Rulebook: Designing Consistent RESTful Web Service

Interfaces," " O'Reilly Media, Inc.," 2011, p. 5.

[17] D. Barker, "Web content management: Systems, features, and best practices," " O'Reilly Media, Inc.," 2016, pp. 9–10.

[18] "What is open source?" https://opensource.com/resources/what-open-source.

[19] H. S. Brad Williams, David Damstra, "Professional WordPress: Design and Development," John Wiley & Sons, 2015, p. 1.

[20] T. Tomlinson, "Beginning Drupal 7," Apress, 2010, p. 2.

[21] J. B. P. K.Patel, Savan, Dr.V.R.Rathod, "Performance Analysis of Content Management Systems- Joomla, Drupal and WordPress," *Int. J. Comput. Appl. (0975 – 8887)*, vol. 21, pp. 39–40, 2011, doi: 10.1.1.206.3027.

[22] D. A. and Contributors, "django cms Documentation Release 3.8.0," 2021. http://docs.django-cms.org/_/downloads/en/latest/pdf/.

[23] L. Welling and L. Thomson, "PHP and MySQL Web development," Sams Publishing, 2003, p. 2.

[24] "About Zend." .

[25] L. CTH, "Everything You Need to Know about Magento," 2018. https://medium.com/@lycth/everything-you-need-to-know-about-magento-447bd19ba51.

[26] M. Krukovsky, "How to Build Effective Content Management in Magento CMS," 2019. https://www.scnsoft.com/ecommerce/magento-cms.

[27] "About strapi." .

[28] "We're a proud non-profit organisation building open source technology for journalism." https://ghost.org/about/.

[29] "Overview." https://www.netlifycms.org/docs/intro/.

[30] D. Kuhlman, "A Python Book: Beginning Python, Advanced Python, and Python Exercises," 2013, p. 12.

[31] E. SCULLY, "Python vs C++: Dynamic and Static," 2020. https://careerkarma.com/blog/python-vs-c-plus-plus/#:~:text=Python is a general-

purpose,first language for new coders.

[32] P. S. Foundation, “History and License,” 2021. https://docs.python.org/3/license.html.

[33] B. Peterson, “Python 2.7.18, the last release of Python 2,” 2020. https://pythoninsider.blogspot.com/2020/04/python-2718-last-release-of-python-2.html.

[34] P. S. Foundation, “Python 3.0 Release.” https://www.python.org/download/releases/3.0/.

[35] M. A. Babak Bashari Rad, Harrison John Bhatti, “An Introduction to Docker and Analysis of its Performance,” *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 17 No. 3, p. 228, 2017.

[36] J. D. Drake and J. C. Worsley, “Practical PostgreSQL,” “ O’Reilly Media, Inc.,” 2002, p. 3.

[37] J. Nelson, “Mastering Redis,” Packt Publishing Ltd, 2016, pp. 2–3.

[38] “Draft ECMA-262 / April 6, 2021,” 2021. https://tc39.es/ecma262/#sec-overview.

[39] D. Flanagan and W. S. Like, “JavaScript: The Definitive Guide, 6th,” O’Reilly Media, 2011, p. 1.

[40] P. Teixeira, “Professional Node. js: Building Javascript based scalable software,” John Wiley & Sons, 2012, p. 1.

[41] “About npm.” https://docs.npmjs.com/about-npm.

[42] D. Spinellis, “Git,” *IEEE Softw.*, vol. 29, no. 3, pp. 100–101, 2012.

[43] C. Gackenheimer, “Introduction to React,” Apress, 2015, p. 1.

5. Spis Ilustracji: