

**Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej**

Praca dyplomowa

*System Web do zarządzania danymi z pomiarów parametrów jakości energii elektrycznej*

*A web system for managing data from power quality parameters measurements*

Autor: *Szymon Lis*

Kierunek studiów: Automatyka i Robotyka

Opiekun pracy: dr inż. Krzysztof Piątek

Kraków, 2023

Spis treści

[1. Wstęp 4](#_Toc143812059)

[1.1. Opis problemu 4](#_Toc143812060)

[1.2. Przegląd istniejących rozwiązań 4](#_Toc143812061)

[1.3. Cel i zakres pracy 5](#_Toc143812062)

[2. Projekt 6](#_Toc143812063)

[2.1. Wymagania funkcjonalne i niefunkcjonalne 6](#_Toc143812064)

[2.2. Wybrane technologie 7](#_Toc143812065)

[2.3.1. ASP.NET Core 7](#_Toc143812066)

[2.3.2. Razor Pages 9](#_Toc143812067)

[2.3.3. JavaScript 9](#_Toc143812068)

[2.3.4. SQL Server, MongoDB 9](#_Toc143812069)

[2.3. Architektura systemu 11](#_Toc143812070)

[2.3.1. MVC 11](#_Toc143812071)

[2.3.2. Wzorzec Repozytorium/Serwis 12](#_Toc143812072)

[2.4. Schemat bazy danych 14](#_Toc143812073)

[3. Implementacja proponowanego rozwiązania 15](#_Toc143812074)

[3.1. Import danych 15](#_Toc143812075)

[3.2. Dostęp do danych 17](#_Toc143812076)

[3.3. Szablony 19](#_Toc143812077)

[3.4. Raporty 20](#_Toc143812078)

[4. Interfejs użytkownika 22](#_Toc143812079)

[4.1. Argon Dashboard 23](#_Toc143812080)

[4.2. Opis funkcjonalności 24](#_Toc143812081)

[4.3. Raport 31](#_Toc143812082)

[5. Porównanie modeli bazy danych 35](#_Toc143812083)

[5.1. Założenia teoretyczne 35](#_Toc143812084)

[5.2. Badane podejścia 37](#_Toc143812085)

[5.2.1. Model relacyjny 37](#_Toc143812086)

[5.2.2. Model nierelacyjny 37](#_Toc143812087)

[5.2.3. Model nierelacyjny z wykorzystaniem kolekcji typu *Time Series* 38](#_Toc143812088)

[5.2.4. Model hybrydowy 38](#_Toc143812089)

[5.3. Porównanie wyników 39](#_Toc143812090)

[6. Podsumowanie 41](#_Toc143812091)

[Bibliografia 42](#_Toc143812092)

**Spis ilustracji**

[Rysunek 1. Najważniejsze technologie wykorzystane podczas tworzenia systemu. 7](#_Toc143812093)

[Rysunek 2. Przykładowy fragment kodu strony internetowej z kodem Razor. 9](#_Toc143812094)

[Rysunek 3. Model wzorca MVC. 11](#_Toc143812095)

[Rysunek 4. Model wzorcu Repozytorium/Serwis. 13](#_Toc143812096)

[Rysunek 5. Schemat modelu relacyjnego bazy danych. 14](#_Toc143812097)

[Rysunek 6. Podział parametru na kategorie. 16](#_Toc143812098)

[Rysunek 7. Schemat kontrolera importu danych. 17](#_Toc143812099)

[Rysunek 8. Schemat blokowy kontrolera podglądu danych. 19](#_Toc143812100)

[Rysunek 9. Schemat kontrolera szablonów. 20](#_Toc143812101)

[Rysunek 10. Schemat kontrolera Raportów. 21](#_Toc143812102)

[Rysunek 11. Przykładowy interfejs zbudowany z wykorzystaniem Argon Dashboard. 23](#_Toc143812103)

[Rysunek 12. Interfejs użytkownika – Strona główna. 24](#_Toc143812104)

[Rysunek 13. Interfejs użytkownika - import danych. 25](#_Toc143812105)

[Rysunek 14. Interfejs użytkownika - okno wyboru importowanego pliku. 25](#_Toc143812106)

[Rysunek 15. Interfejs użytkownika - odczytane nagłówki. 25](#_Toc143812107)

[Rysunek 16. Interfejs użytkownika - edytor nagłówków. 26](#_Toc143812108)

[Rysunek 17. Interfejs użytkownika - zapis do bazy. 26](#_Toc143812109)

[Rysunek 18. Interfejs użytkownika – komunikat. 27](#_Toc143812110)

[Rysunek 19. Interfejs użytkownika – podgląd. 28](#_Toc143812111)

[Rysunek 20. Interfejs użytkownika – szablony 28](#_Toc143812112)

[Rysunek 21. Interfejs użytkownika - formularz dodawania/edycji szablonu. 29](#_Toc143812113)

[Rysunek 22. Interfejs użytkownika – Raport. 29](#_Toc143812114)

[Rysunek 23. Interfejs użytkownika - formularz generowania nowych raportów. 30](#_Toc143812115)

[Rysunek 24. Interfejs użytkownika - formularz wysyłki wiadomości e-mail. 30](#_Toc143812116)

[Rysunek 25. Raport - strona tytułowa. 31](#_Toc143812117)

[Rysunek 26. Raport - opis norm. 32](#_Toc143812118)

[Rysunek 27. Raport - wyniki zgodności z normą. 33](#_Toc143812119)

[Rysunek 28. Raport - przykładowy przebieg. 34](#_Toc143812120)

[Rysunek 29. Schemat modelu hybrydowego bazy danych 38](#_Toc143812121)

# 1. Wstęp

## 1.1. Opis problemu

Energia elektryczna jest kluczowym elementem nowoczesnej cywilizacji. Jakość dostarczanej energii elektrycznej ma bezpośredni wpływ na funkcjonowanie oraz wydajność urządzeń elektrycznych. Monitorowanie i zarządzanie danymi z pomiarów parametrów jakości energii elektrycznej jest zatem ważną kwestią szczególnie w czasach transformacji energetycznej. Rosnąca liczba rozproszonych źródeł energii takich jak instalacje fotowoltaiczne przynosi wyzwania dla stabilności i jakości energii elektrycznej. Odnawialne źródła energii cechują się dużą zmiennością wytwarzanej energii oraz wprowadzaniem zaburzeń do sieci dystrybucyjnej co ma negatywny wpływ na jakość dostawy energii..

Aktualnie wykorzystywane urządzenia do monitorowania jakości energii elektrycznej, takie jak analizatory z serii PQBox lub PQM, generują duże ilości danych. Te dane, można wyeksportować do plików CSV, XSLX, jednak ręczna analiza takich wyników wydaje się zadaniem bardzo czasochłonnym, gdyż często takie pliki są ogromne, oraz w zależności od analizatorów można uzyskać różne parametry. Ponadto często dane takie są rozproszone, ponieważ nie znajdują się w pojedynczym pliku, oraz zwykle brakuje jednolitego ustandaryzowania danych pomiędzy różnymi analizatorami stworzenie bardziej uniwersalnego skryptu który mógłby tę czynność zautomatyzować staje się trudne. Dlatego naturalnym rozwiązaniem takiego problemu wydaje się ustandaryzowanie uzyskiwanych danych oraz import do środowiska które pozwoli na łatwą analizę, sprawdzenie warunków stawianych przez kryteria jakości.

## 1.2. Przegląd istniejących rozwiązań

Na rynku dostępnych jest wiele analizatorów jakości energii elektrycznej. Producenci dostarczają je wraz z oprogramowaniem umożliwiającym m.in. tworzenie wykresów i sprawdzanie zgodności z normami. Jednak są to rozwiązania dedykowane dla konkretnego modelu analizatora. Przykładem może być oprogramowanie WinPQ, które umożliwia ocenę podstawowych danych pomiarowych, przygotowanie przebiegów poszczególnych wartości w czasie jednak obsługuje on tylko analizatory z serii PQI DE oraz PQI DA. Innym przykładem może być oprogramowanie PQView, oprogramowanie to nie jest dedykowane dla konkretnych modeli urządzeń, a stworzone zostało do obsługi standardu *PQDIF* opracowanego przez Electric Power Research Institute oraz Electrotec Concepts [13]. Jednak rynek analizatorów jest dość zróżnicowany i wielu producentów nie implementuje obsługi tego standardu. Ponadto znacząca część takich rozwiązań opiera się na aplikacjach desktopowych, co wiąże się z potrzebą fizycznej instalacji oprogramowania na maszynie

## 1.3. Cel i zakres pracy

Celem niniejszej pracy magisterskiej jest zaprojektowanie i implementacja systemu internetowego do zarządzania danymi z pomiarów parametrów jakości energii elektrycznej.

W pracy przedstawiono działanie systemu, umożliwiającego ładowanie danych z plików CSV, XLSX, tworzenie szablonów, oraz generowanie raportów, które umożliwiają sprawdzenie zgodności wskazanych danych z normami wskazanymi w „Rozporządzenie Ministra Gospodarki z dnia 4 maja 2007 r. w sprawie szczegółowych warunków funkcjonowania systemu elektroenergetycznego,” (Dz.U. 2007 nr 93 poz. 623).

Jednym z kluczowych elementów pracy jest analiza wymagań dla proponowanego systemu. Wymagania te obejmują między innymi, wybór odpowiedniej technologii, proces doboru oraz analiza odpowiedniej bazy danych, proces rozpoznawania poszczególnych parametrów z nagłówków generowanych przez analizator, zaproponowanie generatora szablonów.

W pracy zawarto również porównanie różnych propozycji modeli bazodanowych. Porównane zostaną podejścia oparte o relacyjną, nierelacyjną bazę danych. Jak również podejście hybrydowe wykorzystujące obie technologie.

# 2. Projekt

Zważywszy na problem opisany we wstępie i celu pracy, pierwszym etapem realizacji takiego systemu jest postawienie wymagań, a następnie na ich podstawie dobór odpowiednich technologii, dobraniu wzorców architektonicznych i odpowiedniego systemu zarządzania danymi.

## 2.1. Wymagania funkcjonalne i niefunkcjonalne

Do wymagań funkcjonalnych, czyli takich które opisują funkcjonalności realizowane przez system można zaliczyć:

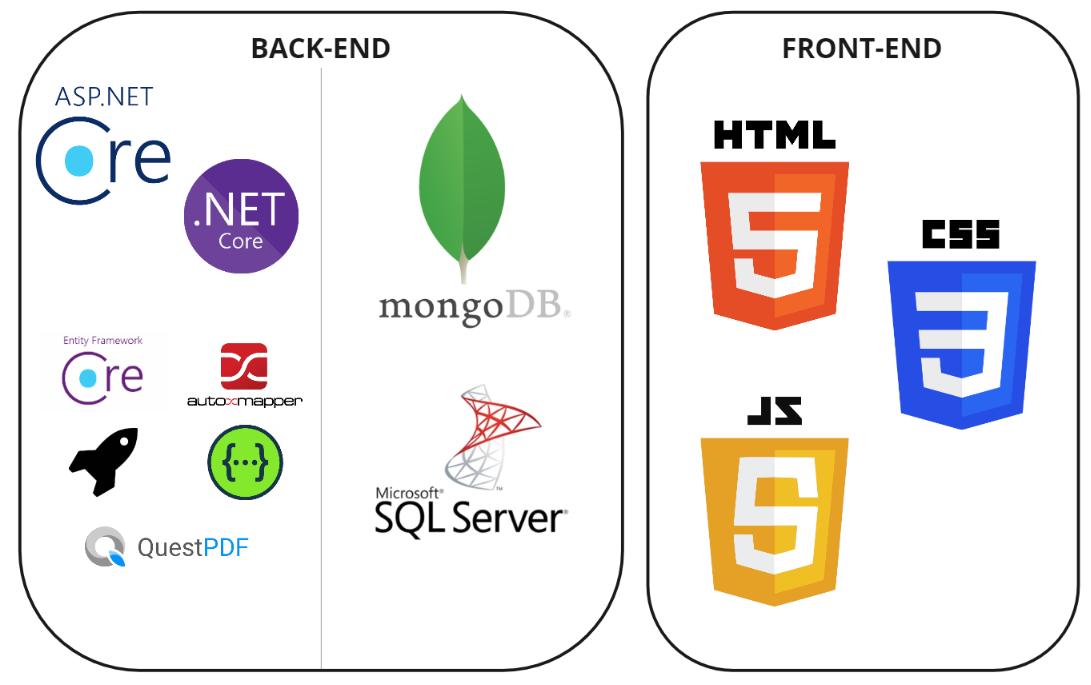
* Import danych z plików CSV, XSLX, generowanych przez analizatory. Import powinien obejmować automatyczne rozpoznawanie pliku, typu danych, separatorów, oraz wykrywać i oznaczać brakujące, błędnie sformatowane dane.
* Automatyczne rozpoznawanie parametrów energii elektrycznej. Detekcja i kategoryzacja poszczególnych parametrów powinna opierać się o nagłówki importowanych plików.
* Tworzenie szablonów na podstawie których wygenerowane zostaną raporty. Szablony powinny być zapisywane i przechowywane w bazie danych. Każdy szablon powinien zawierać zestaw predefiniowanych wykresów przebiegów poszczególnych parametrów w podanym okresie czasu.
* Sprawdzanie zgodności jakości napięcia z wymaganiami przedstawionymi w rozporządzeniu Ministra Gospodarki z dnia 4 maja 2007 roku [1]. W przypadku niespełnienia takich wymagań, system powinien podać przybliżone informacje o niezgodności.
* Generowanie raportów na podstawie zdefiniowanych szablonów, dla zadanego okresu czasu oraz wybranego punktu pomiarowego. Raport powinien zostać wygenerowany w formacie PDF. System powinien udostępniać możliwość pobrania oraz wysyłkę wygenerowanego raportu przez system e-mail.

Wymagania niefunkcjonalne definiują działanie systemu, są to ograniczenia i standardy które system musi spełniać. Poniżej przedstawiono postawione wymagania niefunkcjonalne:

* System, ma opierać się na technologii webowej, nie powinien wymagać od użytkownika dodatkowych instalacji.
* System powinien zapewniać efektywne przetwarzanie dużych ilości danych, tj. brak limitów na wielkość importowanego pliku.
* System powinien być łatwy w użytkowaniu, z intuicyjnym interfejsem użytkownika i jasnymi instrukcjami.
* System powinien zapewniać kompatybilność dla różnych wariacji importowanych danych, powinien obsługiwać różne formaty takie jak: csv, xlsx, xls.
* Zbierane dane powinny być zapisane w bazie danych, w sposób niezależny od formatu oryginalnego pliku wejściowego i umożliwiać swobodny dostęp do dowolnego zbioru danych z dowolnego zakresu czasu.

## 2.2. Wybrane technologie

Dobór odpowiednich narzędzi i technologii powinien być podyktowany przez wszystkim możliwościami i łatwością w spełnieniu postawionych wymagań. W następnym kroku systemy powinny zapewnić skalowalność, bezpieczeństwo oraz być na tyle uniwersalne, nie będąc ograniczone przez wersję ani rodzaj platformy, na której są użytkowane. Wybór konkretnych rozwiązań ma kluczowe znaczenie dla dalszego kształtu prac oraz wpływa na ostateczną użyteczność i przydatność systemu. Dlatego warto stawiać na technologię, z długim wsparciem technicznym od firm z ustabilizowaną pozycją rynkową. Jedną z takich technologii jest *.NET* wraz z językiem *C#* firmy Microsoft i to właśnie na tej technologii została oparta znaczna część systemu. Wybrane technologie zostały ukazane na Rysunek 1, a szczegółowo opisane w poniższych podrozdziałach.



Rysunek . Najważniejsze technologie wykorzystane podczas tworzenia systemu.

### 2.3.1. ASP.NET Core

Kluczowym wyborem technologii dla opisywanego systemu, było wybranie głównej platformy. Zgodnie z celami pracy i wymaganiami stawianymi systemowi, wybór padł na *ASP.NET Core*. Jest to platforma typu open-source. Umożliwia ona tworzenie różnych rodzajów aplikacji webowych. Jest to technologia bieżąco rozwijana przez Microsoft, co gwarantuje ciągły rozwój, oraz długie wsparcie techniczne. Z myślą o wsparciu dla tej technologii aktywnie rozwijane jest również jedno z głównych środowisk programistycznych – Visual Studio, które pozwala znacząco skrócić czas pracy przez wiele wbudowanych prototypów, łatwe testowanie, oraz narzędzia wspomagające budowę aplikacji na platformie *.NET*. Technologia *ASP.NET Core MVC* posiada wbudowane wsparcie dla różnych systemów zarządzania bazami danych, co pozwala na szybkie i elastyczne zarządzanie danymi. Technologia ta pozwala tworzyć aplikację wykorzystując takie języki programowania jak *C#, F#* lub *Visual Basic .NET*. Na potrzeby budowy systemu wybrano język *C#*, ze względu na jego wszechstronność, obiektowość oraz największe wsparcie ze stronu firmy Microsoft. Framework MVC sprawdza się przy tworzeniu aplikacji internetowych i interfejsów API, ponieważ umożliwia wykorzystanie wzorca MVC (Model View Controller), technologii Razor Pages, oraz łatwą integrację z systemami baz danych. Ponadto do stworzenia aplikacji wykorzystano kilka gotowych bibliotek, dostępnych w systemie *NuGet*, który jest otwartoźródłowym, domyślnym systemem zarządzania pakietami dla platfromy *.NET.* Wśród nich znalazły następujące bilbioteki i frameworki:

* Automapper – biblioteka służąca do automatycznego, mapowania (kopiowania) wartości z jednego modelu do drugiego. Działanie takiego frameworku opiera się na refleksji, dzięki czemu pola z jednego modelu zostaną automatycznie dopasowane opierając się na podobieństwie nazw i typów. Szczególnie przydatna przy wykorzystaniu koncepcji *Obiektu transferu danych* (*DTO, ang Data transfer object*).
* Json.Net – biblioteka wspomagająca operację z wykorzystaniem standardu formatu *JSON* takie jak serializacja, rozszerzenie operacji *LINQ* na danych w formacie *JSON.* W projekcie wykorzystana głównie do wspomagania operacji powiązanych z MongoDB, jako, że baza ta przechowywuje dane w formacie BSON (Binary JSON) który jest ściśle powiązany z JSON
* Entity Framework Core – lekka bibilioteka, służaca do mapowania obiektowo relacyjnego (*ORM, ang Object-relational mapping).* Umożliwia deweloperom na pracę z danymi za pomocą obiektów. Ułatwia i przyspiesza operacje związane z łączeniem się z bazą i wykonywaniem zapytań.
* QuestPDF – bilbioteka, pozwalająca na tworzenie dokumentów w formacie PDF, wykorzystująca podejście „code-only”, czyli cała logika i struktura dokumentu jest stworzona na bazie kodu źródłowego. Pozwala to programiście na pełną kontrolę nad dokumentem nie wychodząc poza, kod źródłowy aplikacji. Biblioteka ta pozawala na dodawanie wielu bazowych elementów takich jak teksty, obrazy, tabele, tła, granice. Razem z biblioteką udostępniono narzędzie które pozwala w czasie rzeczywistym obserwować tworzony plik typu PDF, wystarczy zapis kodu źródłowego, a w narzędziu zobaczymy plik PDF z naniesionymi zmianami.
* ScottPlot – biblioteka służąca do wygenerowanie wykresów na podstawie danych uzyskanych z bazy danych , oraz umieszczeniu ich w pliku PDF przy pomocy *QuestPDF*.
* Swashbuckle – pakiet bibliotek, ułatwiający generowanie dokumentacji interfejsu API opartej na standardzie OpenAPI. Pomaga testować oraz usuwać błędy. Biblioteka generuje interfejs który pozwala w łatwy sposób wprowadzić dane wejściowe i przetestować daną operacje API. Dzięki takiemu podejściu możliwe jest sprawne testowanie walidacji danych, czy poprawności uzyskiwanych rezultatów bez pisania specjalnego testowego klienta który będzie konsumował tak wystawione API.

### 2.3.2. Razor Pages

Razor to silnik generujący widoki wprowadzony w wersji trzeciej ASP.NET MVC zastępując bardziej złożoną i mniej przystępną technologię WebForms. Technologia ta działa po stronie serwera i pozwala na dynamiczne budowanie stron internetowych. Umożliwia on osadzanie logiki napisanej w języku C# w kodzie HTML, co pozwala na dynamiczną generację treści dla przeglądarki. Ze strony architektury, technologia Razor pozwala na implementację wzorca MVC, promując podział zadań. W praktyce silnik ten pozwala na mieszanie składni języka C# oraz znaczników HTML [7]. Dla zobrazowania przykładowy fragment kodu z silnika Razor przedstawiono na Rysunek 2.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

Rysunek . Przykładowy fragment kodu strony internetowej z kodem Razor.

### 2.3.3. JavaScript

JavaScript to język programowania, który jest standardem wśród technologii do tworzenia interaktywnych stron internetowych. W tym projekcie JavaScript, używany jest jako narzędzie wspomagające do manipulacji elementami strony. Ponieważ JavaScript działa po stronie klienta, pozwala na natychmiastowe reakcję działania użytkownika, bez konieczności kontaktu z serwerem. Tym samym strona internetowa staje się bardziej responsywna a interfejsy użytkownika będą bardziej dynamiczne i interaktywne. Użycie JavaScript, w połączeniu z technologią ASP.NET oraz silnikiem generującym Razor, zapewniają kompleksowe i efektywne narzędzie do tworzenia prostych jak i bardzo rozbudowanych stron internetowych.

### 2.3.4. SQL Server, MongoDB

Podczas wyboru odpowiednich technologii do zarządzania danymi w systemie, kluczową rolę odgrywa zarówno natura danych, jak i specyfika projektu. W ramach realizowanej pracy zdecydowano się na wykorzystanie dwóch różnych systemów zarządzania bazami danych – MS SQL oraz MongoDB.

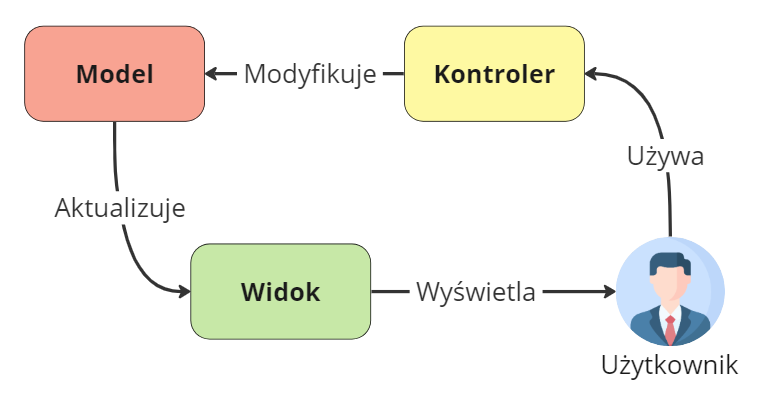
MS SQL jest systemem relacyjnej bazy danych, jest to naturalny wybór dla aplikacji opartych na platformach .NET. System ten tak jak technologia .NET jest opracowywany przez firmę Microsoft, czego następstwem jest ogromna łatwość integracji między tymi platformami. Również środowisko Visual Studio jest dobrze przystosowane do obsługi tego systemu baz danych i oferuje wiele wbudowanych narzędzi wspomagających taką integrację.

MongoDB jest nierelacyjną bazą danych działającą w systemie dokumentowym. Jest to system zarządzania bazami danych typu NoSQL (*Nie tylko SQL ang. Not Only SQL*). Taki typ odnosi się do baz danych, zaprojektowanych do obsługi danych w sposób rozproszony, nie korzystający z tradycyjnych schematów relacyjnych ani z języka zapytań SQL. System ten oferuje możliwość przechowywania danych w formacie dokumentów o typie BSON (Binary JSON). Szczególnie ważnym aspektem wyboru MongoDB jest wsparcie dla kolekcji typu *Time Series*, która jest szczególnie przydatna w przechowywaniu danych z szeregów czasowych – a takie dane będą eksportowane przez analizatory jakości energii elektrycznej.

## 2.3. Architektura systemu

### 2.3.1. MVC

MVC czyli Model-View-Controller to wzorzec architektoniczny, który jest często wykorzystywany w projektowaniu aplikacji internetowych. Pozwala on na osiągniecie jednej z kluczowych zasad dotyczących architektury system czyli separacji problemów. Cel stosowania wzorca MVC dobrze opisuje fragment „Podział na warstwy służy uporządkowaniu architektury systemu. Dzięki temu, że każda logiczna część jest od siebie oddzielona, zmiana w jednym miejscu, nie powoduje konieczności wykonywania lawinowej ilości zmian w innych miejscach systemu” [16]. Separacja jest realizowana przez podział aplikacji na komponenty: model, widok, kontroler [6]. Każdy z nich odpowiadający za konkretną sferę działania. Relacja ta przedstawiona na Rysunek 3.



Rysunek . Model wzorca MVC.

Model jest reprezentacją pewnego stanu aplikacji oraz logiki biznesowej która powinna być przez niego wykonywana. Logika biznesowa wraz z logiką utrzymującą stan aplikacji powinna być zawarta w modelu.

Widok jest odpowiedzialny za prezentację treści przez interfejs użytkownika. Logika znajdująca się w widokach powinna być związana jedynie z prezentacją treści. Widoki powinny prezentować użytkownikowi dane z modelu przekazanego przez kontroler. Ten element jest również odpowiedzialny za utrzymanie elementów z którymi użytkownik wchodzi w interakcję, czyli wskazanie metody i przekazaniu danych do kontrolera.

Kontroler odpowiada za komunikację między modelem i widokiem. Jest to element który reaguje na dane wejściowe i interakcje przekazane z widoku. Wywołuje modyfikacje modelu, dobiera widok który ma być wyświetlony użytkownikowi.

### 2.3.2. Wzorzec Repozytorium/Serwis

Wzorzec architektoniczny MVC w swoim podejściu jest dość ogólny, jego zasady opisują separację elementów prezentacji danych, manipulowania danymi oraz komunikacji między tymi elementami. Jednak taki ogólny podział często nie jest wystarczający, ponieważ zarządzanie danymi to często rozbudowany i skomplikowany kod, dlatego łączenie wzorców MVC, oraz Repozytorium/Serwis jest powszechną znaną praktyką w tworzeniu oprogramowania. Połączenie tych wzorców pomoże zapewnić organizację kodu, oraz pełną separację obowiązków. We wzorcu Repozytorium/Serwis możemy oddzielić warstwę prezentacji danych oraz komunikacji z modelem (odpowiednio widoków i kontrolerów) od dostępu do danych i logiki biznesowej aplikacji.

Warstwa dostępu do danych (DAL – data access layer) składa się z repozytoriów oraz powiązanych z nimi modeli domenowych. Warstwa ta ma za zadanie zapewnić bezpośredni dostęp do danych komunikując się z bazą danych. Pojedyncze repozytorium powinno zapewnić dostęp do pojedynczej encji danych. Biorąc pod uwagę wymagania stawiane systemowi przedstawione w rozdziale 2.1. Wymagania funkcjonalne i niefunkcjonalne, można stwierdzić, że aplikacja będzie składała się z repozytorium serii danych z analizatora, ale również takich jak repozytorium szablonów, punktów pomiarowych. Takie repozytorium składa się najczęściej z prostych operacji typu CRUD (ang. *Create, Read, Update, Delete*) czyli tworzenie, odczyt, modyfikacja, usuwanie elementów z kolekcji danych

Warstwa biznesowa składająca się z serwisów służy do operacji bardziej złożonych niż proste operację CRUD. W warstwie tej powinna znaleźć się główna logika i reguły związane z biznesowym procesem tworzonej aplikacji. Często zdarza się że jeden serwis korzysta z kilku repozytoriów, w przypadku złożonej logiki często jedna usługa wykorzystuje inną usługę, jednak należy mieć na uwadze, iż pojedyncza usługa powinna odpowiadać za operację powiązane z jednym modułem, najczęściej powiązana z pojedynczym widokiem. Ponownie analizując wymagania stawiane systemowi przedstawione w rozdziale 2.1. Wymagania funkcjonalne i niefunkcjonalne, można łatwo wyróżnić, że jedną z usług będzie moduł odpowiadający za szablony. Sam zapis szablonów, czyli operacje dodania, modyfikacji będą znajdować się w repozytorium, natomiast logika związana z walidacją, wyborem odpowiednich danych pomiarowych w szablonie, zakodowaniem ich do odpowiedniego formatu zajmować się będzie usługa. Takie podejście pozwala na spełnienie jednej z fundamentalnych zasad dobrych praktyk programowania obiektowego - *Single Responsibility Principle* – zasada pojedynczej odpowiedzialności.

Podział odpowiedzialności w architekturze z zachowaniem wzorca Repozytorium/Serwis dobrze obrazuje schemat przedstawiony na Rysunek 4.

Obraz zawierający tekst, diagram, zrzut ekranu, Czcionka

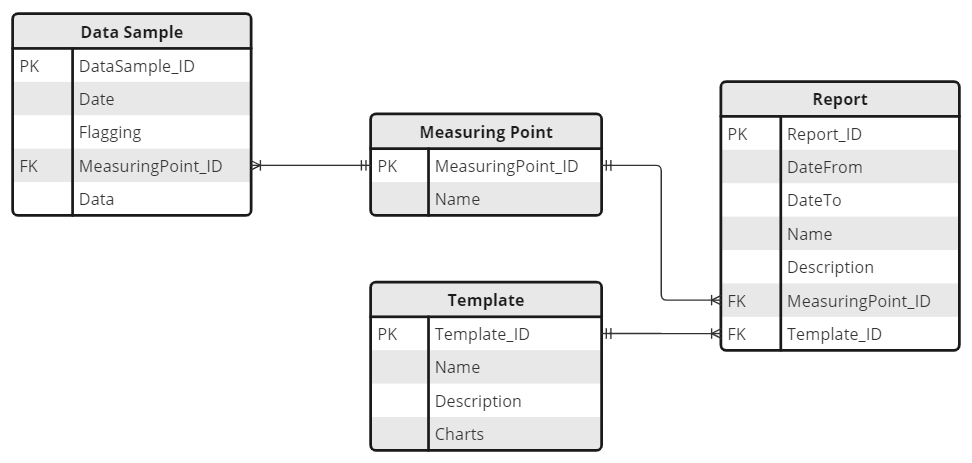
Opis wygenerowany automatycznie

Rysunek . Model wzorcu Repozytorium/Serwis.

## 2.4. Schemat bazy danych

Kluczowym aspektem realizowanego projektu jest analiza i porównanie różnych podejść baz danych. W pracy zostaną zbadane trzy modele: relacyjny z wykorzystaniem MS SQL, nierelacyjny oparty o MongoDB, oraz podejście hybrydowe wykorzystujące zalety obu tych systemów. We wszystkich wymienionych podejściach, modele bazy danych będą się nieznacznie różnić, jednak wszystkie będą składały się z następujących encji:

* **Próbki danych** (ang. *DataSample*)**:** Jest to główna encja, która będzie zajmować dominującą część bazy danych. To w tej encji znajdują się wszystkie próbki danych pomiarowych, oraz pomocnicze informację jak data i czas pomiaru, punkt pomiarowy, itp.
* **Szablon (**ang.*Template)***:** Zawiera informacje o szablonach takie jak nazwa, opis, czy informację o wykresach jakie mają być wygenerowane z użyciem tego szablonu.
* **Punkt pomiarowy** (ang. *MeasuringPoint)***:** Reprezentują poszczególne punkty pomiarowe, z której gromadzone są dane.
* **Raport** (ang. *Report)***:** Zawiera informację na temat wygenerowanych raportów, oraz sam raport w postaci danych binarnych.

Schemat tych encji oraz relacji między nimi przedstawia diagram EDR ukazany na Rysunek 5.

Rysunek . Schemat modelu relacyjnego bazy danych.

Dla modelu bazy nierelacyjnej, encje będą wyglądać niemalże identycznie, jednak będą to osobne kolekcje bez relacji.

# 3. Implementacja proponowanego rozwiązania

W niniejszym rozdziale, zostaną przedstawione kluczowe decyzje, które kształtowały proces implementacji, metodykę pracy, a także wyzwania i problemy napotkane podczas budowy aplikacji. Szczegółowo omówione zostaną zastosowania wskazanych technologii z poprzedniego rozdziału, oraz sposób, w jaki zaprojektowane narzędzia wpływają na efektywność zarządzania danymi. Poszczególny podrozdział, może być traktowany jako opis pojedynczego kontrolera, któremu w dalszym etapie będzie odpowiadać osobna sekcja w interfejsie użytkownika. Dzięki zastosowaniu wzorca *Dependency injection*, budowanie takich kontrolerów jest traktowane jak dołączanie odpowiednich modułów (serwisów i repozytoriów), co zostało zaprezentowanie w opisie poszczególnych kontrolerów w podrozdziałach 3.1. Import danych, 3.2. Dostęp do danych 3.3. Szablony, 3.4. Raporty

## 3.1. Import danych

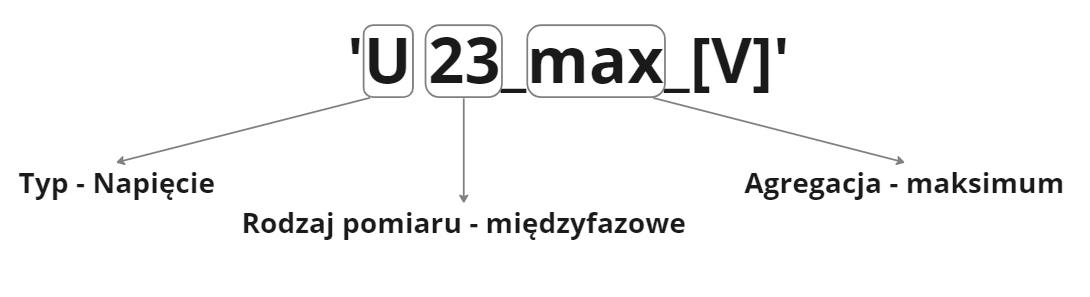
Pierwszym kluczowym etapem w procesie budowania aplikacji było stworzenie modułu odpowiedzialnego za import danych z pliku. Dla tej funkcjonalności postawiono następujące wymagania:

* Stworzenie modelu opisującego parametr pomiaru który w jednoznaczny sposób będzie go identyfikował, zarówno w bazie danych jak również w pozostałych modułach aplikacji.
* Automatyczne rozpoznawanie parametrów, na podstawie nagłówków znajdujących się w importowanych plikach.
* Zapewnienie efektywności i wydajności tego procesu. Dla importowanego pliku nie są przewidziane ograniczenia pod względem wielkości, dlatego należy zastosować mechanizmy optymalizujące import bardzo dużych plików.

Analizując nagłówki z przykładowych dokumentów wyeksportowanych z przykładowych analizatorów, można zauważyć, że występują wiele kombinacji parametrów. Tylko dla napięcia wyznaczono osiemnaście różnych parametrów, ponieważ może to być napięcie fazowe, międzyfazowe, wartości mogą być średnimi, minimami, maksimami itp. Podobnie jest dla prądu, mocy czynnej lub biernej. Ostatecznie możliwe jest uzyskanie setek różnych kombinacji parametrów, tworzenie dla każdego z nich osobnego typu wyliczeniowego, lub mapowania bazując na jakimś wspólnym identyfikatorze wydaje się skomplikowanym i mało efektywnym rozwiązaniem. Jednak zauważono, iż dla każdego z tych parametrów można wyznaczyć pomniejsze kategorie które w jednoznaczny sposób definiują dany parametr. W tworzonym systemie przyjęto trzy takie kategorie:

* Typ – definiuje główną kategorię pomiaru taką jak: napięcie, natężenie prądu, moc, częstotliwość, ale również informację o danej próbce danych takie jak data i czas pomiaru.
* Rodzaj pomiaru – definiuje jak został dokonany pomiar konkretnego typu. W tej kategorii znajdują się informacje które są bardziej specyficzne dla każdego typu, jednak można odnaleźć również części wspólne. Takie informacje to przykładowo, informacja czy pomiar dotyczy wielkości fazowej, międzyfazowej, całkowitej itp.
* Agregacja – ta kategoria definiuje rodzaj wartości danej próbki tj. czy dany parametr zawiera wielkości średnie, maksymalne, minimalne itp.

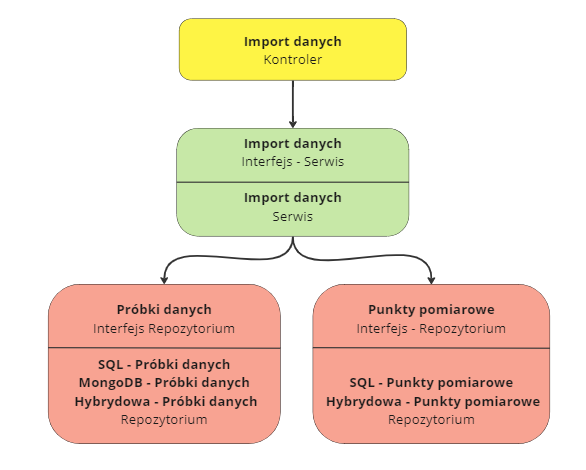
Wykorzystując tak przyjęte kategorię, można jednoznacznie opisać każdy z parametrów, wybierając nie z setek różnych możliwości a ustalając wartości dla tych trzech kategorii, z kilku możliwych opcji. Część z parametrów nie potrzebuje opisywać rodzaju pomiaru oraz rodzaju wartości, ponieważ typ zawiera również wartości unikalne jak czas pomiaru czy flaga decydująca czy pomiar był prawidłowy. Dla lepszego zrozumienia tej koncepcji przedstawiono przykład wraz z opisem na Rysunek 6.



Rysunek . Podział parametru na kategorie.

Do rozpoznania i przyporządkowania parametrów do odpowiednich kategorii, wykorzystano wyrażenia regularne (Regex). Wyrażenia regularne to sekwencje znaków tworzące wzorzec, który można wykorzystać do dopasowania, wyszukiwania czy podziału tekstu. Jest to potężne narzędzie w przetwarzaniu tekstów, które pozwala na wykonanie skomplikowanych operacji na ciągach znaków w sposób skuteczny oraz efektywny czasowo. W tworzonej aplikacji, stworzono trzy różne pule wyrażeń regularnych, co umożliwia precyzyjne dopasowanie do odpowiedniej wartości w każdej z przedstawionych wyżej kategorii.

Dla zapewnienia efektywności i wydajności procesu importu, podzielono, tę część na dwa etapy. W pierwszym etapie po wybraniu importowanego pliku na serwerze zostanie zapisany cały plik, aplikacja odczyta tylko nagłówki i kilka rekordów próbek danych, gdy system poradzi sobie już z tą czynnością użytkownikowi zostanie wyświetlony rezultat przypasowania nagłówków, zostanie udostępniona również możliwość edycji nagłówków. Drugi etap następuje po zaakceptowaniu przez użytkownika odczytanych nagłówków. W jego trakcie zostaną przekopiowane wartości z pliku zapisanego na serwerze do systemu bazy danych z odpowiadającym mu nagłówkiem. Czynność ta będzie odbywać się w tle, pozwalając użytkownikowi przejść do następnych widoków interfejsu użytkownika i rozpoczęcie przygotowywania szablonu lub raportu. Innym rozwiązaniem mającym zapewnić wydajność wskazanego procesu jest zastosowanie kopiowania wartości z pliku do bazy danych w paczkach. Schemat zawierający z jakich modułów składa się opisany kontroler przedstawiono na Rysunek 7.



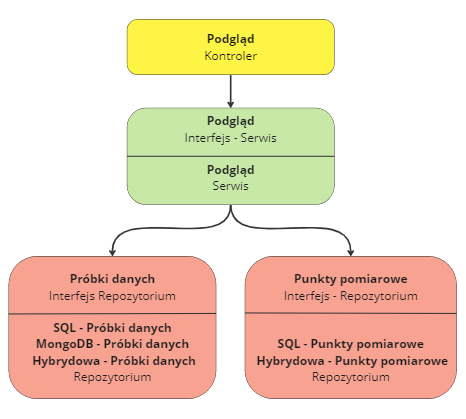
Rysunek . Schemat kontrolera importu danych.

## 3.2. Dostęp do danych

W procesie implementacji kluczową rolę odgrywa sprawny dostęp do danych. W celu zapewnienia elastycznego i optymalnego dostępu dla różnych podejść modeli bazodanowych zastosowano wzorzec projektowy wstrzykiwania zależności (*ang. Dependency injection, DI*). Polega on na dostarczaniu obiektowi jego zależności z zewnątrz, zamiast tworzenia ich wewnątrz obiektu. Zależności obiektu to sytuacje w których jeden obiekt do prawidłowego funkcjonowania używa innej klasy. We wzorcu repozytorium/serwis najczęściej spotykane są sytuacje kiedy kontroler używa serwisu, a serwis używa danego repozytorium. Takie podejście zwiększa możliwości testowania i rozbudowy dzięki odseparowaniu logiki tworzenia obiektów zależności od logiki biznesowej. Zapewnia również zmianę implementacji zależności bez konieczności modyfikacji kodu źródłowego tego konkretnego obiektu.

Jednym z wykorzystanych narzędzi w procesie budowania logiki dostępu do danych jest Entity Framework Core. Jest to typowy przedstawiciel narzędzi typu ORM (*ang. Object-Relational Mapping*), który umożliwia zarządzanie danymi relacyjnymi bez bezpośredniego pisania zapytań SQL. Dzięki niemu, operacje na danych stają się znacznie prostsze i czytelniejsze z perspektywy programisty, zachowując jednocześnie wysoką wydajność. Dla bazy nierelacyjnej, wykorzystano podobne narzędzie o nazwie MongoDB Driver. Jest to również narzędzie typu ORM stworzone do integracji z bazą danych MongoDB. Zapewnia on podobną funkcjonalność jak Entity Framework, jednak ze względu na architekturę nierelacyjnej bazy danych, operację te są jeszcze łatwiejsze i szybsze.

Kluczowym aspektem operacji na zbiorach danych w C# jest wykorzystanie technologii LINQ (*Language integrated Query).* Pozwala ona na pisanie zapytań bezpośrednio w języku C#. Dzięki kompatybilności zarówno z Entity Framework jak i MongoDB Driver, możliwe jest tworzenie zapytań do różnych źródeł baz danych tak, jakby były one kolekcjami w języku C#. Zapewnia to większą kontrolę i czytelność dostępu do bazy danych, ponieważ całość kodu z tym związana znajduje się w jednym miejscu. Ponadto kolejnymi zaletami tej technologii jest pełne wsparcie operacji asynchronicznych, co przy budowaniu interaktywnych interfejsów jest szczególnie ważne, ze względu na brak wstrzymywania aplikacji podczas wykonywania operacji. Jednak najważniejszą zaletą tej technologii jest leniwa ewaluacja, wg dokumentacji „odroczone wykonanie” [12] . Oznacza to, że operacja zostanie wykonana dopiero wtedy, gdy jej wyniki będą używane. Odroczona ewaluacja obiektu oznacza, że jego tworzenie jest opóźnione do momentu, gdy jest po raz pierwszy używany. Odroczona ewaluacja jest głównie używana do poprawy wydajności, unikania zbędnych obliczeń i zmniejszenia wymagań pamięci programu. Jest to możliwe m. in dzięki słowu kluczowemu języka C# *yield* które wskazuje że metoda lub operator w którym się pojawia, jest iteratorem, co powoduje że zwracany jest każdy element pojedynczo, oszczędzając pamięć, bez strat na wydajności. Odroczona ewaluacja wraz z możliwościami łączenia zapytań, wpływa pozytywnie na optymalizowanie zapytań przez silnik baz danych. Ponadto technologia LINQ, posiada ogromne wsparcie i jest mocno rozbudowana o czym może świadczyć około stu wbudowanych instrukcji takich jak: *select, where, any, join, intersect* i wiele innych. Rysunek 8 przedstawia schemat modułów składowych kontrolera podglądu danych. Technologia LINQ w języku C# może być zaimplementowana w dwóch notacjach. Pierwsza przypomina składnię języka baz danych, gdzie tworzone są kwerendy analogicznie jak w języku SQL, druga natomiast jest bliższa programowaniu obiektowemu, gdzie metody te są zaimplementowane jako metody na obiekcie kolekcji.



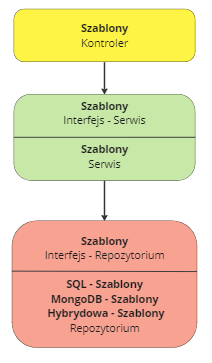
Rysunek . Schemat blokowy kontrolera podglądu danych.

## 3.3. Szablony

Na potrzeby obsługi szablonów, stworzono indywidualny kontroler. Umożliwia on tworzenie edycję i zarządzanie szablonami raportów. Jest to ważny moduł, dzięki któremu możliwe jest tworzenie spersonalizowanych raportów przedstawiających wybrane dane w postaci wykresów. Każdy z szablonów składa się z dwóch sekcji:

* Części stałej: Jest to część szablonu która jest niezmienna. Składa się ona z informacji o szablonie, tj. danych wprowadzonych przy generowaniu raportu takich jak zakres dat, czy nazwa punktu pomiarowego, notatki czym jest i jak został wygenerowany raport. Następnym fragmentem są kryteria oceny wskazane w *Rozporządzeniu Ministra Gospodarki,* oraz tabela z wynikami weryfikacji wskazanych danych z kryteriami. Do części stałej dołączono również podstawowe przebiegi takie jak wartości średnie, maksymalne, minimalne napięcia.
* Części edytowalnej: W tej sekcji użytkownik może zdefiniować dodatkowe przebiegi, które powinny znaleźć się w raporcie. Szablon pozwala na dodanie tytułu przebiegu i wskazanie parametrów które mają się na nim znaleźć.

Dzięki zastosowaniu szablonów, generowanie raportów staje się bardziej elastyczne i dostosowane do różnych potrzeb użytkowników. Kontroler obsługujący szablony został zaprojektowany w taki sposób, aby zapewnić niezbędną funkcjonalność, bez potrzeby integracji z innymi modułami. Wykorzystano jedyne narzędzie ułatwiające wskazanie danych które mają znaleźć się na przebiegach. Opis jak identyfikowane są badane wielkości został wskazany w rozdziale 3.1. Import danych na Rysunek 6. Schemat blokowy przedstawiający z jakich modułów składa się ten kontroler przedstawiono na Rysunek 9.



Rysunek . Schemat kontrolera szablonów.

## 3.4. Raporty

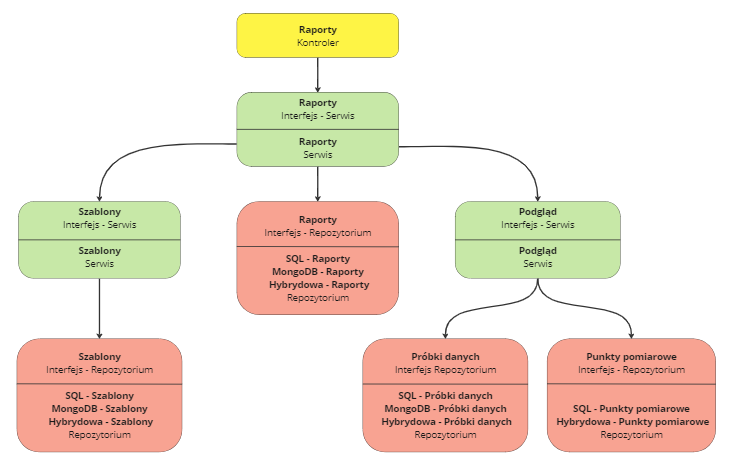
Osobny kontroler został również wydzielony dla kluczowej funkcjonalności tworzonej aplikacji webowej jaką jest tworzenie i obsługa raportów. Jest to część systemu, której zadaniem jest:

* Pobieranie z bazy danych próbek danych
* Sprawdzenie zgodności wybranego zbioru parametrów z kryteriami zdefiniowanymi w wymogach *Rozporządzenia Ministra Gospodarki*.
* Tworzenie przebiegów zdefiniowanych przez użytkownika w szablonie.
* Zarządzanie raportami, czyli operacje takie jak podgląd wygenerowanego raportu w przeglądarce, pobranie go na dysk, oraz przechowywanie w bazie danych.
* Obsłużenie serwera SMTP, umożliwiając przy tym wysyłkę wygenerowanego raportu bezpośrednio na pocztę wskazaną przez użytkownika

Wygenerowany raport przechowywany jest w bazie danych. Raporty przechowywane są w bazie w postaci plików binarnych, zapewniając oryginalny format i strukturę pliku, bez ryzyka utraty jakości czy zniekształceń powodowanych konwersją lub kompresją. Eliminuje to również potrzebę konwersji do formatów tekstowych, obniżając przy tym zajętość pamięci. Ponadto dane binarne umożliwiają bezpośredni zapis i odczyt bez dodatkowego przetwarzania, dzięki temu takie operacje są znacznie szybsze.

Do wysyłki maili, wykorzystano serwer SMTP, co jest uniwersalnym rozwiązaniem, zapewniającym prostą, szybką i niezawodną wysyłkę wiadomości e-mail. W pracy wykorzystano serwer Gmail, który w wersji darmowej umożliwia pełną funkcjonalność dla celów innych niż komercyjne.

W tym kontrolerze natomiast niezbędne okazało się dołączenie, większości modułów opisanych w tym rozdziale, ponieważ korzysta i jest bezpośrednio zależny od takich encji jak próbki danych oraz szablony. Schemat połączeń serwisów i repozytoriów przedstawiono na rysunku Schemat zawierający z jakich modułów składa się ten kontroler przedstawiono na Rysunek 10.



Rysunek . Schemat kontrolera Raportów.

# 4. Interfejs użytkownika

W dobie rozwoju rozwiązań webowych, a co za tym idzie dostępnej różnorodności, użytkownik został przyzwyczajony do nowoczesnych, estetycznych interfejsów użytkownika. To właśnie interfejs decyduje o pierwszym wrażeniu i ogólnym odbiorze aplikacji przez użytkownika. Dlatego ważne jest, aby interfejs zapewniał przyjazny dla oka design, ale również był prosty w obsłudze, oraz pozwalał na dynamiczną komunikację z użytkownikiem. Aby spełnić te wymagania, nie wystarczy dostarczenie interfejsu opierającego się o domyślnie stylizowane pola tekstowe, czy przyciski. Dążąc do stworzenia interfejsu, który będzie funkcjonalny i interaktywny, należy dostarczyć, nowocześnie wystylizowane obiekty, zapewnić animacje i responsywność wszelkiego rodzaju pól, przycisków, list. W proponowanym rozwiązaniu, poza możliwościami które oferują widoki z silnika Razor, wykorzystano również najczęściej stosowane rozwiązanie składające się z trzech elementów: HTML, CSS, oraz JavaScript.

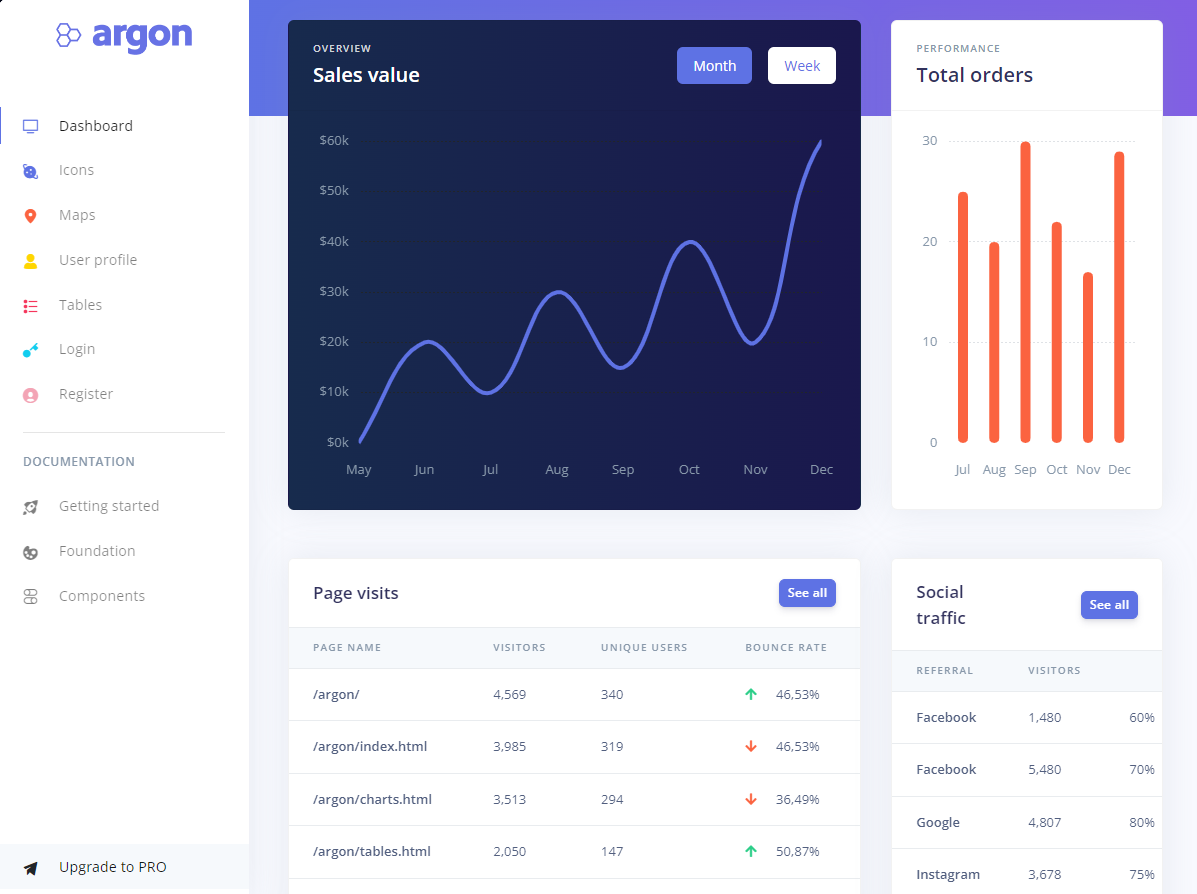
HTML jest językiem opisowym, odpowiadającym za główną strukturę wyświetlanych treści. Definiuje on takie elementy jak nagłówki, paragrafy, elementy nawigacji. CSS jest to język stylów, który opisuje prezentację dokumentów HTML, Pozwala on kontrolować wygląd, poprzez ustawianie kolorów, czcionek, odstępów czy kształtów elementów treści opisanych w HTML. Zastosowanie obu tych języków pozwala w łatwy sposób oddzielić strukturę od zarządzania stylami, co znacznie upraszcza konstruowanie części wizualnej aplikacji internetowych. Uzupełnieniem dla tych dwóch języków jest JavaScript, jest to dynamicznie typowany język programowania którego skrypty znajdują się po stronie klienta umożliwiając natychmiastową reakcję, bez potrzeby komunikacji z serwerem. Dzięki zastosowaniu JavaScript możliwe jest dynamiczne manipulowanie strukturą dokumentu HTML, zmienianie treści, reagowanie na zdarzenia użytkownika, komunikacja z serwerem.

W tworzonej aplikacji wykorzystano bibliotekę JavaScript o nazwie JQuery. Jest to łatwo rozszerzalna, biblioteka typu open-source upraszczająca podstawowe wykorzystanie języka JavaScript. Najważniejszymi zaletami JQuery jest uproszczenie manipulacji DOM (document object model), czyli znajdowania zmieniania i ukrywania elementów dokumentu HTML. Wiele najczęściej wykorzystywanych zdarzeń na interfejsie użytkownika takich jak kliknięcia w przycisk, ukrycia, odkrycia elementów czy wyświetlanie komunikatów, zostały skrócone do postaci krótkich jednolinijkowych wyrażeń. Kolejną ogromną zaletą jest uproszczenie korzystania z AJAX, która jest zbiorem technik umożliwiających interakcje użytkownika z serwerem, bez konieczności przeładowywania strony HTML. Pozwala to na dynamiczne aktualizowanie treści na stronie.

## 4.1. Argon Dashboard

Stworzenie nowoczesnego i atrakcyjnie wyglądającego interfejsu użytkownika stanowi spore wyzwanie, które wymaga nie tylko technicznego doświadczenia, ale również wyczucia estetyki. Celem uproszczenia tego procesu i zagwarantowania spójności wyglądu w projekcie zdecydowano się wykorzystać szablon interfejsu *Argon Dashboard*. Jest to gotowy zestaw stylów CSS, komponentów HTML. Oraz skryptów JavaScript, który został zaprojektowany, aby wspierać twórców w szybkim tworzeniu atrakcyjnych i funkcjonalnych interfejsów użytkownika. Szablon ten został udostępniony na zasadach licencji MIT, co za tym idzie można go wykorzystywać zarówno w projektach prywatnych i komercyjnych. Szablon interfejsu *Argon Dashboard* wykorzystuje również popularny framework CSS Bootstrap [10]. Udostępnia on zestaw narzędzi, stylów które pozwala na szybkie i efektywne projektowanie interfejsów użytkownika. Jego komponenty opierają się na wielu dodatkowych bibliotekach JavaScript, jednak na potrzeby budowanej aplikacji niezbędne okazało się wykorzystanie jedynie bibliotek:

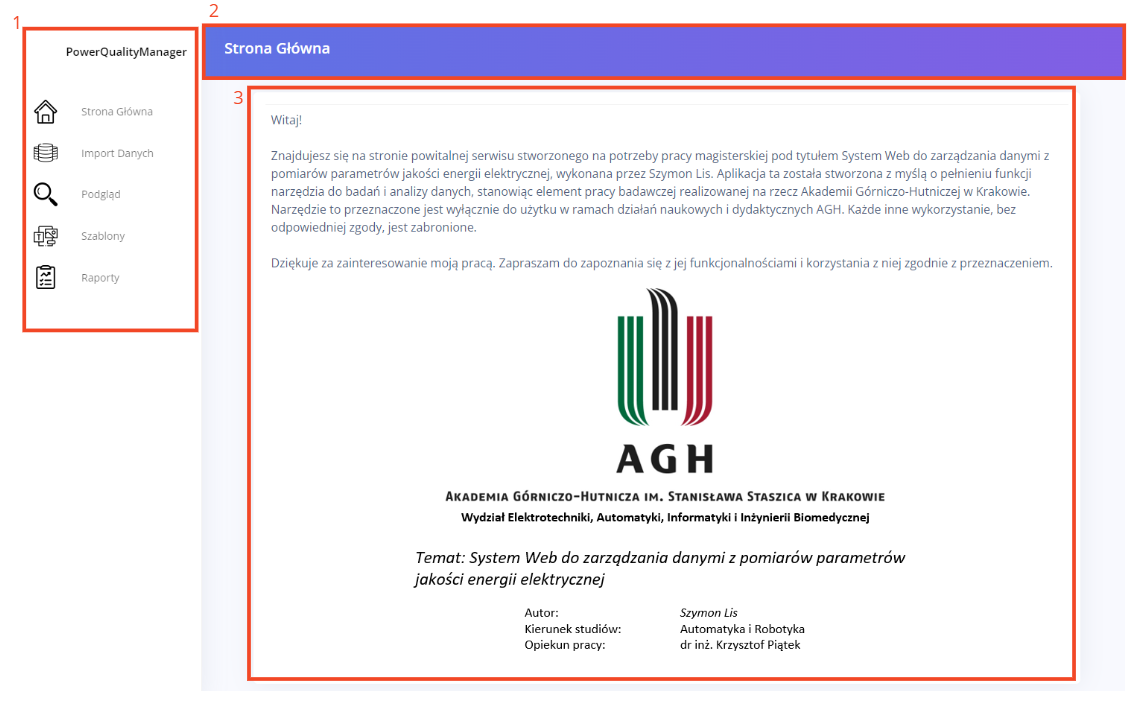
* Flatpickr – prosty, interaktywny selektor dat
* Datatables – biblioteka wspomagająca budowanie rozbudowanych tabel z danymi



Rysunek . Przykładowy interfejs zbudowany z wykorzystaniem Argon Dashboard.

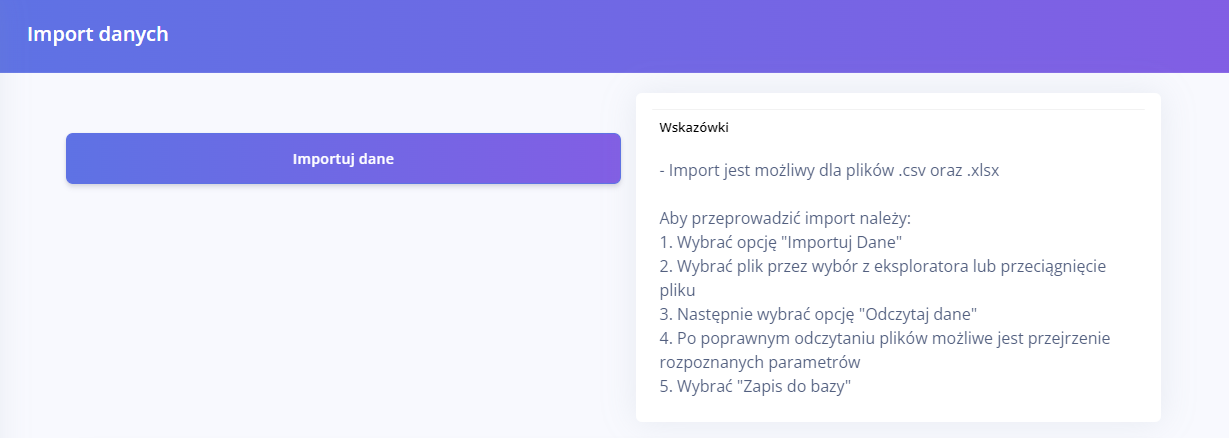
## 4.2. Opis funkcjonalności

W tym podrozdziale przedstawiono elementy i funkcje dostępne w interfejsie użytkownika aplikacji. Ukazanie tych aspektów, pozwoli poznać możliwości oferowane przez system. Opisane również zostaną rozwiązania które zapewniają spełnienie wymagań postawionych w podrozdziale 2.1. Wymagania funkcjonalne i niefunkcjonalne



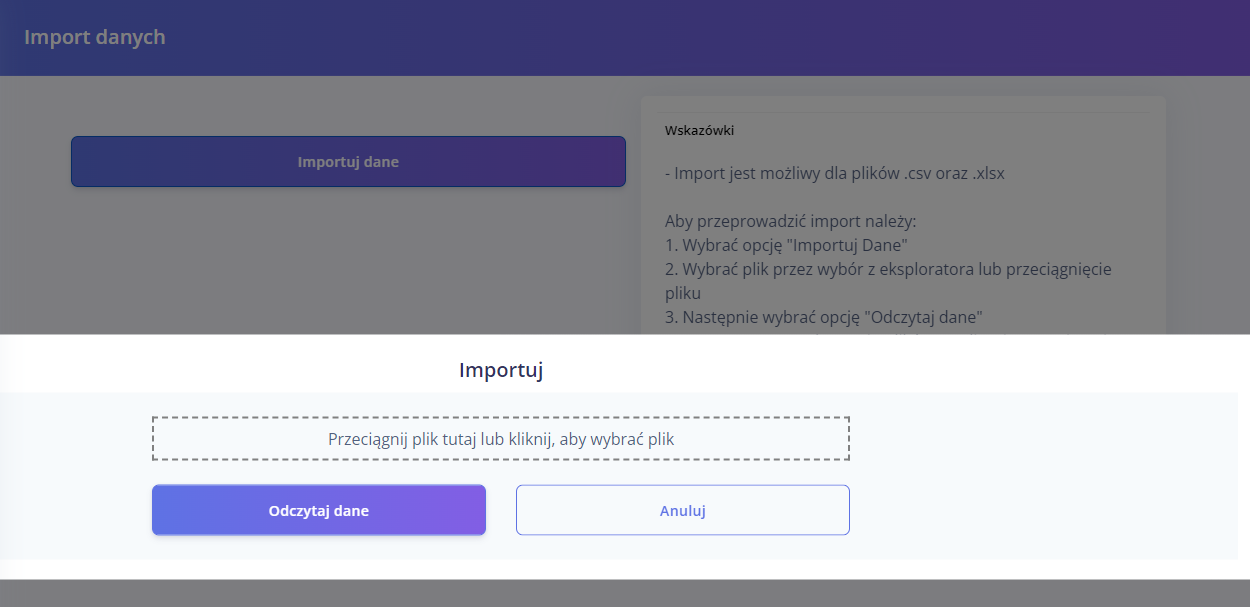
Rysunek . Interfejs użytkownika – Strona główna.

Na Rysunek 12. pokazane zostały główne elementy interfejsu. Każda z podstron aplikacji *PowerQualityManager* składa się dokładnie z tych trzech elementów, jednak zmienia się treść w poszczególnych elementach. Elementem nr 1 oznaczono boczny pasek nawigacyjny, pasek ten przy mniejszych oknach wyświetlania automatycznie zwija się do boku nie zasłaniając treści głównej. Na pasku tym znajdują się przekierowania do poszczególnych funkcji aplikacji. Każda z funkcji reprezentuje osobny kontroler, dokładniej opisany w podrozdziałach 3.1. Import danych, 3.2. Dostęp do danych, 3.3. Szablony, 3.4. Raporty. Ponadto dołączona została strona główna zawierająca statyczne informacje o celu powstania i przeznaczeniu aplikacji, jej dokładna treść została pokazana na Rysunek 12. Element nr 2 to pasek górny, zawierający jedynie nagłówek z nazwą podstrony wyświetlanej użytkownikowi. Element nr 3. Zawiera treść główna dla poszczególnej podstrony. Aby skupić się jedynie na kluczowych elementach interfejsu, w dalszej części pracy fragmenty interfejsu zostaną ukazane bez bocznego paska nawigacji (element nr 1), ponieważ jest on identyczny w całej aplikacji.



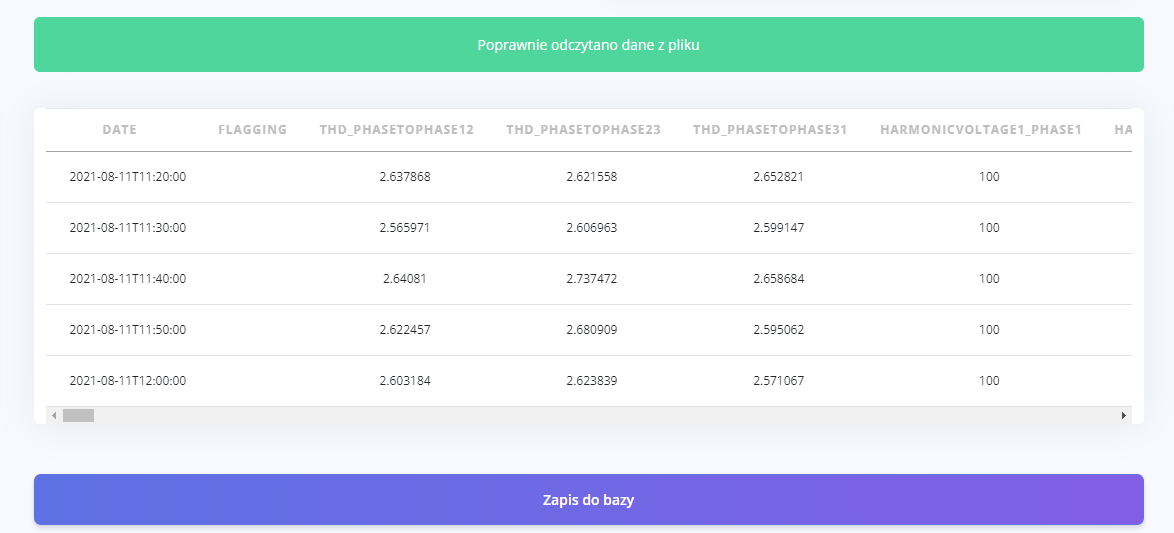
Rysunek . Interfejs użytkownika - import danych.

Strona umożliwiająca import danych została przedstawiona na Rysunek 13. Na każdej z podstron została umieszczona krótka instrukcja postępowania, z informacjami ogólnymi dla danej funkcjonalności.



Rysunek . Interfejs użytkownika - okno wyboru importowanego pliku.

Do wyboru pliku udostępniono zarówno klasyczną metodę wyboru pliku przez eksplorator plików, który jest uruchamiany po kliknięciu we wskazane pole, lub popularną metodę typu drag and drop, umożliwiając „przeciąganie” pliku z urządzenia do wskazanego pola. Okno wyboru wskazanego pliku przedstawiono na Rysunek 14.



Rysunek . Interfejs użytkownika - odczytane nagłówki.

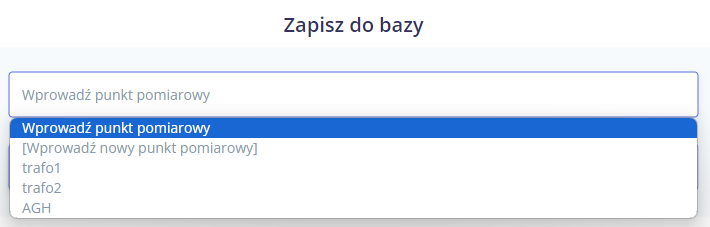
Po wybraniu i zaimportowaniu pliku rozpoczyna się odczytywanie nagłówków zgodnie z procedurą opisaną w podrozdziale 3.1. Import danych Po zakończeniu odczytywania wyświetlony zostanie komunikat z informacją o pozytywnym, lub negatywnym odczytaniu nagłówków, oraz tabela zawierająca odczytane nagłówki oraz kilka wierszy przykładowych danych, co pozwoli użytkownikowi na weryfikację poprawności odczytanych nagłówków. Przykładowy rezultat przedstawiono na Rysunek 15. Interfejs użytkownika - odczytane nagłówki

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

Rysunek . Interfejs użytkownika - edytor nagłówków.

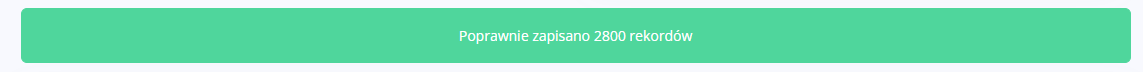
Dzięki wprowadzeniu opisu parametrów złożonego z trzech oddzielnych kategorii, przedstawionego na Rysunek 6, możliwe jest optymalne tworzenie modyfikowanie nazwy parametru. Opierające się o wybranie wartości z trzech niewielkich list. Takie rozwiązanie jest zarówno szybsze jak i bardziej intuicyjne dla użytkownika niż wybieranie spośród listy kilkuset różnych kombinacji nazw parametrów, lub ręcznym wpisywaniu nazwy. Okno tworzenia nazwy parametru ukazano na Rysunek 6. Podział parametru na kategorie.



Rysunek . Interfejs użytkownika - zapis do bazy.

Proces importu danych kończy się wybraniem punktu pomiarowego z listy punktów pomiarowych znajdujących się w bazie, lub dodaniem nowego punktu pomiarowego. Przykładowa lista z wyborem punktu pomiarowego została ukazana na Rysunek 17. Interfejs użytkownika - zapis do bazy Po potwierdzeniu, rozpoczyna się proces importu, który ze względu na sporą ilość danych jest uruchamiany jako zadanie w tle, nie blokując innych operacji na interfejsie użytkownika.

Po zakończeniu operacji importu wyświetlony zostanie komunikat niezależnie od podstrony na której znajduje się użytkownik o sukcesie bądź błędzie importu. Przykładowy komunikat wyświetlono na Rysunek 18. Interfejs użytkownika – komunikat



Rysunek . Interfejs użytkownika – komunikat.

Następna zakładka oferuje podgląd danych zapisanych w bazie filtrując po punkcie pomiarowym oraz dacie początku i końca. Dla pola w formularzu dotyczącego punktu pomiarowego zastosowano, rozwijaną listę z wyborem, natomiast dla wyboru dat, zastosowano interaktywny kalendarz z biblioteki flatpickr zawartej w szablonie interfejsu ArgonDashboard. Dodatkowo na kalendarz nałożone są ograniczenia, które pozwalają na wybranie dat które znajdują się w bazie danych dla wskazanego punktu pomiarowego. Tutaj wykorzystano dynamiczne zapytania na podstawie wybranego punktu pomiarowego z wykorzystaniem technologii AJAX oraz odpowiednio przygotowanej logice po stronie backendu. Przykładowe dane wraz z opcjami podstrony podglądu danych zostały ukazane na Rysunek 19. Interfejs użytkownika –

Obraz zawierający tekst, zrzut ekranu, numer, oprogramowanie

Opis wygenerowany automatycznie

Rysunek . Interfejs użytkownika – podgląd.

Następna z sekcji obejmuje narzędzia i funkcje, które pozwalają na zarządzanie szablonami w systemie. Użytkownik ma możliwość dodania, edycji oraz usuwania szablonów. Warto podkreślić, że jest jeden specyficzny szablon bazowy, który służy jako podstawa dla wszystkich pozostałych. Jest to element stały – nieedytowalny oraz nieusuwalny. Widok podstrony obejmującej obsługę szablonów przedstawiono na Rysunek 20.

**Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie**

Rysunek . Interfejs użytkownika – szablony

Do dodawania oraz edycji stworzono dedykowany formularz składający się z unikalnej nazwy szablonu według której będzie identyfikowany w bazie danych, opisu, który może być wskazówką dla innych użytkowników zawierająca informację o celu i przeznaczeniu szablonu. Ostatnia część formularza szablonu, obejmuje tabelę z nazwą wyświetlaną dla poszczególnych wykresów, oraz informacją jakie parametry mają się znajdować się na takim szablonie. Tutaj przyjęto uproszczenie względem użytkownika, polegające na domyślnym zawarciu przebiegów ze wszystkich dostępnych faz dla danego parametru wraz z legendą. Możliwe jest również ręczne wskazanie konkretnej fazy przez dopisanie cyfry wskazującej na daną fazę. W tym przypadku zastosowano osobne okno, pozwalające na wygenerowanie nagłówka. Narzędzie to przedstawiono na Rysunek 16, umożliwia ono wskazanie parametrów które mają być wyświetlone na przebiegu. Widok formularza dodawania/edycji szablonów został przedstawiony na Rysunek 21.

**Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie**

Rysunek . Interfejs użytkownika - formularz dodawania/edycji szablonu.

Ostatnia z sekcji interfejsu użytkownika obejmuje generowania i zarządzanie raportami. Strona ta zawiera przycisk pozwalający na stworzenie nowego raportu, oraz listę z już wygenerowanymi raportami, w której po wskazaniu raportu pojawiają się opcję takie jak podgląd raportu w przeglądarce, pobranie na dysk, usunięcie lub wysyłka wiadomość e-mail z raportem. Widok z tej części interfejsu, wraz z przykładowymi danymi przedstawiono na .

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Rysunek . Interfejs użytkownika – Raport.

Sekcja raportów obejmuje dodatkowo dwa formularze, pierwszy z nich obejmuje wygenerowanie nowego raportu. Składa się on z pól takich jak nazwa, nazwa wykorzystywanego szablonu, data rozpoczęcia oraz zakończenia oraz wskazanie punktu pomiarowego. Logika obejmująca ułatwiania wprowadzania danych została przedstawiona na Rysunek 17. Interfejs użytkownika - zapis do bazy, oraz Rysunek 19. Interfejs użytkownika – podgląd. Wzór takiego formularza ukazano na Rysunek 23. Następnym jest formularz wysyłki wiadomości e-mail. Zawiera on pola tekstowe obejmujące adresata, tytuł oraz treści wiadomości. Załącznik jest automatycznie dodawany w zależności od wskazanego raportu (Rysunek 22). Przykład formularza obejmującego wysyłkę wiadomości e-mail został pokazany na Rysunek 24.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Rysunek . Interfejs użytkownika - formularz generowania nowych raportów.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rysunek . Interfejs użytkownika - formularz wysyłki wiadomości e-mail.

## 4.3. Raport

Raport jest dokumentem w formie PDF, tworzony jest automatycznie, co wynika z potrzeby generowania raportów dostosowanych do wymogów użytkownika. Elementy statyczne, takie jak teksty czy stałe opisy przechowywane są w plikach .resx. Są to pliki zasobów, szeroko wykorzystywane w technologii .NET. Opierają się o format XML, oraz definiują wartości w parach typu nazwa/wartość. Mogą być tworzone oraz modyfikowane zarówno z poziomu kodu oraz edytora plików [11]. Takie podejście umożliwia centralne zarządzenie tekstem, lub adaptacje aplikacji do różnych języków bez konieczności modyfikacji kodu aplikacji. Generowany raport składa się z czterech sekcji:

* Strony tytułowej z informacjami o wygenerowanym raporcie
* Definicji i opisu norm
* Wynikami spełnienia norm dla wskazanych danych
* Przebiegami parametrów wskazanych przez użytkownika w szablonie

Strona tytułowa oraz definicja norm to statyczne teksty, wycinki z tych części raportu przedstawiono na Rysunek 25, Rysunek 26.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Rysunek . Raport - strona tytułowa.

Obraz zawierający tekst, zrzut ekranu, Czcionka, Równolegle

Opis wygenerowany automatycznie

Rysunek . Raport - opis norm.

Normy te zaczerpnięto z [1] opisują one parametry jakościowe energii elektrycznej dla sieci funkcjonującej bez zakłóceń. Przedstawione wyżej wymogi są stawiane grupom przyłączeniowym III – VI.

Wyniki zgodności, zostały natomiast przedstawione w formie tabelarycznej. Gdzie kolumnami jest odpowiednio nr normy, wynik, oraz powód braku zgodności, wiersze natomiast to kolejne normy, rozbite ewentualnie na poszczególne fazy w przypadku danych z więcej niż jednej fazy. Przykładowe wyniki wraz ze wzorem tabeli przedstawiono na .

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Rysunek . Raport - wyniki zgodności z normą.

Poszczególne parametry na przebiegach prezentowane są pojedynczo, łączone są jedynie przebiegi dotyczące tej samej wartości dla różnych faz, przykładowy przebieg przedstawiony został na . Jednostki dla poszczególnych parametrów nie są zapisywane w bazie danych, oraz nie zawsze są zawarte w dostarczanych nagłówkach. Dla dostarczonych danych podczas importu następuje konwersja dla jednostek innych niż podstawowe – dotyczy to głównie przedrostków. Dlatego, jeśli dostarczono jednostki w nagłówkach pliku przebiegi prezentowane są zawsze w jednostce podstawowej dla danego parametru.

Obraz zawierający tekst, linia, Wykres, zrzut ekranu

Opis wygenerowany automatycznie

Rysunek . Raport - przykładowy przebieg.

# 5. Porównanie modeli bazy danych

W niniejszym rozdziale przedstawiono porównanie różnych podejść do przechowywania danych: relacyjnego, nierelacyjnego oraz hybrydowego. Celem takiego porównania jest zrozumienie, jakie zalety i wady niesie ze sobą każde z tych podejść. Pierwszym elementem porównania będzie analiza, jak każdy z modeli wpływa na łatwość tworzenia oraz integracji z punktu widzenia programisty. W dalszej części zostanie przeprowadzony eksperyment mający na celu doświadczalne porównanie wydajności stosowanych podejść. Konkretnie zbadane zostaną czasy zapisu oraz odczytu w różnych scenariuszach operacyjnych. Takie podejście pozwoli na pełniejsze zrozumienie, jakie korzyści oraz wyzwania mogą wynikać z wyboru modelu bazy danych.

## 5.1. Założenia teoretyczne

Relacyjna baza danych jest rodzajem bazy danych która przechowuje, oraz operuje na danych za pomocą tabel oraz relacji między nimi. Model ten powstał już w latach 70, szybko stając się dominującym modelem bazy danych. Dane w relacyjnej bazie danych są uporządkowane i zorganizowane, co skraca znacząco skraca czas przeszukiwania takiej bazy danych. Każda tabela musi składać się z klucza głównego który w jednoznaczny sposób identyfikuje wiersz tabeli. Natomiast relacje pomiędzy tabelami definiowane są przy użyciu kluczy obcych. Warto zwrócić uwagę na fakt, że tabela w relacyjnej bazie danych, posiada z góry zdefiniowany schemat. Oznacza to, że dane zapisywane do tabeli mogą składać się tylko z wyznaczonych w schemacie kolumn oraz być w odpowiadającym tej kolumnie typie danych. Do zarządzania i manipulacji danymi w relacyjnych bazach danych, służy język SQL. Jest to uniwersalny język obsługiwany przez większość wiodących systemów zarzadzania relacyjnymi bazami danych. Umożliwia on szeroki zakres operacji na bazie danych, oraz zapewnia dodatkowe bezpieczeństwo poprzez możliwości zarządzania uprawnieniami dostępu. Nowoczesne systemy zarządzania relacyjnymi bazami danych posiadają również tzw. optymalizatory zapytań, które analizują zapytania i zapewniają efektywny sposób na ich wykonanie, między innymi dzięki indeksowaniu czy buforowaniu. Biorąc pod uwagę założenia systemu, szczególnie pomocne mogą okazać się wspomniane już indeksy. Są to pomocnicze struktury danych, które pomagają lokalizować dane dla wskazanej kolumny również innej niż klucz główny. Dzięki indeksom znalezienie rekordów spełniających wskazane warunki staje się dużo szybsze, ponieważ indeks zapewnia dodatkowe sortowanie tabeli, co pozwala na uniknięcie pełnego skanu tabeli. Takie podejście do baz danych jest również obarczone pewnymi wadami. Relacje, zdefiniowany schemat czy indeksy znacząco przyspieszają wyszukiwanie i odczyt danych, natomiast są dużym obciążeniem przy dodawaniu danych do bazy, ponieważ poza samym procesem dodawania rekordów do bazy, system zarządzania bazami danych musi stworzyć dodatkowe struktury jak relacje czy indeksy. Taka operacja w przypadku wielu indeksów jest stosunkowo kosztowna, ponieważ dla każdego indeksu konieczne jest przeszukanie indeksów, dopasowanie dodawanego rekordu oraz wszystkie pozostałe indeksy muszą zostać zaktualizowane. Operacje te również będą coraz to bardziej kosztowne wraz ze zwiększającą się ilością rekordów. Takie skalowanie nazywamy wertykalnym, oznacza to że wraz z powiększaniem się bazy danych, niezbędne jest zwiększanie zasobów serwera utrzymującego taką danych takich jak RAM czy moc obliczeniowa. Co oczywiście przy ogromnych bazach danych stanowi spory problem ponieważ takie skalowanie posiada swój fizyczny limit.

Nierelacyjna baza danych jest rodzajem bazy danych która nie korzysta z klasycznego schematu tabelarycznego. Zamiast tego wykorzystywane są schematy dokumentowe takie JSON, XML, schematy typu klucz/wartość, czy schematy grafowe składające się z krawędzi i wierzchołków. W pracy wykorzystano bazę danych MongoDB, która jest przedstawicielem bazy danych o schemacie dokumentowym. Oznacza to że zestaw danych składa się pól ciągów i wartości w jednostce, która jest nazywana dokumentem. Wykorzystuje ona dokumenty typu BSON (*ang. Binary JSON*). Jest to odmiana typu JSON, która dzięki zastosowaniu kodowania binarnego jest nieznacznie szybsza. Format BSON został również rozszerzony o typ danych reprezentujący znacznik czasu (data i czas). Nierelacyjne bazy danych są również bardziej elastyczne jeśli chodzi o przechowywane dane, nie posiadają one sztywnego schematu przechowywanych danych. Pozwala to na przechowywanie danych o różnych strukturach, ważne jest tylko żeby klucze przechowywanych danych były unikalne w obrębie pojedynczego dokumentu. Nierelacyjne bazy danych posiadają również odpowiednik klucza głównego który jest unikalny i według którego system zarządzania bazą danych lokalizuje przechowywane dane. Dlatego odczyt danych po jego identyfikatorze jest bardzo szybki dla nierelacyjnych baz danych. Natomiast wydajność pobierania danych znacząco spada, gdy niezbędne jest przefiltrowanie danych po innej wartości, z czym dobrze radzą sobie bazy relacyjne. Domyślnym identyfikatorem kolekcji w przypadku bazy MongoDB jest typ o nazwie *ObjectId* – jest to ciąg 12 bajtów w którym zakodowane są takie informacje jak data utworzenia, unikalny identyfikator maszyny, procesu, oraz części losowej zapewniającej unikalność. Takie podejście pozwala systemowi zarządzania bazą danych na zoptymalizowane wyszukiwanie danych. Ponadto MongoDB oferuje również kolekcję typu *Time Series [17]*, dla której domyślny identyfikator *ObjectId* zastąpiony jest znacznikiem czasu, co wydaje się dobrym podejściem do przechowywania danych składających się z serii danych z analizatora jakości energii elektrycznej. Brak struktur takich jak relacje czy indeksy, pozwala przechowywać dane w postaci kolekcji ze zoptymalizowanym dostępem pod kątem określonych wymagań. Dzięki takiemu podejściu, powiększająca się baza danych może zostać zoptymalizowana nie przez zwiększenie wydajności pojedynczego sprzętu, a może zostać rozłożona na węzły zarządzające tylko fragmentem kolekcji. Takie podejście nazywane jest skalowaniem horyzontalnym, zdecydowanie bardziej pożądane w nowoczesnych rozwiązaniach które zbierają, oraz przechowują coraz to większe zbiory danych.

## 5.2. Badane podejścia

W opisanym systemie web, przeprowadzono analizę różnych podejść do bazy danych. Każde z nich prezentuje nieco odmienne podejście do przechowywania i organizacji danych. Jak również każdy z nich może odpowiadać na bardziej specyficzne potrzeby i wymagania. Omówienie oraz porównanie tych modeli pozwoli wskazać zalety oraz potencjalne ograniczenia. Wymagania jakie dane muszą być przechowywane w bazie danych zostały przedstawione w podrozdziale 2.4. Schemat bazy danych.

### 5.2.1. Model relacyjny

Pierwszy z prezentowanych modeli to model opierający się w pełni o relacyjną bazę danych jaką jest MS SQL. Dokładny schemat relacyjnej bazy danych został przedstawiony na Rysunek 5. Schemat modelu relacyjnego bazy danych. Ze względu na specyfikę danych z analizatora, nie jest możliwe rozłożenie ich na pomniejsze relacje. Najczęściej będzie to rekord z dziesiątkami, lub setkami parametrów. W takim podejściu pierwszym problemem staje się schemat tabeli przechowywującej serie danych pomiarowych z analizatora. Jak zostało opisane w rozdziale 3.1. Import danych, można wyróżnić setki różnych parametrów, a relacyjna baza danych wymaga stworzenia tabeli reprezentującej je wszystkie wraz z odpowiednim typem danych. W tworzonej aplikacji webowej w schemacie ograniczono się do kolumn które występowały przynajmniej raz w przykładowych danych z analizatorów. Reszta z nich jest serializowana do formatu pliku XML i umieszczana w jednej kolumnie, tak aby w razie potrzeb był do nich dostęp, ale nie obciążały dodatkowo procesu pobierania danych. W takim podejściu wymagane jest również zdefiniowanie klasy reprezentującej każdą z kolumn w tabeli, co byłoby procesem czasochłonnym jednak dzięki Entity Framework możliwe jest wygenerowanie takich klas automatycznie na podstawie schematu dostarczonej tabeli. Dla pozostałych danych przechowywanych w bazie takich jak punkty pomiarowe, czy szablony nie ma takiego problemu, są to raczej niewielkie tabele, z prostymi konstrukcjami.

### 5.2.2. Model nierelacyjny

Kolejnym modelem bazy danych, jest model opierający się jedynie o nierelacyjną bazę MongoDB. Na potrzeby takiego modelu stworzono kolekcje analogiczne do tych opisanych w rozdziale 3.1. Import danych. W podejściu nierelacyjnym natomiast napotkano problem, gdy dla danego znacznika czasu oraz punktu pomiarowego dokonywany jest ponowny zapis z innymi parametrami. Z perspektywy funkcjonalności opisanej w wymaganiach 2.1. Wymagania funkcjonalne i niefunkcjonalne, system powinien obsłużyć import różnych parametrów dla jednego znacznika czasu nawet jeśli pochodzą one z różnych plików. Dlatego aby rozwiązać ten problem, oraz zachować wszystkie importowane parametry, dane te są zapisywane w osobnych dokumentach. Natomiast aby zapewnić spójność wyliczania parametrów jakościowych przy generowaniu raportu, takie dokumenty podczas pobierania z bazy danych są łączone. Parametry występujące tylko w jednym dokumencie są dopisywane, natomiast dla parametrów występujących w kilku dokumentach do wyliczeń pobierane są te które zostały później dodane do bazy danych.

### 5.2.3. Model nierelacyjny z wykorzystaniem kolekcji typu *Time Series*

Ten model jest niemal identyczny jak model nierelacyjny przedstawiony w podrozdziale 5.2.2. Model nierelacyjny z jedyną różnica, że nie posiada kluczowego dla kolekcji MongoDB pola \_id, zamiast niego przy tworzeniu kolekcji wskazano, które pole będzie reprezentować znacznik czasu. Co ważniejsze znacznik ten nie musi być unikalny względem całej kolekcji, dlatego można zastosować po prostu znacznik czasu który jest dołączony wraz z parametrami.

### 5.2.4. Model hybrydowy

Ostatnim zaprezentowanym podejściem jest model hybrydowy, wykorzystujący zarówno relacyjną bazę danych MS SQL, oraz nierelacyjną MongoDB. W podejściu tym wykorzystano mocne strony obu baz danych, minimalizując przy tym wady poprzez przeniesienie słabszych aspektów na drugą z nich. Do mocnych stron modelu relacyjnego można zaliczyć bardzo szybkie wyszukiwanie odpowiednich rekordów dzięki zastosowaniu indeksów, możliwość budowania bardziej złożonych struktur danych, oraz prostą rozbudowę polegającą jedynie na dołączeniu relacji do już istniejących struktur, gdyby zaszła taka potrzeba w przyszłości. Natomiast ogromną zaletą nierelacyjnej bazy danych jest możliwość przechowywania dużej ilości nieustrukturyzowanych danych, a właśnie takie są dostarczane przez analizatory. Biorąc pod uwagę te czynniki, stworzono model, wykorzystujący strukturę modelu relacyjnego, jednak bez przechowywania serii danych z analizatorów a jedynie identyfikator odwołujący się do kolekcji w MongoDB. Dzięki takiemu podejściu dane z bazy MongoDB będą pobierane w najbardziej optymalny sposób czyli przez wbudowany identyfikator, natomiast relacyjna baza danych nie będzie przechowywać ogromnych ilości danych, przenosząc przy tym większość problemów ze skalowalnością, na nierelacyjną bazę danych. Takie podejście przedstawiono na Rysunek 29. Schemat modelu hybrydowego bazy danych

Obraz zawierający tekst, diagram, Plan, Równolegle

Opis wygenerowany automatycznie

Rysunek . Schemat modelu hybrydowego bazy danych

## 5.3. Porównanie wyników

Eksperyment badający czasy zapisu i odczytu z poszczególnych modeli baz danych, opierał się jedynie o serie danych z analizatorów. Dane te są jedynymi, które mogą sprawić realny problem i wydłużyć czas odpowiedzi w funkcjach tworzonej aplikacji. Pozostałych danych takich jak szablony, lub raporty z założenia nie będzie zbyt wielu, będą to raczej jedności lub dziesiątki rekordów danych, podczas gdy ilości rekordów serii danych z analizatorów w bazie można spodziewać się w dziesiątkach, lub setkach tysięcy. Ponadto to właśnie wykorzystanie danych z analizatorów służą do spełnienia głównych funkcjonalności aplikacji jakimi są import plików oraz odczyt danych celem stworzenia przebiegów oraz przeprowadzenia niezbędnych obliczeń. Do analizy wykorzystano pięć scenariuszy akcji, które mogą być wykorzystywane najczęściej:

* Zapis pliku zawierającego znaczną ilość danych, w eksperymencie przyjęto 30 tysięcy rekordów.
* Odczyt fragmentu danych dla bazy zawierającej niewielką ilość danych, przyjęto 30 tysięcy rekordów.
* Odczyt fragmentu danych dla bazy zawierającej znaczną ilość danych, przyjęto 300 tysięcy rekordów.
* Zapis pliku zawierającego znaczniki czasu na całym przedziale istniejących danych, celem sprawdzenia wpływu indeksowania, sortowania zapisywanych dania
* Odczyt danych dla których istnieje więcej niż jeden rekord dla danego znacznika czasu, celem sprawdzenia wpływu „łączenia” rekordów dla identycznych znaczników czasu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Model relacyjny | Model nierelacyjny | Model nierelacyjny (Time Series) | Model hybrydowy |
| Czas zapisu 30 tysięcy rekordów [s] | 30,7147 | 1,4665 | 3,5510 | 2,1520 |
| Czas odczytu z bazy składającej się z 30 tysięcy rekordów [s] | 0,1245 | 0,1338 | 0,1297 | 0,1283 |
| Czas odczytu z bazy składającej się z 300 tysięcy rekordów [s] | 0,1317 | 0,1859 | 0,1455 | 0,1409 |
| Czas zapisu danych z różnymi znacznikami czasu [s] | 8,5169 | 0,9495 | 1,9578 | 1,1232 |
| Czas odczytu danych łączonych [s] | 0,1616 | 0,5247 | 0,4489 | 0,4283 |

Tabela 1. Wyniki porównania czasów zapisu i odczytu dla różnych modeli baz danych.

Czasy dla poszczególnych scenariuszy zostały zmierzone, za pomocą wbudowanych metod języka C#. Badany był tylko czas samego odczytu lub zapisu danych, bez dodatkowych konwersji wynikających z różnic pomiędzy stosowanymi modelami. Dla każdego scenariusza czas był badany wielokrotnie, następnie zostały wyciągnięte średnie z przeprowadzonych pomiarów.

Na podstawie wyników widocznych w Tabela 1. Wyniki porównania czasów zapisu i odczytu dla różnych modeli baz danych.nie jest możliwe jednoznaczne wskazanie najoptymalniejszego modelu. Każdy z modeli posiada widoczne mocne oraz słabe strony. Model relacyjny jest najszybszy w przypadku odczytu danych, natomiast ze względu na potrzebę indeksowania danych posiada znacząco bo kilkukrotnie dłuższy czas zapisu, dyskwalifikujący ten model z praktycznego zastosowania w badanej aplikacji. Model nierelacyjny wykorzystujący MongoDB natomiast prezentuje sytuację odwrotną, zapis danych jest najszybszy, jednak odczyt zajmuje najwięcej czasu, ponadto ten model posiada największy wzrost czasu odczytu dla powiększającej się bazy danych. Zastosowanie kolekcji typu *Time Series*, prezentuje natomiast wyniki bardziej optymalne, nieznacznie większy czas zapisu niż przy zastosowaniu klasycznej kolekcji MongoDB, natomiast czas odczytu zbliżony do modelu relacyjnego. Rozwiązaniem jeszcze lepszym okazuje się zastosowanie modelu hybrydowego, który również prezentuje, bardziej wyważone wyniki, natomiast są one w każdej kategorii lepsze w porównaniu z kolekcją MongoDB typu *Time Series*.

Biorąc pod uwagę przedstawione powyżej wyniki, najlepszym modelem dla tego typu aplikacji wydaje się być model hybrydowy. Jest to podejście zrównoważone z dobrymi wynikami zarówno w zapisie jak i odczycie danych. Nieznaczne gorsze wyniki uzyskał model nierelacyjny z wykorzystaniem kolekcji typu Time Series. Jeśli chodzi o łatwość implementacji poszczególnych rozwiązań z perspektywy programistycznej, zastosowanie nierelacyjnej bazy MongoDB jest znacznie prostsze, ponieważ ten typ bazy danych nie wymaga definiowania schematu dla kolekcji. Podczas zapisu nie jest wymagane tworzenie pełnych modeli, możliwe jest umieszczenie danych przykładowo w postaci słownikowej. Takie podejście nie wymaga również stosowania osobnego narzędzia typu ORM. Jednak stosowanie wyłącznie modelu nierelacyjnego może stać się problematyczne w przypadku przyszłego rozszerzania funkcjonalności aplikacji w której często relacje między danymi odgrywają ważną role. Dlatego w finalnej wersji aplikacji zastosowano model hybrydowy bazy danych, jest to podejście najbardziej optymalne jeśli chodzi o czas zapisu oraz odczytu. Posiada również największe możliwości do rozwoju w przyszłości, ponieważ w prosty sposób można dodawać kolejne dane powiązane z istniejącymi encjami poprzez rozszerzanie relacji w bazie SQL, jak również dodawać kolejne parametry z analizatora do bazy MongoDB.

# 6. Podsumowanie

Rozważana praca przedstawia proces tworzenia aplikacji webowej. Rozpoczynając od fundamentalnego etapu określania wymagań, przez wybór oraz implementację odpowiednich technologii, wzorców, aż po finalne kroki związane z realizacją interfejsu użytkownika. W pracy zastosowano wiodące technologię, które wciąż zyskują na popularności, dzięki czemu wykorzystane w pracy metody i techniki nieprędko staną się przestarzałe. Technologię te zostały krótko opisane, z naciskiem na ich praktyczne zastosowanie w kontekście tworzonej aplikacji. Dzięki temu czytelnik może nie tylko zrozumieć teoretyczne podstawy stojące za poszczególnymi narzędziami, ale także dostrzec ich rzeczywiste możliwości w bardziej praktycznym zastosowaniu.

Jednym z kluczowych elementów pracy było stworzenie, zbadanie oraz porównanie różnych modeli baz danych. Analiza ta okazała się zgodna z założeniami teoretycznymi wynikającymi ze sposobów działania poszczególnych rodzajów baz danych, oraz pozwoliła na wybranie najbardziej optymalnego rozwiązania dla tworzonej aplikacji. Wyniki tego porównania pozwoliły uwypuklić wady oraz zalety poszczególnych rozwiązań bazodanowych. Opisane zostały również takie aspekty jak ich skalowalność, oraz proces integrowania poszczególnych baz danych z perspektywy programisty, co dostarcza czytelnikowi jasnego obrazu, który z modeli najlepiej spełnia wymagania współczesnych aplikacji webowych.

Stworzona na potrzeby pracy aplikacja jest produktem w fazie MVP (ang. Minimum viable product), czyli jest produktem który w minimalnym stopniu jest gotowy do publikacji. Aplikacja zapewnia podstawowe funkcjonalności opisane w trakcie pracy, takie jak import danych z pliku, automatyczne rozpoznawanie dostarczonych nagłówków, generowanie raportów sprawdzających kryteria jakościowe napięcia, oraz pozwala na tworzenie przebiegów. Zostało zaimplementowane również wiele dodatkowych ułatwień dla użytkownika, jednak nie są to wszystkie w możliwości. W dalszym etapie rozwoju aplikacji, do zaimplementowania zostały takie aspekty jak: rozszerzenie funkcji automatycznego rozpoznawania nagłówków o kolejne analizatory, rozszerzenie możliwości tworzonych przez użytkownika szablonów przykładowo o dodatkowe kryteria wynikające z indywidualnych potrzeb użytkownika, oraz wprowadzanie kolejnych uproszczeń na interfejsie użytkownika.

W pracy pokazano, że proces tworzenia aplikacji webowej to nie tylko zestaw technicznych czynności, ale przede wszystkim ciągłe podejmowanie decyzji, które kształtują finalny produkt.

# Bibliografia

1. „Rozporządzenie Ministra Gospodarki z dnia 4 maja 2007 r. w sprawie szczegółowych warunków funkcjonowania systemu elektroenergetycznego,” (Dz.U. 2007 nr 93 poz. 623).
2. Freeman Adam, Pro ASP.NET Core MVC: sixth edition, Apress Berkeley, ISBN-13 : 978-1-4842-0389-9
3. Troelsen Andrew, Japikse Philip, Pro C# 7 eighth edition, Apress Berkeley, ISBN-13 : 978-1-4842-3017-6
4. Lock Andrew, ASP .NET Core in action, Manning Publications, ISBN 9781617294617
5. Microsoft, „Overview of ASP.NET Core” [Online]. Dostęp: <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core>
6. Microsoft, „Architectural principles” [Online]. Dostęp: <https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles>
7. Microsoft, „Overview of ASP.NET Core MVC” [Online]. Dostęp: <https://learn.microsoft.com/en-us/aspnet/core/mvc>
8. Microsoft, „Introduction to Razor Pages in ASP.NET Core” [Online]. Dostęp: <https://learn.microsoft.com/en-us/aspnet/core/razor-pages>
9. MongoDb, „Welcome to the MongoDB Documentation” [Online], Dostęp: <https://www.mongodb.com/docs>
10. Microsoft, „What's new in SQL Server 2022 ” [Online], Dostęp: <https://learn.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-2022>
11. Creative Tim, „Argon Dashboard Bootstrap” [Online], Dostęp: <https://www.creative-tim.com/learning-lab/bootstrap/overview/argon-dashboard>
12. Microsoft, „Create resource files for .NET apps” [Online], Dostęp: <https://learn.microsoft.com/en-us/dotnet/core/extensions/create-resource-files>
13. Microsoft, „Introduction to LINQ Queries (C#)” [Online], Dostęp: <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries>
14. Firlit Andrzej, Błajszczak Grzegorz, Narzędzia do oceny i analizy jakości energii elektrycznej, Miesięcznik **Energetyka**, 2020, nr 9 (795)
15. Martin Robert, Czysty Kod. Podręcznik dobrego programisty, tłum. Paweł Gonera, Gliwice: Helion, ISBN: 978-83-8322-344-5
16. Martin Robert, Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów,tłum.Wojciech Moch, Gliwice: Helion, ISBN: 978-83-283-9109-3
17. Fragment szkolenia dotyczącego wzorców architektonicznych prowadzonego na stronie Vavatech. [Online] <https://vavatech.pl/technologie/architektura/mvc> (dostęp: 24.08.2023r.).
18. MongoDB, „ Time Series Collections” [Online]. Dostęp: <https://www.mongodb.com/docs/v5.3/core/timeseries-collections/>

­­­