Implementacja aplikacji internetowej do organizacji wizyt duszpasterskich w parafiach

(Implementation of web app for organising pastoral visits in parishes)

Michał Chawar

Praca inżynierska

Promotor: dr Wiktor Zychla

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

28 stycznia 2026

Streszczenie

Celem niniejszej pracy inżynierskiej było zaprojektowanie i zaimplementowanie aplikacji internetowej wspierającej organizację corocznych wizyt duszpasterskich w parafiach rzymskokatolickich. Praca skupia się na rozwiązaniu problemów logistycznych związanych z modelem „kolędy na zaproszenie", który w ostatnich latach zyskuje na popularności, a którego tradycyjna organizacja bywa czasochłonna i skomplikowana.

Aplikacja została zrealizowana w oparciu o framework ASP.NET Core MVC [1] na platformie .NET 8 [2]. W warstwie danych wykorzystano serwer Microsoft SQL Server 2022 [3]. Interfejs użytkownika zaprojektowano przy użyciu silnika szablonów Razor [4] oraz biblioteki TailwindCSS [5], a dla zapewnienia interaktywności wybranych modułów (np. formularzy) wykorzystano framework Vue.js [6]. Całość rozwiązania została przygotowana do łatwego wdrażania z użyciem konteneryzacji (Docker [7]).

Opracowany system umożliwia parafianom elektroniczne zgłaszanie chęci przyjęcia wizyty duszpasterskiej, eliminując konieczność osobistej wizyty w kancelarii. Z perspektywy duszpasterzy aplikacja oferuje narzędzia do zarządzania bazą zgłoszeń, grupowania wizyt oraz planowania harmonogramu kolędy. Realizacja projektu wykazała, że cyfryzacja tego procesu może znacząco usprawnić zarządzanie parafią i poprawić komunikację na linii duszpasterz-wierni.

Spis treści

Rozdział 1.

Wprowadzenie

1.1. Repozytorium kodu

Kod źródłowy aplikacji opisanej w niniejszej pracy dostępny jest w publicznym repozytorium GitHub pod adresem:

`https://github.com/michalchawar/SOK.NET`

1.2. Problematyka

Doroczna wizyta duszpasterska (tzw. kolęda) jest tradycyjnym elementem życia religijnego w Polsce. Polega ona na odwiedzinach księdza w domach parafian, podczas których udziela on błogosławieństwa, rozmawia z mieszkańcami oraz zbiera ofiary na potrzeby parafii. Organizacja wizyt duszpasterskich może być jednak wyzwaniem logistycznym, zwłaszcza w większych parafiach. Często wymaga to koordynacji wielu osób, ustalania terminów oraz zarządzania informacjami o parafianach.

W dzisiejszych czasach w zależności od parafii wizyty duszpasterskie przeprowadzane są w dwóch modelach:

- **Kolęda tradycyjna**, podczas której duszpasterze starają się dotrzeć do wszystkich parafian.
- **Kolęda na zaproszenie**, podczas której duszpasterze odwiedzają tylko tych parafian, którzy wcześniej samodzielnie zgłosili chęć przyjęcia wizyty.

W niniejszej pracy skupiono się na opracowaniu aplikacji internetowej, wspierającej organizację wizyt duszpasterskich, przeprowadzanych w modelu *kolędy na zaproszenie*. Gdy dalej mowa o kolędzie, chodzi właśnie o ten model wizyty duszpasterskiej.

1.2.1. Kolęda tradycyjna

Model tradycyjny stosowany jest w większości parafii w Polsce. Funkcjonuje od dziesiątek lat i jest dobrze znany zarówno duszpasterzom, jak i parafianom. W tym modelu księża starają się odwiedzić wszystkich parafian w określonym czasie, zwyczajowo w okresie Bożego Narodzenia. Czynią to według ustalonego harmonogramu, zazwyczaj prostego i niewymagającego skomplikowanej logistyki.

Podczas gdy to podejście ma wiele zalet duszpasterskich, to jednak jest mocno czasochłonne i obciążające dla duszpasterzy oraz wiernych. Cierpi na nim często też jakoś wizyt, zarówno z powodu bardziej ścisłego ograniczenia czasowego na wizytę, jak i znacznego odsetka parafian, którzy wizytę przyjmują z obowiązku lub przyzwyczajenia, a nie z potrzeby duchowej. W większych parafiach taka forma dodatkowo angażuje ministrantów do dodatkowej pracy organizacyjnej — zapowiadania kolędy, czyli uprzedniego chodzenia od drzwi do drzwi i informowania o terminie wizyty oraz zbierania wstępnych deklaracji chęci przyjęcia wizyty. Jest to również główny sposób wczesnego prognozowania liczby wizyt danego dnia.

1.2.2. Kolęda na zaproszenie

Model ten zyskuje na popularności w ostatnich latach, zwłaszcza w większych miastach, gdzie parafianie mogą mieć bardziej zróżnicowane potrzeby, a odsetek odwiedzanych domów może być mniejszy. Znany jest również poza granicami Polski. W tej formie parafianie sami zgłaszają chęć przyjęcia wizyty duszpasterskiej poprzez odpowiedni formularz (papierowy lub elektroniczny).

Z perspektywy organizacyjnej, model ten jest znacznie bardziej efektywny. Pozwala dokładnie planować wizyty na każdy dzień, co zmniejsza obciążenie duszpasterzy, zwiększając jednocześnie kontrolę nad rozłożeniem wizyt w czasie oraz ich równomierność. Likwiduje dodatkowo potrzebę uprzedniego zapowiadania kolędy przez ministrantów, tworząc jednak nową odpowiedzialność za szczegółowe zaplanowanie porządku na dany dzień. Tym zajmuje się albo sam duszpasterz, albo wyznaczona do tego osoba (np. kościelny lub inny pracownik parafii) jako koordynator wizyty duszpasterskiej.

Kolęda na zaproszenie generuje jednak całkiem nową potrzebę — skutecznego zarządzania zgłoszeniami parafian. W większych parafiach liczba zgłoszeń może być znaczna, co wymaga odpowiednich narzędzi do ich rejestracji, przetwarzania i planowania wizyt. Wymaga to również odpowiedniej komunikacji z parafianami, aby zapewnić im jasne informacje o terminach wizyt oraz ewentualnych zmianach w harmonogramie. Dokładnie tym wymaganiom ma na celu sprostać opisywana w niniejszej pracy aplikacja internetowa.

1.3. Plan pracy

Praca składa się z siedmiu rozdziałów. W rozdziale drugim przedstawiono ogólny opis zagadnienia, jego analizę oraz wymagania stawiane aplikacji. Rozdział trzeci zawiera szczegółowy opis architektury systemu oraz zastosowanych technologii. Opisuje także modele danych i wybrane rozwiązania implementacyjne. W rozdziale czwartym omówiono inne rozwiązania, używane przez parafie do organizacji kolędy na zaproszenie, wraz z ich zaletami i wadami. Rozdział piąty zawiera instrukcję użytkownika aplikacji, opisujący jej funkcjonalności i interfejs. W rozdziale szóstym przedstawiono proces konfiguracji środowiska, uruchamianie aplikacji oraz przeprowadzanie testów jednostkowych, a także zaprezentowano statystyki kodu źródłowego. W ostatnim, siódmym rozdziale, podsumowano całą pracę, wskazując jej osiągnięcia oraz niedoskonałości, również w kontekście możliwych kierunków dalszego rozwoju aplikacji.

Rozdział 2.

Opis i analiza zagadnienia

Zagadnienie organizacji wizyty duszpasterskiej w modelu *kolędy na zaproszenie* jest dosyć skomplikowane i wymaga szeregu funkcjonujących i ściśle ze sobą współpracujących narzędzi (przede wszystkim do zbierania zgłoszeń, układania planów i informowania o nich). Dodatkowo potrzeba również personelu, który zajmie się zarówno wstępnym zaplanowaniem kolędy i ustaleniem harmonogramu, jak i późniejszą koordynacją przeprowadzania wizyty według planu.

Brak któregokolwiek z tych elementów znacznie utrudnia, a często nawet uniemożliwia, organizację kolędy na zaproszenie w sposób efektywny i zadowalający. Z kolei niewłaściwy dobór tych narzędzi lub nieścisła współpraca między nimi może prowadzić do licznych problemów organizacyjnych, błędów w planowaniu i komunikacji z parafianami.

Aby model kolędy na zaproszenie osiągał rzeczywistą przewagę nad modelem tradycyjnym, wszystkie te elementy muszą być starannie dobrane i nadzorowane. W wielu parafiach brakuje jednak odpowiednich narzędzi i zasobów. Nie ma też na rynku dedykowanych rozwiązań informatycznych, które kompleksowo wspierałyby ten model wizyty duszpasterskiej. Z tego powodu często korzysta się z rozwiązań prymitywnych, prowizorycznych lub niedostosowanych do specyficznych potrzeb kolędy na zaproszenie, co z kolei nie tylko prowadzi do licznych trudności organizacyjnych, ale dodatkowo powoduje frustrację zarówno wśród parafian, jak i koordynatorów kolędy.

Opisywana aplikacja wychodzi naprzeciw tym wyzwaniom, oferując kompleksowe narzędzie do zarządzania kolędą, które integruje wszystkie niezbędne funkcje w jednym miejscu. Dzięki temu parafie mogą skutecznie organizować wizyty duszpasterskie, minimalizując ryzyko błędów i usprawniając komunikację z parafianami.

2.1. Wymagania ogólne

Jak wspomniano wcześniej, potrzebne narzędzia do organizacji kolędy muszą być ze sobą ściśle zintegrowane i współpracujące. Brak takich na rynku, powiązany z koniecznością przeprowadzenia kolędy, był główną motywacją do stworzenia i rozwoju opisywanej aplikacji. Już w początkowym etapie projektowania systemu wyklarowały się najważniejsze wymagania, które musiał spełniać, aby nie tylko skutecznie pomagać w organizacji kolędy, ale również mieć realną szansę na szerokie zastosowanie w różnych parafiach.

2.1.1. Jedność

Aplikacja musi integrować wszystkie kluczowe funkcje potrzebne do organizacji kolędy na zaproszenie w jednym miejscu. Oznacza to, że minimalnie powinna implementować funkcjonalności:

- zbierania zgłoszeń od parafian,
- zarządzania zebranymi zgłoszeniami oraz wprowadzania nowych,
- tworzenia i edycji harmonogramu.

Niemniej jednak biorąc pod uwagę współczesne możliwości technologiczne oraz dążenie do maksymalnej automatyzacji i uproszczenia procesu organizacji możemy na podstawie implementacji tych trzech punktów zbudować nowoczesny system, który dodatkowo będzie oferował wiele innych przydatnych funkcji:

- zautomatyzowane i zindywidualizowane powiadamianie mailowe,
- portal dla parafian, umożliwiający wgląd w dane swojego zgłoszenia, jego status, termin wizyty, a nawet przewidywane godziny,
- przeprowadzanie wizyty według planu,
- zarządzanie personelem zaangażowanym w kolędę,
- raportowanie i statystyki dotyczące przebiegu kolędy.

Powyższy zestaw funkcji zebrany w jednym miejscu pozwala na kompleksowe zarządzanie kolędą oraz zapewnia znaczną przewagę nad rozbitymi, prowizorycznymi narzędziami, które często są wykorzystywane w parafiach.

2.1.2. Intuicyjność

Aplikacja powinna być prosta i intuicyjna w obsłudze, tak aby nawet osoby nieposiadające zaawansowanych umiejętności technicznych mogły z niej skutecznie korzystać. Interfejs użytkownika musi być przejrzysty, zrozumiały i jednolity.

Należy jednak dołożyć wszelkich starań, aby nie upraszczać zbytnio funkcjonalności aplikacji kosztem jej użyteczności. Wdrożenie zbyt ograniczonego zestawu funkcji może sprawić, że aplikacja nie będzie w stanie spełnić wszystkie potrzeby parafii, co z kolei może prowadzić do jej odrzucenia na rzecz bardziej rozbudowanych, choć mniej zintegrowanych rozwiązań.

2.1.3. Poprawność

Aplikacja musi działać niezawodnie i bezbłędnie, zapewniając poprawne funkcjonowanie wszystkich swoich funkcji. Wszelkie błędy lub awarie mogą prowadzić do poważnych problemów organizacyjnych, a nawet do utraty zaufania parafian. Dlatego tak ważne jest, aby aplikacja była starannie przetestowana i regularnie aktualizowana, aby zapewnić jej stabilność i niezawodność.

Oprócz samej poprawności implementacji aplikacji należy wziąć pod uwagę, że dane wprowadzone do systemu muszą być również poprawne i spójne. W tym celu aplikacja powinna implementować mechanizmy walidacji danych, które zapewnią, że wprowadzone informacje są zgodne z określonymi standardami i nie zawierają błędów.

Ostatecznie aplikacja będzie wykorzystywana przez koordynatorów kolędy, którzy mogą nie mieć zaawansowanych umiejętności technicznych. Z tego powodu należy przyłożyć szczególną uwagę do upewnienia się, że operacje dostępne z poziomu interfejsu użytkownika są dobrze przemyślane i nie prowadzą do niezamierzonych konsekwencji, a także dobrze opisane, aby użytkownicy mogli z nich korzystać bez ryzyka popełnienia błędów.

2.1.4. Reużywalność

Aplikacja powinna być zaprojektowana w sposób umożliwiający jej łatwe dostosowanie i ponowne wykorzystanie w różnych parafiach. Oznacza to, że powinna być elastyczna i konfigurowalna, a także umożliwiać coroczne przeprowadzanie wizyt kolędowych bez konieczności manualnej rekonfiguracji lub ponownego wdrażania. Dzięki temu parafie będą mogły korzystać z aplikacji przez wiele lat, co znacznie zwiększy jej wartość i użyteczność.

Sam fakt wykorzystania w różnych parafiach wymusza również konieczność uwzględnienia tej możliwości w systemie, aby różne parafie mogły korzystać z aplikacji jednocześnie, bez konieczności tworzenia odrębnych instancji lub wersji aplikacji

dla każdej z nich.

2.2. Wymagania funkcjonalne

Podsumowując powyższe rozważania, aplikacja powinna spełniać następujące wymagania funkcjonalne:

- umożliwiać parafianom zgłaszanie chęci uczestnictwa w kolędzie na zaproszenie poprzez formularz online,
- pozwalać koordynatorom kolędy na przeglądanie, edytowanie i zarządzanie zebranymi zgłoszeniami,
- umożliwiać tworzenie i edycję harmonogramu wizyt duszpasterskich na podstawie zebranych zgłoszeń,
- automatycznie wysyłać powiadomienia mailowe do parafian o potwierdzeniu przyjęcia zgłoszenia,
- umożliwiać manualne wysyłanie powiadomień mailowych z informacjami o zmianie danych zgłoszenia, zaplanowaniu zgłoszenia, itp.,
- oferować portal dla parafian, gdzie mogą oni sprawdzać status swojego zgłoszenia, termin wizyty oraz przewidywane godziny,
- wspierać koordynację personelu zaangażowanego w kolędę, umożliwiając przypisywanie zadań i monitorowanie postępów,
- generować raporty i statystyki dotyczące przebiegu kolędy, takie jak liczba zgłoszeń, liczba przeprowadzonych wizyt itp.,
- umożliwiać konfigurację i dostosowanie aplikacji do specyficznych potrzeb różnych parafii,
- zapewniać możliwość corocznego przeprowadzania kolędy bez konieczności manualnej rekonfiguracji lub ponownego wdrażania aplikacji,
- umożliwiać jednoczesne korzystanie z aplikacji przez różne parafie, bez konieczności tworzenia odrębnych instancji lub wersji aplikacji dla każdej z nich.

2.3. Wymagania niefunkcjonalne

Oprócz wymagań funkcjonalnych, aplikacja powinna również spełniać następujące wymagania niefunkcjonalne:

- być dostępna online, aby parafie i parafianie mogli z niej korzystać z dowolnego miejsca i o dowolnym czasie,
- być skalowalna, aby mogła obsługiwać rosnącą liczbę użytkowników i zgłoszeń bez utraty wydajności,
- być bezpieczna, aby chronić dane osobowe parafian i zapewnić poufność informacji,
- być zgodna z obowiązującymi przepisami dotyczącymi ochrony danych osobowych, takimi jak RODO,
- być łatwa w utrzymaniu i aktualizacji, aby zapewnić jej długotrwałe funkcjonowanie i możliwość dostosowywania do zmieniających się potrzeb parafii.

Rozdział 3.

Implementacja

3.1. Technologie i narzędzia

Aplikacja została zaimplementowana w technologii ASP.NET Core MVC [1] w środowisku .NET 8 [2]. Do zarządzania bazą danych wykorzystano Microsoft SQL Server 2022 [3]. Interfejs użytkownika został zbudowany przy użyciu TailwindCSS [5], zapewniającego nowoczesny i responsywny wygląd, oraz Vue.js [6] do obsługi interaktywnych komponentów i wieloetapowych procedur. Całość projektu została objęta konteneryzacją przy użyciu narzędzia Docker [7], co umożliwia łatwe wdrożenie i utrzymanie aplikacji.

3.1.1. Strona kliencka

Część frontendowa aplikacji wykonana jest w dwóch metodykach. Pierwsza z nich przeznaczona jest do tworzenia statycznych stron HTML, które są renderowane po stronie serwera w sposób typowy dla wzorca MVC. Drugie podejście łączy wstępną generację HTML z dynamicznym uzupełnianiem treści po stronie klienta przy użyciu biblioteki Vue.js. Takie podejście zostało zastosowane w miejscach, gdzie wymagana jest większa interaktywność, na przykład w formularzach wieloetapowych czy dynamicznych listach.

Razor

Do tworzenia statycznych stron HTML wykorzystano silnik szablonów Razor [4], który jest integralną częścią frameworka ASP.NET Core MVC. Razor umożliwia łączenie kodu C# z HTML w sposób czytelny i efektywny, co pozwala na dynamiczne generowanie treści stron na serwerze przed ich wysłaniem do przeglądarki klienta.

Z perspektywy klienta, strony wygenerowane za pomocą Razor są tradycyjnymi stronami HTML, które mogą zawierać osadzone skrypty JavaScript i style

CSS. Dzięki temu możliwe jest tworzenie responsywnych i interaktywnych interfejsów użytkownika, nawet jeśli główna logika renderowania stron odbywa się po stronie serwera.

Vue.js

Niektóre części aplikacji wymagają większej interaktywności i dynamicznego zarządzania stanem interfejsu użytkownika (w celu zachowania wymogu intuicyjności). Do ich implementacji wykorzystano bibliotekę Vue.js [6].

Strony te są początkowo generowane na serwerze przy użyciu Razor, a następnie po załadowaniu w przeglądarce klienta, Vue.js przejmuje kontrolę nad interaktywnymi elementami interfejsu użytkownika. Dzięki temu możliwe jest dynamiczne aktualizowanie treści, obsługa zdarzeń użytkownika oraz zarządzanie stanem aplikacji bez konieczności ponownego ładowania całej strony.

Aplikacje Vue.js są implementowane bezpośrednio w tagach *script* w niektórych plikach widoków. Ładowany jest wówczas globalny plik biblioteki, który udostępnia wszelkie funkcjonalności jako właściwości globalnego obiektu Vue. Następnie za jego pomocą tworzone są instancje aplikacji Vue, które są przypisywane do określonych elementów DOM na stronie. Ten proces następuje w przeglądarce użytkownika, co pozwala na płynne przejście od statycznego renderowania do dynamicznej interaktywności.

W opisywanym projekcie aplikacje Vue.js używane są na dwa sposoby:

- do zwiększania interaktywności prostych bądź zaawansowanych elementów interfejsu po stronie klienta bez potrzeby komunikacji z serwerem,
- do zarządzania bardziej złożonymi komponentami interfejsu, które wymagają komunikacji z serwerem w celu pobierania lub wysyłania danych.

W pierwszym przypadku często zachowywana jest logika renderowania po stronie serwera, włącznie z tworzeniem formularzy za pomocą pomocników HTML ASP.NET Core przy użyciu modeli widoków. Vue.js jest wtedy wykorzystywany do obsługi interakcji użytkownika, takich jak walidacja danych wprowadzanych w formularzach, dynamiczne dodawanie lub usuwanie elementów listy, czy aktualizacja widoku na podstawie działań użytkownika bez konieczności ponownego ładowania strony. Czasem jednak dane osadzane są bezpośrednio w zmiennej JavaScript w widoku (po przekonwertowaniu do JSON), co pozwala na pełną kontrolę nad renderowaniem interfejsu po stronie klienta przez samą aplikację Vue.

W drugim przypadku Vue.js zarządza bardziej złożonymi komponentami, które wymagają komunikacji z serwerem. Wówczas aplikacja przypomina bardziej wzorzec SPA (Single Page Application), gdzie Vue.js odpowiada za renderowanie interfejsu

użytkownika, a komunikacja z serwerem odbywa się za pomocą asynchronicznych żądań HTTP (przy użyciu Fetch API [8]). Dane są pobierane z serwera w formacie JSON, a następnie wykorzystywane do aktualizacji widoku w czasie rzeczywistym. Podobnie, dane wprowadzone przez użytkownika są wysyłane z powrotem na serwer w formacie JSON, gdzie są przetwarzane i zapisywane w bazie danych. Nie jest to jednak pełna aplikacja SPA, ponieważ nawigacja między różnymi stronami nadal odbywa się poprzez tradycyjne przeładowanie strony.

JQuery i JavaScript

Do obsługi prostych interakcji na stronach wykorzystano również bibliotekę jQuery [9] oraz czysty JavaScript. W miejscach, gdzie nie jest wymagana pełna funkcjonalność Vue.js, jQuery pozwala na szybkie i efektywne manipulowanie elementami DOM oraz obsługę zdarzeń. Dodatkowo odpowiada za walidację formularzy po stronie klienta, co poprawia doświadczenie użytkownika poprzez natychmiastowe informowanie o błędach przed wysłaniem danych na serwer.

JavaScript jest również używany do implementacji powszechnych funkcji dla aplikacji, znajdujących się w plikach zewnętrznych, które są dołączane do odpowiednich widoków.

TailwindCSS

Do stylizacji interfejsu użytkownika wykorzystano framework CSS o nazwie TailwindCSS [5]. Jest to narzędzie oparte na podejściu utility-first, które umożliwia szybkie tworzenie responsywnych i estetycznych interfejsów użytkownika poprzez stosowanie gotowych klas CSS bez konieczności pisania własnych stylów od podstaw.

Głównymi zaletami TailwindCSS są jego elastyczność i możliwość tworzenia responsywnych projektów. Framework oferuje szeroki zestaw klas, które pozwalają na precyzyjne kontrolowanie wyglądu elementów interfejsu, takich jak marginesy, wypełnienia, kolory, typografia i układ. Dzięki temu aplikacja została w prosty sposób dostosowana do różnych rozmiarów ekranów, zapewniając optymalne doświadczenie użytkownika na urządzeniach mobilnych, tabletach i komputerach stacjonarnych.

Nad to użyto również wtyczki DaisyUI [10], która rozszerza możliwości TailwindCSS o gotowe komponenty UI, takie jak przyciski, formularze, karty i nawigacje, tworzone za pomocą określonych klas. Dzięki temu proces tworzenia interfejsu użytkownika był szybszy i bardziej efektywny, pozwalając skupić się na funkcjonalności aplikacji zamiast na szczegółach stylizacji.

3.1.2. Strona serwerowa

Część backendowa aplikacji została zaimplementowana przy użyciu frameworka ASP.NET Core [1], opartego na platformie ṄET w wersji 8 [2]. Wykorzystano w nim różne biblioteki i narzędzia dostępne w ekosystemie ṄET, aby zapewnić wydajność, skalowalność i bezpieczeństwo aplikacji.

ASP.NET Core

Framework ASP.NET Core umożliwia tworzenie aplikacji webowych zgodnych z wzorcem Model-View-Controller (MVC) [11], co pozwala na oddzielenie logiki biznesowej od warstwy prezentacji i danych. Nadal pozwala przy tym udostępniać interfejs API w obrębie tej samej aplikacji, co zostało wykorzystane w projekcie. ASP.NET Core oferuje wbudowane mechanizmy do obsługi routingu, autoryzacji, uwierzytelniania oraz zarządzania sesjami, co ułatwia tworzenie bezpiecznych i wydajnych aplikacji webowych. Dodatkowo, framework ten jest wysoce konfigurowalny i wspiera nowoczesne praktyki programiste, takie jak wstrzykiwanie zależności i middleware.

Entity Framework Core Do komunikacji z bazą danych wykorzystano Entity Framework Core (EF Core) [12], który jest popularnym narzędziem ORM (Object-Relational Mapping) dla platformy ṄET. EF Core umożliwia programistom pracę z bazą danych za pomocą obiektów C#, eliminując potrzebę pisania bezpośrednich zapytań SQL. Dzięki temu proces tworzenia, odczytu, aktualizacji i usuwania danych (CRUD) staje się bardziej intuicyjny i zintegrowany z logiką aplikacji.

ASP.NET Core Identity Do zarządzania uwierzytelnianiem i autoryzacją użytkowników wykorzystano bibliotekę ASP.NET Core Identity [13]. Jest to kompleksowe rozwiązanie, które umożliwia tworzenie i zarządzanie kontami użytkowników, obsługę ról oraz implementację mechanizmów bezpieczeństwa, takich jak resetowanie haseł czy weryfikacja dwuetapowa. ASP.NET Core Identity integruje się bezproblemowo z frameworkiem ASP.NET Core, co pozwala na łatwe dodanie funkcji logowania i zarządzania użytkownikami do aplikacji webowej oraz przechowywanie ich danych w bazie danych za pośrednictwem Entity Framework Core.

WebOptimizer Do optymalizacji dostarczania statycznych plików JavaScript zastosowano bibliotekę WebOptimizer. Narzędzie to umożliwia minifikację, łączenie i kompresję plików statycznych, co prowadzi do zmniejszenia rozmiaru przesyłanych zasobów i przyspieszenia ładowania stron internetowych. WebOptimizer automatycznie przetwarza pliki podczas uruchamiania aplikacji, co ułatwia zarządzanie zasobami i poprawia wydajność aplikacji webowej.

MailKit Do obsługi wysyłania wiadomości e-mail z aplikacji wykorzystano bibliotekę MailKit [14]. Jest to nowoczesne i wydajne narzędzie do obsługi protokołów SMTP, POP3 i IMAP w środowisku ṄET. MailKit oferuje szeroki zakres funkcji, takich jak tworzenie i wysyłanie wiadomości e-mail, obsługa załączników, szyfrowanie oraz autoryzacja. Biblioteka ta jest znana ze swojej wydajności i niezawodności, co czyni ją idealnym wyborem do integracji funkcji e-mail w aplikacjach webowych.

DataProtection Do zapewnienia trwałości kluczy kryptograficznych wykorzystano bibliotekę ASP.NET Core Data Protection. Dzięki integracji z Entity Framework Core, klucze są przechowywane w bazie danych, co zapewnia ich persystencję między restartami aplikacji, umożliwiając zachowanie sesji użytkowników i likwidując wymóg ponownego logowania po restarcie serwera.

QuestPDF Do generowania dokumentów PDF w aplikacji wykorzystano bibliotekę QuestPDF [15]. Jest to nowoczesne narzędzie do tworzenia wysokiej jakości dokumentów PDF w środowisku ṄET. QuestPDF oferuje prosty i intuicyjny interfejs programistyczny, który umożliwia definiowanie układu i stylu dokumentów za pomocą kodu C#. Biblioteka obsługuje różnorodne funkcje, takie jak dodawanie tekstu, obrazów, tabel i wykresów, co pozwala na tworzenie profesjonalnie wyglądających raportów i dokumentów bez konieczności korzystania z zewnętrznych narzędzi do edycji PDF.

Narzędzie QuestPDF zostało wykorzystane na licencji Community MIT [16], która pozwala na darmowe użycie biblioteki w projektach niekomercyjnych i komercyjnych, generujących dochód poniżej $1,000,000 rocznie.

Microsoft SQL Server 2022

Aplikacja korzysta z relacyjnej bazy danych Microsoft SQL Server 2022 [3] do przechowywania wszystkich danych niezbędnych do jej funkcjonowania. Komunikacja pomiędzy aplikacją a bazą danych odbywa się za pośrednictwem opisanego wyżej Entity Framework Core, który umożliwia mapowanie obiektów C# na tabele i rekordy w bazie danych. Dodatkowo z bazą danych komunikują się narzędzia ASP.NET Core Identity oraz Data Protection (poprzez integrację z EF Core).

Traefik

W środowisku produkcyjnym do zarządzania ruchem sieciowym i obsługi certyfikatów SSL/TLS wykorzystano narzędzie Traefik [17]. Jest to nowoczesny i wydajny reverse proxy oraz load balancer, który automatycznie wykrywa usługi i konfiguruje trasowanie ruchu na podstawie reguł zdefiniowanych przez użytkownika. Traefik zintegrowany jest z Dockerem, co umożliwia dynamiczne zarządzanie ruchem

sieciowym w środowiskach kontenerowych. Dodatkowo, Traefik oferuje wbudowaną obsługę Let's Encrypt [18], co pozwala na automatyczne generowanie i odnawianie certyfikatów SSL/TLS, zapewniając bezpieczną komunikację między klientami a serwerem.

3.1.3. Ciągła integracja i dostarczanie (CI/CD)

Główna część aplikacji została objęta konteneryzacją przy użyciu narzędzia Docker [7]. Konteneryzacja pozwala na zapakowanie aplikacji wraz ze wszystkimi jej zależnościami w odizolowane środowisko, co umożliwia łatwe wdrożenie i dalsze utrzymanie. Dodatkowo za pomocą pliku Dockerfile zdefiniowano proces budowania obrazu kontenera od podstaw, co zapewnia spójność środowiska uruchomieniowego aplikacji niezależnie od miejsca jej wdrożenia.

Wraz z kontenerem aplikacji ASP.NET Core przy pomocy narzędzia orkiestracji kontenerów Docker Compose [19] można w prosty sposób uruchomić również kontener bazy danych Microsoft SQL Server 2022 oraz (w środowisku produkcyjnym) kontener Traefik, jednocześnie konfigurując ich współpracę, sieć wewnętrzną oraz wolumeny do trwałego przechowywania danych.

3.1.4. Testy

W osobnym projekcie umieszczony został zestaw testów jednostkowych aplikacji. Testy te zostały zaimplementowane przy użyciu frameworka xUnit.net [20], który jest popularnym narzędziem do tworzenia i uruchamiania testów jednostkowych w środowisku ṄET. Testy jednostkowe pozwalają na weryfikację poprawności działania poszczególnych komponentów aplikacji, co przyczynia się do zwiększenia jakości kodu oraz ułatwia wykrywanie i naprawianie błędów na wczesnym etapie rozwoju oprogramowania.

Projekt testów jednostkowych korzysta z biblioteki Moq [21] do tworzenia atrap (*mock*'ów) zależności, co umożliwia izolowanie testowanych jednostek kodu od ich rzeczywistych zależności. Dzięki temu testy są bardziej niezawodne i skupiają się wyłącznie na logice testowanego komponentu.

Same testy zostały zaprojektowane w metodologii AAA [22] (*Arrange, Act, Assert*), co pozwala na czytelne i zrozumiałe strukturyzowanie przypadków testowych.

3.1.5. Zarządzanie kodem źródłowym

Kod źródłowy aplikacji jest przechowywany w repozytorium Git na platformie GitHub. Wykorzystano funkcje zarządzania wersjami, takie jak gałęzie i pull requesty, aby umożliwić współpracę zespołową (w przyszłości) oraz śledzenie zmian w

kodzie. Dodatkowo, repozytorium zawiera dokumentację projektu, instrukcje dotyczące wdrożenia oraz konfiguracji środowiska deweloperskiego.

3.1.6. Narzędzia modelowania diagramów

Do tworzenia diagramów UML oraz innych wizualizacji na potrzeby tej pracy użyto dwóch narzędzi:

- Structurizr — do tworzenia diagramów architektury oprogramowania (C4 Model),
- Visual Paradigm (w wersji Community) — do tworzenia pozostałych diagramów (np. modelu danych).

3.2. Modele danych

Poniżej przedstawiono dwa diagramy. Pierwszy odpowiada modelowi dziedzinowemu, reprezentującemu główne pojęcia oraz ich relacje w kontekście logiki aplikacji. Drugi odnosi się do modelu obiektowego, który odwzorowuje strukturę encji EF Core, tym samym determinujących strukturę bazy danych.

3.2.1. Model pojęciowy

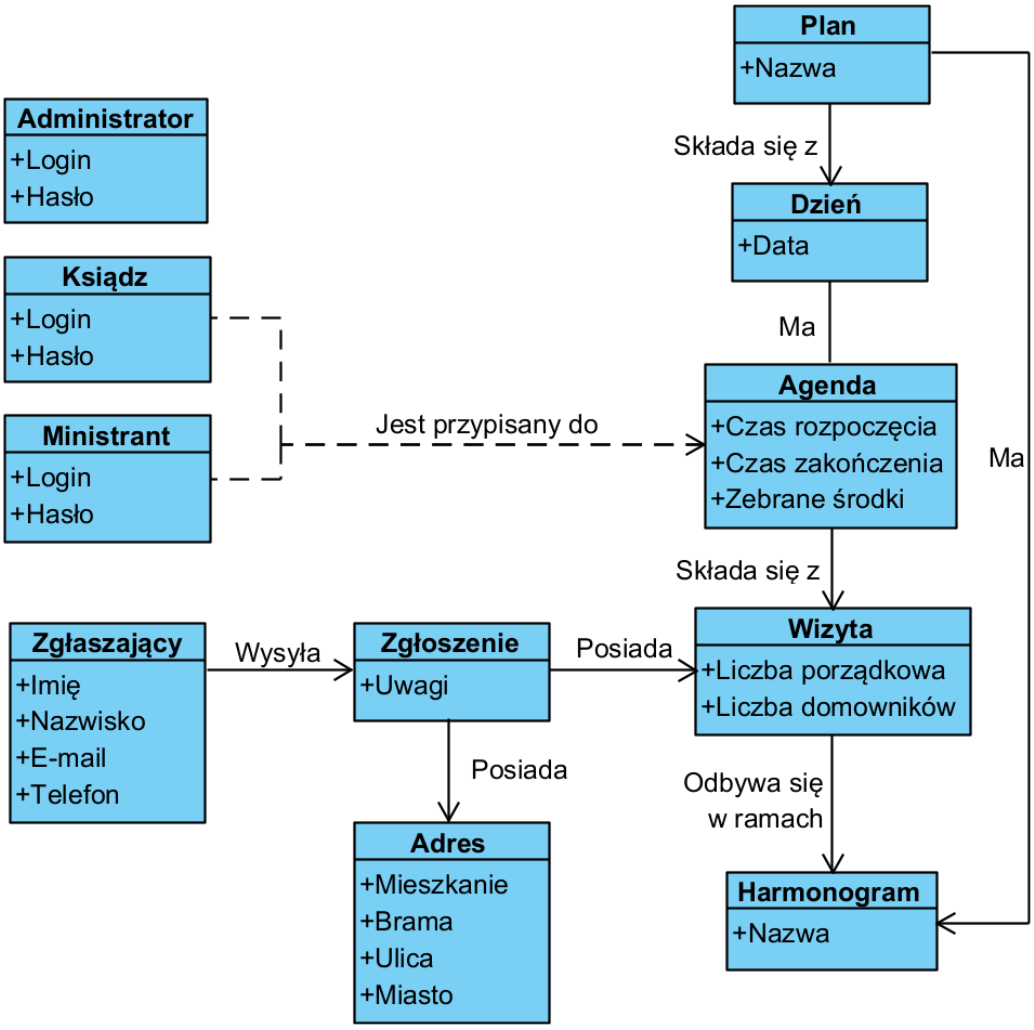
Zaprezentowany diagram modelu pojęciowego (rys. 3.1) ilustruje główne pojęcia oraz ich wzajemne relacje w kontekście logiki aplikacji. Model ten stanowi abstrakcyjną reprezentację struktur danych i zależności między nimi, niezależnie od konkretnej implementacji technicznej.

Całość modelu mieści się w logicznych granicach jednej **parafii** — wymienione pojęcia odnoszą się do zarządzania kolędą w obrębie konkretnej parafii, a tym samym do jednej domeny aplikacji.

Personel

Personel parafialny składa się z różnych ról, które mają określone uprawnienia i obowiązki w kontekście zarządzania kolędą. Główne role to:

- **Administrator** — osoba odpowiedzialna za zarządzanie aplikacją, w tym tworzenie i modyfikowanie planów kolędy oraz zarządzanie użytkownikami.
- **Ksiądz** — duchowny, który odwiedza parafian podczas kolędy. Może zarządzać danym planem i planować wizyty.



Rysunek 3.1: Model pojęciowy aplikacji

- **Ministrant** — osoba, która towarzyszy księdzu podczas wizyt. Może być przypisana do konkretnych agend w planie kolędy i na bieżąco aktualizować status wizyt.

Plan

Najbardziej ogólnym pojęciem w zakresie planowania kolędy jest **Plan**. Reprezentuje on konkretny plan kolędy dla danego roku. Planów może być wiele, są zarządzane przez parafialnego administratora aplikacji.

Harmonogram

Każdy plan kolędy musi zawierać przynajmniej jeden **Harmonogram**. Harmonogramy umożliwiają podział zgłoszeń parafian na różne grupy, które z kolei mogą służyć do filtrowania zgłoszeń, a także do tworzenia odrębnych planów wizyt

dla różnych księży. Przykładowo można za ich pomocą rozdzielić kolędę w terminie zasadniczym od kolędy dodatkowej.

Dzień

Każdy plan składa się z wielu **Dni**, które reprezentują konkretne dni w trakcie trwania kolędy.

Agenda

Każdy dzień zawiera wiele **Agend**, które grupują i porządkują wizyty danego dnia. Każda agenda jest przypisana do konkretnego dnia i zawiera informacje o ministrantach towarzyszących podczas wizyt. Agendy są odpowiednikiem list wizyt, które ksiądz i ministranci realizują w danym dniu kolędy. W najprostszym rozumieniu mogą służyć więc jako podział wizyt na księdza pierwszego, drugiego, itd.

Wizyta

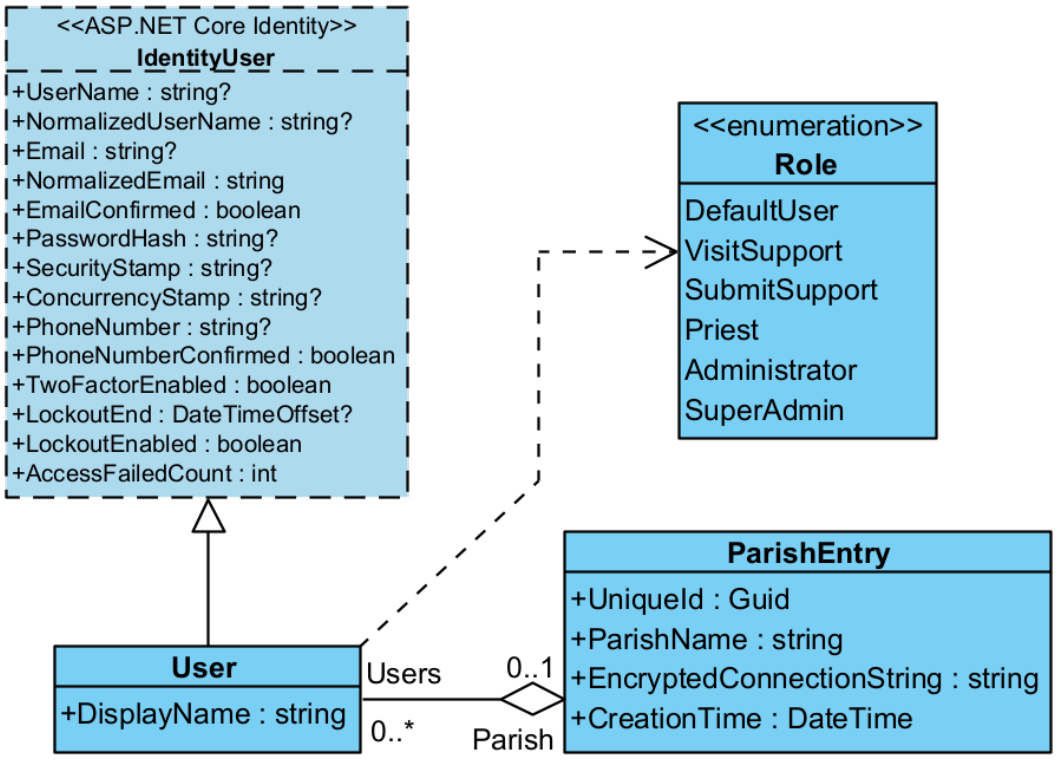
Podstawowym elementem planu kolędy jest **Wizyta**. Reprezentuje ona konkretną wizytę księdza u parafianina. Wizyta może być przypisana do konkretnej agendy, a tym samym być zaplanowana na konkretny dzień. Kolejność wizyt w agendzie jest wyznaczana przez liczbę porządkową wizyty. Zasadniczo przypisana jest też do konkretnego harmonogramu.

Zgłoszenie, zgłaszający i adres

Każda wizyta jest powiązana z konkretnym **Zgłoszeniem** parafianina. Zgłoszenie zawiera wszystkie niezbędne informacje o parafianinie, takie jak dane kontaktowe oraz adres zamieszkania (w powiązanych pojęciach). Do zgłoszenia można dołączyć także dodatkowe uwagi (np. prośby dotyczące wizyty lub preferencje dotyczące terminu).

3.2.2. Model obiektowy

Poniżej zaprezentowano dwa diagramy modelu obiektowego aplikacji. Pierwszy z nich (rys. 3.2) odnosi się do kontekstu ogólnego (centralnego), tj. do logiki zarządzania użytkownikami (docelowo w różnych domenach — parafiach). Drugi (rys. 3.3) przedstawia model obiektowy funkcjonujący w obrębie konkretnej domeny (kontekst parafialny).



Rysunek 3.2: Model obiektowy aplikacji (kontekst centralny)

Kontekst centralny

W kontekście centralnym aplikacji zajęto się zarządzaniem użytkownikami oraz ich rolami w różnych domenach (parafiach). Główne klasy w tym modelu to **User**, **Role** oraz **ParishEntry**.

User Reprezentuje użytkownika aplikacji. Zawiera podstawowe informacje o nim, w większości dziedziczone z klasy `IdentityUser` z ASP.NET Core Identity. Jest przypisany do konkretnej parafii (o ile nie jest w roli *SuperAdmin*).

Najważniejsze pola klasy `User` to:

- `UserName` — login użytkownika,
- `Email` — adres e-mail użytkownika,
- `PasswordHash` — hash hasła użytkownika,
- `DisplayName` — nazwa użytkownika.

Role Reprezentuje role użytkowników w aplikacji. Każda rola definiuje zestaw uprawnień i obowiązków, które użytkownik posiada w kontekście zarządzania kolędą.

`Role` jest enumeracją o następujących wartościach:

- **DefaultUser** — uprawnia do podstawowego dostępu do aplikacji,
- **VisitSupport** — uprawnia do przeprowadzania wizyt kolędowych (odpowiednik ministranta),
- **SubmitSupport** — uprawnia do wprowadzania zgłoszeń i zarządzania nimi,
- **Priest** — uprawnia do wszystkich powyższych czynności oraz do planowania wizyt,
- **Administrator** — uprawnia do zarządzania aplikacją i wszystkimi jej funkcjami,
- **SuperAdmin** — uprawnia do zarządzania wszystkimi parafiami, a przez to również ich wszystkimi danymi.

ParishEntry Reprezentuje rzeczywistą parafię, która używa aplikacji do zarządzania kolędą. Do każdej może być przypisanych wielu użytkowników.

Najważniejsze pola klasy `ParishEntry` to:

- `UniqueId` — unikalny identyfikator parafii,
- `ParishName` — nazwa parafii,
- `EncryptedConnectionString` — zaszyfrowany łańcuch połączenia do bazy danych parafii,
- `CreationTime` — data utworzenia parafii w systemie.

Kontekst parafialny

Kontekst parafialny zbiera wszystkie klasy związane z zarządzaniem kolędą w obrębie konkretnej parafii. Główne klasy w tym modelu to **Plan**, **Schedule**, **Day**, **Agenda**, **Submission** oraz **Visit**.

W tym modelu wyróżniają się dwie najistotniejsze grupy encji (w podziale funkcjonalnym): encje służące do *Przyjmowania zgłoszeń* i do *Planowania wizyt*. Wszystkie opisane są poniżej.

Przyjmowanie zgłoszeń

Submission Reprezentuje zgłoszenie na kolędę. Zawiera wszystkie niezbędne informacje o parafianinie, takie jak dane kontaktowe oraz adres zamieszkania (w powiązanych klasach).

Najważniejsze pola klasy `Submission` to:

- `SubmitterNotes` — dodatkowe uwagi (od zgłaszającego do administratora),
- `AdminMessage` — informacja systemowa (od administratora do zgłaszającego),
- `AdminNotes` — wewnętrzne notatki (widoczne tylko dla zalogowanych użytkowników),
- `NotesStatus` — status realizacji dodatkowych uwag (np. oczekujące, zrealizowane).

Submitter Reprezentuje parafianina, który składa zgłoszenie na kolędę. Jest bezpośrednio powiązany ze zgłoszeniem (lub wieloma).

Najważniejsze pola klasy `Submitter` to:

- `Name` — imię zgłaszającego,
- `Surname` — nazwisko zgłaszającego,
- `PhoneNumber` — numer telefonu zgłaszającego,
- `Email` — adres e-mail zgłaszającego.

Address Reprezentuje adres zamieszkania parafianina. Jest bezpośrednio powiązany ze zgłoszeniem (w relacji *1 do 1* z dokładnością do konkretnego planu kolędy). Jest on prawie w całości zatomizowany, aby ujednolicić format adresów, podawanych w formularzu zgłoszeniowym (by zachować jednoznaczność).

Jest powiązany z szeregiem klas pomocniczych, które reprezentują poszczególne elementy adresu (np. budynek (brama), ulica, miasto).

Najważniejsze pola klasy `Address` to:

- `ApartmentNumber` — numer mieszkania,
- `ApartmentLetter` — litera mieszkania (opcjonalnie, raczej rzadko),
- właściwości *cache* — kopia tekstowa właściwości z klas pomocniczych (dla przyspieszenia operacji bazodanowych),
- `FilterableString` — znormalizowany łańcuch znaków do celów filtrowania i wyszukiwania adresów (*computed* — obliczony przez bazę danych na podstawie pól *cache*).

Building Reprezentuje budynek (bramę) w adresie zamieszkania parafianina. Każdy budynek może mieć wiele adresów (mieszkań). Jest tworzony przez administratora parafii i wybierany z listy w formularzu zgłoszeniowym (i kilku innych miejscach).

Najważniejsze pola klasy `Building` to:

- `Number` — numer budynku,
- `Letter` — litera budynku (opcjonalnie),
- `FloorCount` — liczba pięter w budynku (opcjonalnie, do celów statystycznych),
- `ApartmentCount` — liczba mieszkań w budynku (opcjonalnie, do celów statystycznych),
- `HighestApartmentNumber` — najwyższy numer mieszkania w budynku (opcjonalnie, do celów statystycznych),
- `HasElevator` — czy budynek posiada windę (może mieć wpływ na planowanie wizyt),
- `AllowSelection` — czy budynek może być wybrany w formularzu zgłoszeniowym (w przeciwnym przypadku jest ukryty i może zostać wybrany tylko przez zalogowanego użytkownika).

Street Reprezentuje ulicę w adresie zamieszkania parafianina. Każda ulica może mieć wiele budynków. Podobnie jak budynek, jest tworzona przez administratora parafii i wybierany z listy w formularzu zgłoszeniowym (i kilku innych miejscach).

Najważniejsze pola klasy `Street` to:

- `Name` — nazwa ulicy (bez tytułów typu ulica, aleja, plac itd.),
- `PostalCode` — kod pocztowy ulicy (opcjonalnie).

StreetSpecifier Reprezentuje typ ulicy (np. ulica, aleja, plac itd.). Każda ulica jest powiązana z jednym `StreetSpecifier`, który określa jej typ. Podobnie jak budynek i ulica, jest tworzony przez administratora parafii.

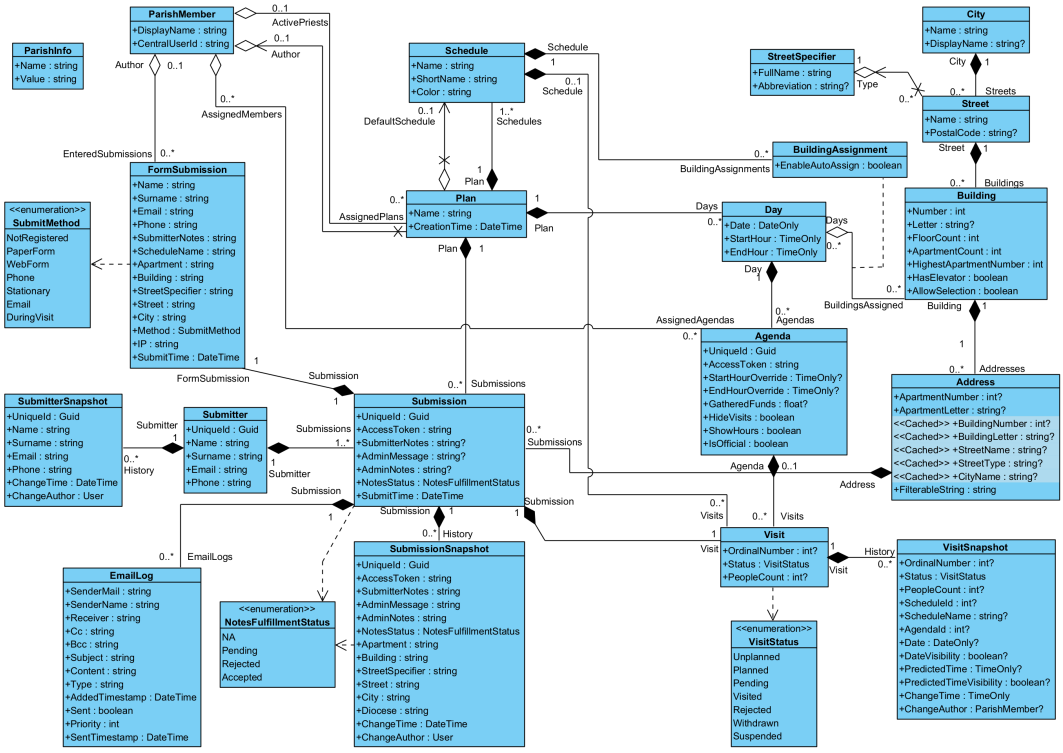
Najważniejsze pola klasy `StreetSpecifier` to:

- `FullName` — pełna nazwa typu ulicy (np. ulica, aleja, plac itd.),
- `Abbreviation` — skrócona nazwa typu ulicy (np. ul., al., pl.).

City Reprezentuje miasto w adresie zamieszkania parafianina. Każde miasto może mieć wiele ulic. Podobnie jak powyższe klasy, jest tworzone przez administratora parafii.

Najważniejsze pola klasy `City` to:

- `Name` — pełna nazwa miasta,
- `DisplayName` — wyświetlana nazwa miasta.



Rysunek 3.3: Model obiektowy aplikacji (kontekst parafialny)

Planowanie wizyt

Visit Jest odpowiednikiem zgłoszenia w kontekście planowania wizyt. Reprezentuje konkretną wizytę księdza u parafianina. Jest powiązana z danym zgłoszeniem w relacji *1 do 1*, w ten sposób są ze sobą ściśle powiązane.

Najważniejsze pola klasy `Visit` to:

- `OrdinalNumber` — liczba porządkowa wizyty w agendzie,
- `Status` — status wizyty (np. zaplanowana, zaakceptowana, odrzucona),
- `PeopleCount` — liczba domowników uczestniczących w wizycie (do celów statystycznych).

Plan Reprezentuje plan wizyt kolędowych (najczęściej na dany rok). Zawiera wiele dni, które z kolei zawierają agendy i wizyty. Jest zarządzany przez parafialnego administratora aplikacji. Można przypisać do niego księży, którzy będą w nim występować.

Najważniejsze pola klasy `Plan` to:

- `Name` — nazwa planu,
- `CreationTime` — czas utworzenia planu.

Schedule Reprezentuje harmonogram wizyt w planie kolędy. Służy do podziału zgłoszeń parafian na różne grupy, np. na kolędę w terminie i kolędę dodatkową. Każdy plan musi zawierać przynajmniej jeden harmonogram.

Najważniejsze pola klasy `Schedule` to:

- `Name` — nazwa harmonogramu,
- `ShortName` — skrócone oznaczenie harmonogramu,
- `Color` — kolor harmonogramu (zapisany w formacie szesnastkowym z poprzedzającym znakiem #).

Day Reprezentuje konkretny dzień w trakcie trwania kolędy. Zawiera wiele agend, które grupują wizyty danego dnia.

Najważniejsze pola klasy `Day` to:

- `Date` — data dnia,
- `StartHour` — godzina rozpoczęcia kolędy w danym dniu,
- `EndHour` — godzina zakończenia kolędy w danym dniu.

Agenda Reprezentuje listę wizyt w konkretnym dniu kolędy. Każda agenda jest przypisana do konkretnego dnia i można do niej przypisywać księdza oraz ministrantów. Porządek wizyt w agendzie jest wyznaczany przez liczbę porządkową wizyty.

Najważniejsze pola klasy `Agenda` to:

- `StartHourOverride` — godzina rozpoczęcia kolędy dla danej agendy (opcjonalnie, nadpisuje godzinę z dnia),
- `EndHourOverride` — godzina zakończenia kolędy dla danej agendy (opcjonalnie, nadpisuje godzinę z dnia),

- `GatheredFunds` — suma zebranych funduszy podczas wizyt w danej agendzie,
- `HideVisits` — czy ukryć wizyty w danej agendzie przed parafianami (np. przed publikacją planu, przy tworzeniu szkicu),
- `ShowHours` — czy pokazywać parafianom przewidywane godziny ich wizyty (dopiero gdy plan jest zatwierdzony).
- `IsOfficial` — czy pokazywać i liczyć agendę w statystykach jako pełnoprawną (wizyty w niej liczą się bez względu na tę właściwość).

BuildingAssignment Reprezentuje przypisanie konkretnego budynku do konkretnego dnia w planie kolędy (w ramach wybranego harmonogramu). Pozwala to na automatyczne sugerowanie terminów wizyt dla każdego zgłoszenia (na podstawie adresu oraz harmonogramu), a także automatyczny zapis przy rejestracji zgłoszenia.

Jedyne pole klasy `BuildingAssignment` to:

- `EnableAutoAssign` — czy włączyć automatyczne przypisywanie wizyt dla danego budynku w danym dniu (dla danego budynku w danym harmonogramie tylko jeden dzień może mieć włączony auto-zapis).

Tworzenie historii zmian Oprócz powyższych dwóch grup w modelu znajduje się również pięć klas pomocniczych, służących do rejestrowania historii zmian dla wybranych encji modelu. Trzy z nich to klasy typu *snapshot*: **SubmissionSnapshot**, **SubmitterSnapshot** oraz **VisitSnapshot**. Przechowują one kopię stanu odpowiadającej im encji w momencie dokonania zmiany wraz z informacją o autorze zmiany. Czwarta klasa to **FormSubmission**, która zachowuje kopię oryginalnych danych zgłoszenia (w momencie wprowadzenia do systemu). Pozwala na odtworzenie pierwotnej wersji zgłoszenia w przypadku nadużyć lub problemów. Ostatnia klasa to **EmailLog**, która rejestruje wysyłane wiadomości e-mail z aplikacji, wraz z ich zawartością i odbiorcami. Pozwala to na audyt i śledzenie komunikacji prowadzonej przez system, a także na ponawianie próby wysyłania wiadomości (gdy serwer pocztowy był niedostępny itp.).

Zarządzanie parafią W modelu znajdują się również dwie encje do zarządzania informacjami dot. parafii: **ParishMember** oraz **ParishInfo**.

Pierwsza z nich reprezentuje użytkownika systemu, odpowiadającego dokładnie jednemu użytkownikowi z kontekstu centralnego (klasa `User`) poprzez pole `CentralUserId`. Podczas gdy do celów zarządzania użytkownikami i pobierania ich informacji służy klasa `User` w kontekście centralnym, to klasa `ParishMember` pozwala na tworzenie relacji między użytkownikami (reprezentowanymi przez nią w kontekście parafii) a innymi encjami, np. planem kolędy (przypisanie księży do planu) lub agendą (przypisanie ministrantów do agendy).

Druga z nich jest prostym magazynem dla wszelakich informacji (w tym tych o parafii), które mogą być potrzebne w aplikacji, ale nie mają dedykowanej encji w modelu. Składa się z par klucz-wartość, gdzie klucz jest unikalnym identyfikatorem informacji, a wartość przechowuje jej treść.

Typy wyliczeniowe W modelu zastosowano również kilka typów wyliczeniowych (*enum*'ów), które służą do definiowania stałych wartości dla określonych właściwości encji.

SubmitMethod Określa metodę, za pomocą której parafianin złożył zgłoszenie na kolędę. Dostępne wartości to:

- **NotRegistered** — nie zarejestrowano (domyślnie),
- **PaperForm** — formularz papierowy,
- **WebForm** — formularz internetowy,
- **Phone** — telefonicznie,
- **Stationary** — osobiście (np. w kancelarii parafialnej),
- **Email** — mailowo,
- **DuringVisit** — osobiście podczas sąsiednich wizyt danego dnia.

NotesFulfillmentStatus Określa status realizacji dodatkowych uwag przez personel parafii. Dostępne wartości to:

- **NA** — nie dotyczy (domyślnie, gdy brak uwag),
- **Pending** — oczekujące (domyślnie gdy są uwagi),
- **Rejected** — niezrealizowane,
- **Accepted** — zrealizowane.

VisitStatus Określa status wizyty kolędowej. Dostępne wartości to:

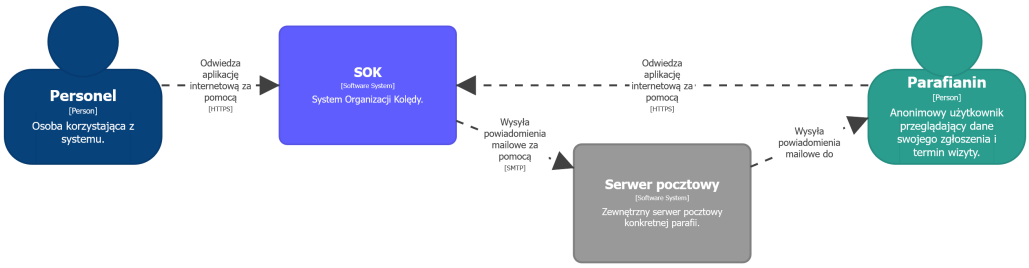
- **Unplanned** — niezaplanowana (domyślnie),
- **Planned** — zaplanowana (w agendzie),
- **Pending** — trwająca,
- **Visited** — zrealizowana (odbyta),

- **Rejected** — nieodbyta (np. parafianin nie otworzył drzwi),
- **Withdrawn** — wycofana (np. parafianin wycofał zgłoszenie),
- **Suspended** — wstrzymana (w szczególnych okolicznościach, stan tymczasowy podczas przeprowadzania wizyt).

3.3. Architektura

3.3.1. Poziom I — diagram kontekstowy systemu

Architektura aplikacji została zaprezentowana poniżej z użyciem modelu C4. Jego najwyższy poziom (*System Context Diagram*, rys. 3.4) przedstawia ogólny widok na system oraz jego interakcje z użytkownikami i zewnętrznymi systemami.



Rysunek 3.4: Diagram kontekstowy systemu

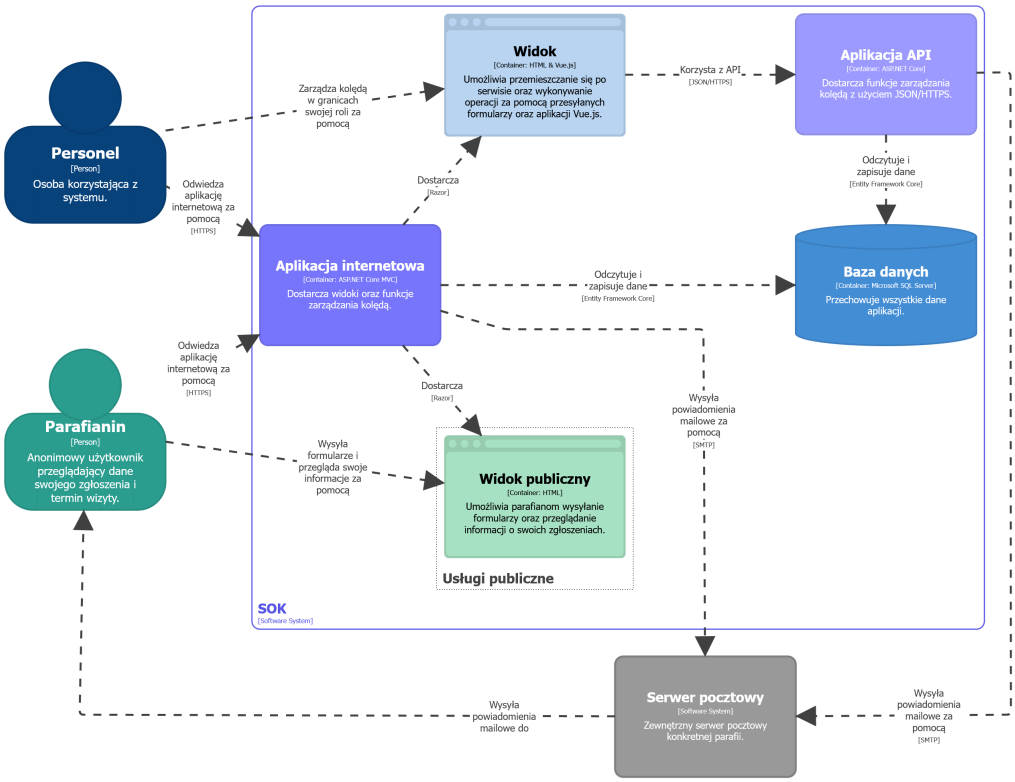
W naszym przypadku głównym komponentem jest aplikacja webowa **SOK**. Umożliwia ona zarządzanie kolędą w parafiach poprzez interfejs użytkownika dostępny z poziomu przeglądarki internetowej. Aplikacja komunikuje się dodatkowo z serwerem pocztowym (do wysyłania powiadomień e-mail).

Na diagramie zaznaczono również dwóch aktorów: **Parafianina** oraz **Personel parafialny**. Parafianin może składać zgłoszenia na kolędę oraz przeglądać swój panel zgłoszenia (wszystko jako anonimowy użytkownik). Personel parafialny zarządza zgłoszeniami i planuje wizyty kolędowe.

3.3.2. Poziom II — diagram kontenerów

Drugi poziom modelu C4 (*Container Diagram*, rys. 3.5) przedstawia główne kontenery aplikacji oraz ich interakcje.

Na tym poziomie widzimy rozróżnienie między personelem a parafianinami. Personel, odwiedzając aplikację webową, korzysta z pełnego interfejsu użytkownika, który udostępnia wszystkie funkcje aplikacji (w granicach roli). Parafianin natomiast korzysta z *widoków publicznych*, które są dostępne bez logowania. W dalszej



Rysunek 3.5: Diagram kontenerów aplikacji

części pracy odnosząc się do *widoku* będziemy mieli na myśli interfejs użytkownika zalogowanego.

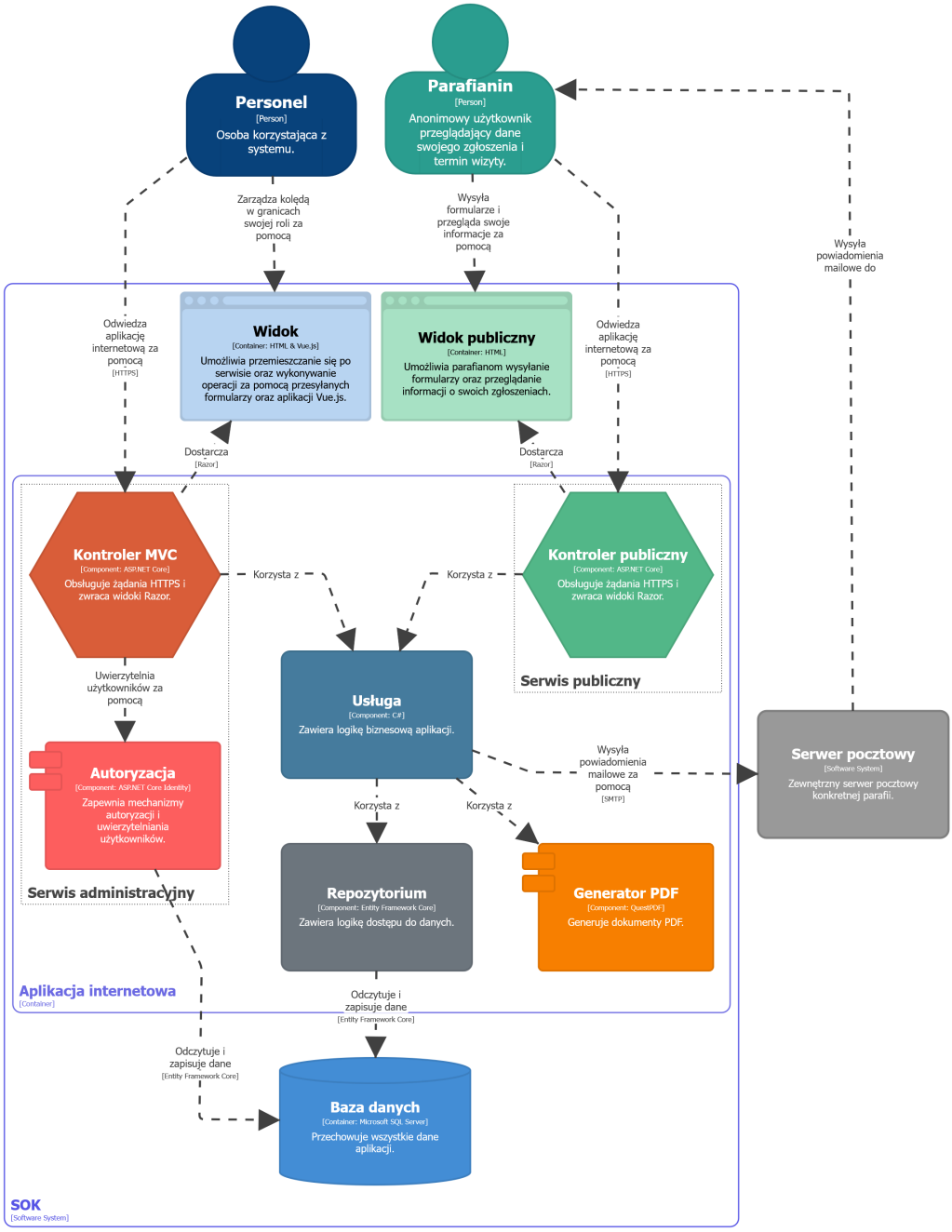
Widok aplikacji webowej może również korzystać z API, które udostępnia wybrane funkcje aplikacji w formie usług sieciowych (por. rozdział 3.1.1.). Aplikacja API, tak samo jak główny komponent aplikacji, komunikuje się z bazą danych oraz z serwerem pocztowym.

3.3.3. Poziom III — diagram komponentów

Aplikacja webowa

Aplikacja webowa jest zaprogramowana w architekturze MVC (Model-View-Controller). Podział ten widoczny jest na diagramie (rys. 3.6), choć nie jest wyraźny przy pierwszej analizie z powodu dodatkowych komponentów, które zostały zaprezentowane ze względu na ich istotną rolę w aplikacji.

Kontrolery Pierwszym rodzajem komponentu jest **Kontroler**, który obsługuje żądania HTTP od użytkowników i koordynuje przepływ danych między modelem a widokiem. W aplikacji znajdują się dwa rodzaje kontrolerów: kontrolery aplikacji



Rysunek 3.6: Diagram komponentów aplikacji webowej

(dla personelu parafialnego) oraz kontrolery publiczne (dla parafian).

Pierwszy z nich obsługuje wszystkie funkcje aplikacji dostępne dla zalogowanych użytkowników (w granicach ich ról). Korzysta przy tym z komponentu do autoryzacji, aby sprawdzić uprawnienia użytkownika przed wykonaniem danej akcji. Drugi z nich obsługuje funkcje dostępne bez logowania, tj. składanie zgłoszeń na kolędę oraz przeglądanie panelu zgłoszenia.

Widoki Kolejnym rodzajem komponentu są **Widoki**, które zawierają interfejs użytkownika aplikacji, prezentowany w przeglądarce. Widoki są tworzone przy użyciu technologii Razor, która umożliwia dynamiczne generowanie stron HTML na podstawie danych z modelu. Podobnie jak kontrolery, podzielone są na dwie kategorie: widoki dla zalogowanych użytkowników oraz widoki publiczne dla parafian.

Podczas gdy widoki publiczne są prostsze i raczej statyczne (głównie formularze i strony informacyjne), widoki dla zalogowanych użytkowników są często bardziej rozbudowane i interaktywne, oferując zaawansowane funkcje zarządzania kolędą. W sekcji 3.1.1. opisano podejścia wykorzystane do tworzenia interfejsu użytkownika w widokach aplikacji webowej.

Modele W architekturze MVC modele są przekazywane z kontrolerów do widoków, aby prezentować dane użytkownikowi. W opisywanej aplikacji jest to każdorazowo realizowane za pomocą jednego ze specjalnych typów modeli:

- **ViewModels** (*modele widoków*) — specjalne modele zaprojektowane do reprezentowania danych w widokach. Mogą zawierać dodatkową logikę prezentacyjną lub formatowanie danych.
- **DTO (Data Transfer Objects)** — proste obiekty, pochodzące bezpośrednio z warstwy aplikacji (por. sekcja 3.4.2.), które często są wystarczające do prezentacji danych.
- **Encje EF Core** — bezpośrednie modele danych z warstwy domeny. Stosowane głównie w prostych widokach, gdzie nie jest wymagana dodatkowa logika prezentacyjna.

Logika biznesowa Logika biznesowa aplikacji jest zaimplementowana w warstwie aplikacji (por. sekcja 3.4.2.) i jest wykorzystywana przez kontrolery aplikacji przy użyciu **Serwisów aplikacji**, reprezentowanych na diagramie (rys. 3.6) przez komponent *Usługi*. Serwisy aplikacji zawierają metody do wykonywania operacji biznesowych, takich jak zarządzanie zgłoszeniami, planowanie wizyt czy wysyłanie powiadomień e-mail. Kontrolery aplikacji nigdy nie komunikują się bezpośrednio z warstwą domeny lub infrastrukturą, lecz zawsze poprzez serwisy aplikacji.

Dostęp do danych Dostęp do danych w aplikacji jest realizowany za pomocą wzorca **Repozytoriów**, które są reprezentowane na diagramie przez komponent *Repozytoria*. Repozytoria zapewniają abstrakcję nad warstwą dostępu do danych, umożliwiając kontrolerom i serwisom aplikacji interakcję z bazą danych bez konieczności bezpośredniego korzystania z ORM (Entity Framework Core). Repozytoria zawierają metody do wykonywania operacji CRUD (tworzenie, odczyt, aktualizacja, usuwanie) na encjach domeny.

Repozytoria również są zebrane w jednym komponencie. W tym przypadku jednak ma to dodatkowy wymiar, ponieważ wszystkie one są dostępne poprzez jeden obiekt `UnitOfWork`, który dodatkowo koordynuje transakcje i zapewnia spójność danych.

Aplikacja API

Aplikacja API działa w podobny sposób jak aplikacja webowa, jednak jej głównym celem jest udostępnianie funkcji aplikacji w formie usług sieciowych (dostępnych poprzez protokół HTTP). Diagram komponentów aplikacji API został zaprezentowany na rys. 3.7.

Podobnie jak w aplikacji webowej, głównymi komponentami są tutaj **Kontrolery**, **Serwisy aplikacji** oraz **Repozytoria**. Kontrolery API obsługują żądania HTTP od klientów API (widoków aplikacji) i wykonują operacje biznesowe za pomocą serwisów aplikacji. Serwisy aplikacji i repozytoria działają identycznie jak w aplikacji webowej, zapewniając abstrakcję nad logiką biznesową i dostępem do danych. Jedyną istotną różnicą jest brak części publicznej, ponieważ aplikacja API jest przeznaczona wyłącznie do użytku przez widoki aplikacji webowej dla zalogowanych użytkowników.

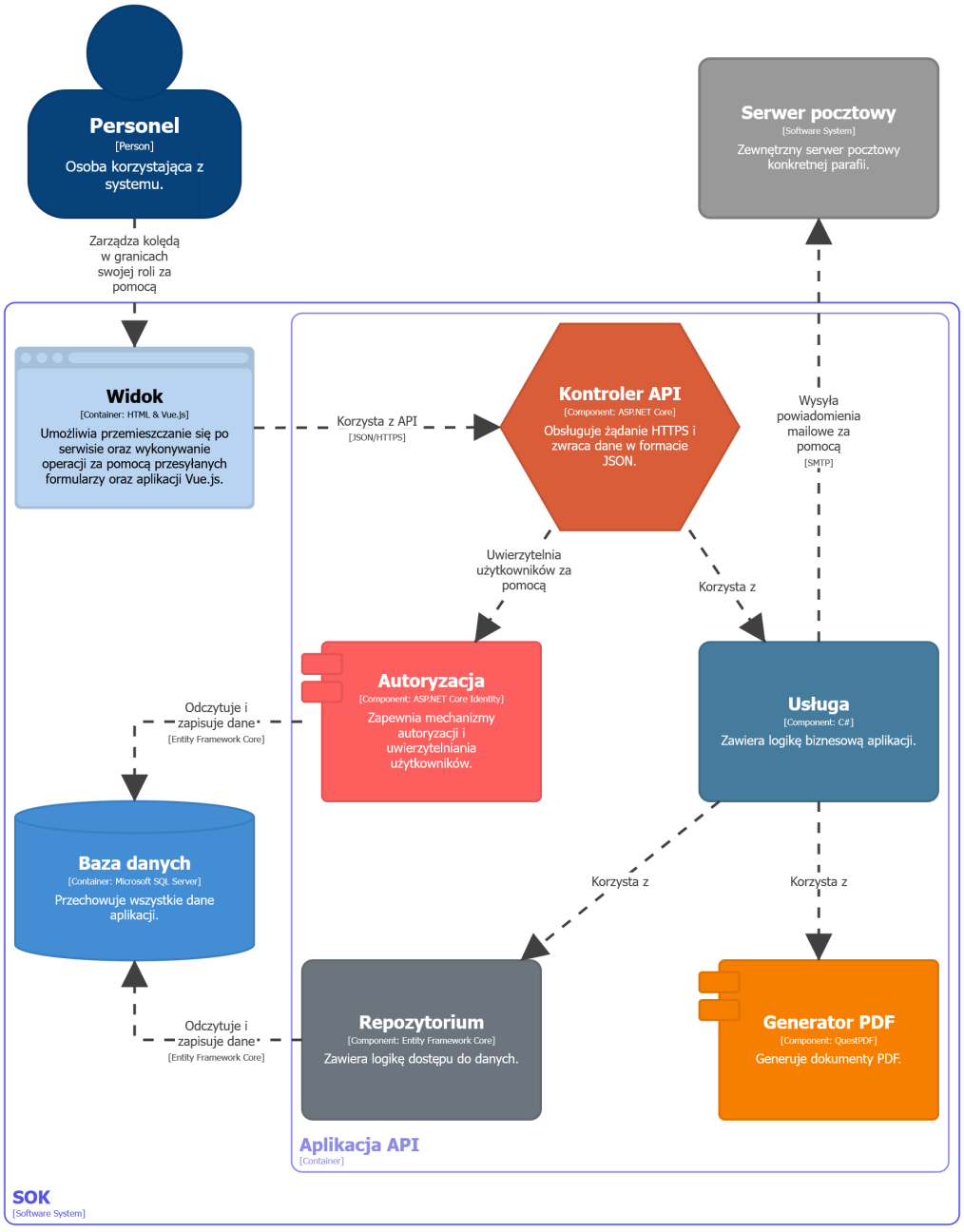
Aplikacja API udostępnia jedynie wybrane funkcje aplikacji, które bezpośrednio wspierają interfejs użytkownika. Z tego powodu nie wszystkie serwisy aplikacji i repozytoria są dostępne poprzez API. Również punkty końcowe często są dostępne tylko poprzez niektóre metody HTTP (spośród GET, POST, PUT, PATCH, DELETE).

3.4. DDD i warstwy systemu

Aplikację zaprojektowano i zaimplementowano w podejściu Domain-Driven Design (DDD), tworząc warstwową architekturę, która składa się z następujących poziomów:

- Warstwa domeny (Domain Layer)
- Warstwa aplikacji (Application Layer)
- Warstwa infrastruktury (Infrastructure Layer)
- Warstwa interfejsu użytkownika (UI Layer)

Każdy z nich jest odpowiedzialny za określone aspekty aplikacji i komunikuje się z innymi warstwami w sposób jasno zdefiniowany.



Rysunek 3.7: Diagram komponentów aplikacji API

3.4.1. Warstwa domeny

Najniższą warstwą w architekturze jest **Warstwa domeny**, która zawiera wszystkie elementy związane z modelem domeny aplikacji. Umieszczone zostały tutaj głównie encje domenowe (por. 3.2.2.). Warstwa domeny jest niezależna od innych warstw i nie zawiera żadnych odwołań do technologii zewnętrznych (np. baz danych, frameworków webowych itp.).

3.4.2. Warstwa aplikacji

Warstwa aplikacji znajduje się powyżej warstwy domeny i jest odpowiedzialna za implementację logiki biznesowej aplikacji. Zawiera serwisy aplikacji, które realizują operacje biznesowe, korzystając z encji domeny. Warstwa aplikacji komunikuje się z warstwą domeny poprzez bezpośrednie odwołania do encji oraz z warstwą infrastruktury poprzez repozytoria. Jest wykorzystywana przez warstwę interfejsu użytkownika i najczęściej przekazuje do niej dane w formie DTO (Data Transfer Objects).

3.4.3. Warstwa infrastruktury

Warstwa infrastruktury zapewnia implementację techniczną dla aplikacji. Zawiera realizację repozytoriów, które umożliwiają dostęp do bazy danych oraz inne komponenty infrastrukturalne, w tym komponent do wysyłania wiadomości e-mail czy komponent autoryzacji. Główną odpowiedzialnością warstwy infrastruktury jest izolowanie pozostałych warstw od szczegółów technicznych, takich jak konkretna baza danych czy protokoły komunikacyjne.

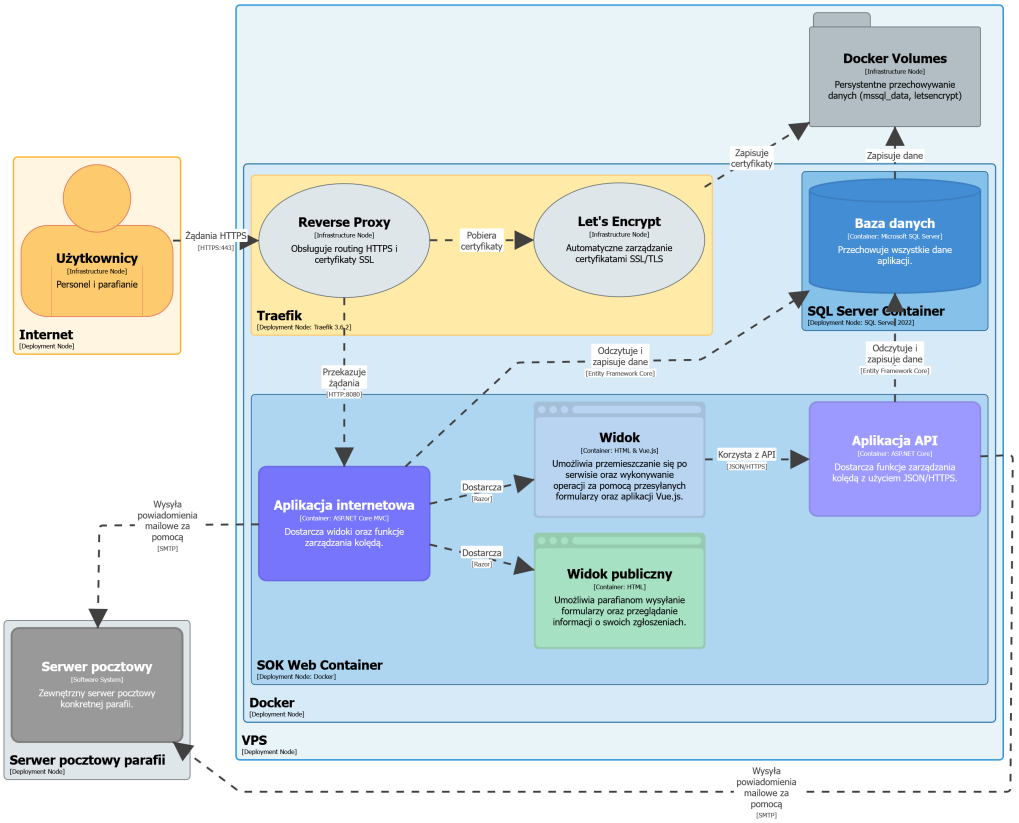
3.4.4. Warstwa UI

Najwyższym poziomem w architekturze jest **Warstwa interfejsu użytkownika**, która zawiera komponenty odpowiedzialne za interakcję z użytkownikami. W naszym przypadku są to aplikacja webowa oraz aplikacja API. Warstwa UI komunikuje się z warstwą aplikacji, korzystając z serwisów aplikacji i przekazuje dane do prezentacji użytkownikowi do widoków.

3.5. Architektura wdrażania

Jak wspomniano w sekcji 3.1.3. aplikacja została zaopatrzona w stosowny plik `Dockerfile`, który pozwala na zbudowanie obrazu aplikacji. Na jego podstawie można następnie tworzyć kontenery Dockera.

W projekcie umieszczono również plik `docker-compose.yml` (wraz z kilkoma odmianami), który pozwala na uruchomienie całego środowiska aplikacji (z aplikacją, bazą danych i menadżerem ruchu) za pomocą jednego polecenia z uwzględnieniem opcji konfiguracyjnych za pomocą orkiestratora Docker Compose.



Rysunek 3.8: Diagram wdrażania aplikacji

Na diagramie wdrażania (rys. 3.8) zaprezentowano strukturę środowiska aplikacji. Zaznaczone są na nim trzy wyżej wymienione usługi: aplikacja webowa, baza danych oraz menadżer ruchu (*reverse proxy*).

Konteneryzacja

Wszystkie trzy usługi są uruchamiane w oddzielnych kontenerach Dockera, co pozwala na ich izolację i łatwe zarządzanie. Konteneryzacja umożliwia również łatwe skalowanie aplikacji oraz przenoszenie jej między różnymi środowiskami (deweloperskim, testowym, produkcyjnym).

Przy zmianach w kodzie aplikacji lub jej konfiguracji wystarczy więc zbudować nowy obraz Dockera i uruchomić nowy kontener. Zachowuje się w ten sposób izolację od bazy danych, która nie odnotowuje żadnych przerw w działaniu.

Zarządzanie siecią

W środowisku aplikacji zastosowano również wirtualną sieć Dockera, która pozwala na zachowanie bezpośredniej komunikacji między kontenerami przy pełnej ich izolacji od sieci zewnętrznej. Kontenery mogą się ze sobą komunikować za pomocą nazw usług (np. `database` dla bazy danych), co upraszcza konfigurację połączeń.

Jedynym punktem dostępu do aplikacji z zewnątrz jest menadżer ruchu, który przekazuje żądania do aplikacji webowej. Pozwala to na dodatkowe zabezpieczenie aplikacji oraz na łatwe zarządzanie ruchem sieciowym (np. poprzez konfigurację certyfikatów SSL/TLS).

Trwałość danych

Baza danych jest skonfigurowana z użyciem wolumenu Dockera, który zapewnia trwałość danych pomiędzy restartami kontenera. Oznacza to, że dane przechowywane w bazie danych nie zostaną utracone w przypadku ponownego uruchomienia kontenera lub aktualizacji aplikacji. Wolumen jest mapowany na lokalny katalog na hoście, co pozwala na łatwe tworzenie kopii zapasowych i zarządzanie danymi bazy.

Środowisko produkcyjne

Dzięki zastosowaniu powyższego zestawu technologii i podejść architektonicznych aplikacja jest łatwa do wdrożenia i utrzymania na dowolnym serwerze obsługującym Dockera, w szczególności na popularnych platformach chmurowych. Nie jest jednak do nich ograniczone — cały stos technologiczny może być uruchomiony na dowolnym serwerze fizycznym lub wirtualnym, do którego użytkownik ma dostęp, za pomocą jednego polecenia (bez konieczności manualnego konfigurowania środowiska — instalowania oprogramowania i jego zależności).

3.6. Opis rozwiazania wybranych problemów

3.6.1. Dostęp do bazy danych

W aplikacji ze względu na wielodomenowość (wzorzec *multi-tenant*) zastosowano izolację danych na poziomie bazy danych. Oznacza to, że każda parafia posiada oddzielną bazę danych, co zapewnia pełną separację danych między parafiami i zwiększa bezpieczeństwo. Aby umożliwić aplikacji dostęp do odpowiedniej bazy danych w zależności od parafii, zastosowano dynamiczne tworzenie ciągów połączeń w kontekście *EF Core*.

Określenie parafii

Pierwszym elementem mechanizmu uzyskiwania dostępu do odpowiedniej bazy danych jest *middleware* `ParishResolver`, umieszczone w potoku przetwarzania. W nim pobierane są informacje o parafii (z ciasteczka lub parametru adresu URL).

Pobranie informacji o parafii

Następnie `ParishResolver` korzysta z serwisu `ICurrentParishService`, który umożliwia zapisywanie i pobieranie informacji o aktualnie wybranej parafii w trakcie trwania żądania HTTP. Serwis ten ma zasięg *scoped*, dzięki czemu jego stan jest zachowywany tylko w trakcie przetwarzania danego żądania.

Główną funkcją serwisu jest pobranie informacji o parafii z centralnej bazy danych, odszyfrowanie ciągu połączenia i zapisanie go w swoim stanie. W ten sposób pozostaje on dostępny dla pozostałych komponentów aplikacji w trakcie przetwarzania żądania.

Utworzenie kontekstu

Po wykonaniu powyższych kroków następuje utworzenie kontekstu bazy danych `ParishDbContext` w kontenerze *DI*. Wtedy wywoływana jest nadpisana metoda `OnConfiguring`, w której pobierany jest ciąg połączenia z serwisu `ICurrentParishService` i konfigurowany jest kontekst do korzystania z odpowiedniej bazy danych.

Dalej w aplikacji wszystkie operacje na bazie danych są wykonywane za pomocą kontekstu `ParishDbContext`, który jest już poprawnie skonfigurowany do komunikacji z bazą danych odpowiedniej parafii.

3.6.2. Zarządzanie parafiami

Architektura

Ze względu na izolację danych na poziomie bazy danych, tworzenie nowej parafii wymaga utworzenia nowej bazy danych oraz odpowiedniej jej konfiguracji, włączając utworzenie użytkownika bazy i wykonanie wszystkich należnych migracji. Aby zautomatyzować ten proces, w aplikacji zaimplementowano serwis `IParishProvisioningService`, który wykonuje wszystkie niezbędne kroki. Posiada on dwie metody publiczne: `CreateParishAsync` oraz `EnsureAllParishDatabasesReadyAsync`.

Pierwsza z nich służy do utworzenia bazy danych na poziomie serwera baz danych i jest wywoływana przy rejestracji nowej parafii. Druga z nich jest wykorzystywana podczas uruchamiania aplikacji, sprawdzając, czy wszystkie bazy danych parafii (zapisanych w centralnej bazie danych) są poprawnie skonfigurowane i gotowe

do użycia. Obie metody korzystają ze wspólnej metody prywatnej, która działa w następujących krokach:

- Połączenie z serwerem baz danych przy użyciu konta administracyjnego,
- Utworzenie nowej bazy danych o unikalnej nazwie (z losowym sufiksem),
- Utworzenie nowego użytkownika z unikalną nazwą (z losowym sufiksem) oraz przypisanie mu odpowiednich uprawnień do tej bazy danych,
- Wykonanie wszystkich (nowych) migracji Entity Framework Core na tej bazie danych,
- Zapisanie informacji o parafii w centralnej bazie danych, w tym zaszyfrowanego ciągu połączenia do tej bazy danych.

Gdy metoda ma utworzyć parafię, wszystkie kroki wykonają się bezwarunkowo, gdy jednak służy do sprawdzenia istnienia bazy danych, to pomija adekwatne kroki w razie braku konieczności ich wykonania.

W ten sposób proces tworzenia parafii jest w pełni zautomatyzowany i nie wymaga ręcznej interwencji administratora bazy danych. Używa przy tym niskopoziomowych poleceń SQL do zarządzania bazami danych i użytkownikami, co zapewnia pełną kontrolę nad procesem.

Interfejs użytkownika

Aby zapewnić administratorowi całego systemu sprawne zarządzanie wieloma parafiami, umieszczono go w dedykowanej roli i pozbawiono przypisania do jednej, konkretnej parafii, a umożliwiono dynamiczny wybór parafii do zarządzania w trakcie korzystania z aplikacji. W tym celu w interfejsie użytkownika zaimplementowano specjalny widok wyboru parafii, który jest wyświetlany administratorowi zaraz po zalogowaniu się do aplikacji. Widok ten pobiera listę wszystkich parafii z centralnej bazy danych i pozwala na wybranie jednej z nich do zarządzania. Po wybraniu parafii, aplikacja zapisuje jej identyfikator w ciasteczku, dzięki któremu użytkownik ten przy następnym żądaniu zostanie potraktowany jako administrator danej parafii.

3.6.3. System powiadomień

W celu zachowania jednolitości aplikacji i jej przystępności dla przeciętnego użytkownika, został zaimplementowany prosty, lecz elastyczny i efektywny system powiadomień wewnętrznych, który umożliwia wyświetlanie komunikatów o różnym stopniu ważności w interfejsie użytkownika.

Cały system zaimplementowany jest jako obiekt *JavaScript*, udostępniający metody do tworzenia różnego rodzaju powiadomień. Obiekt ten jest obiektem globalnym, dostępnym w dowolnym miejscu części frontendowej. Każde powiadomienie wyświetla się przez pewien czas po jego utworzeniu, a następnie znika automatycznie. Jedynym warunkiem jest umieszczenie w układzie strony specjalnego kontenera (`div` o identyfikatorze `notification-container`), do którego będą dodawane elementy powiadomień. Pozwala to na łatwe zarządzanie wyglądem i pozycjonowaniem powiadomień.

Dodatkowo, aby umożliwić komunikację z części backendowej aplikacji, w widoku częściowym umieszczono specjalny mechanizm, odczytujący komunikaty z obiektu *TempData* i mapujący je na wywołania metod obiektu powiadomień w JavaScript. Dzięki temu kontrolery aplikacji mogą tworzyć powiadomienia, które zostaną wyświetlone po załadowaniu widoku przez użytkownika.

3.6.4. Właściwości *cache*'owane

W celu optymalizacji wydajności aplikacji i zmniejszenia obciążenia bazy danych umieszczono w encji `Address` cztery właściwości, które są *cache*'owane (przechowywane w bazie danych, ale wyliczane na podstawie innych danych). Są to:

- `BuildingNumber`,
- `BuildingLetter`,
- `StreetName`,
- `StreetType`,
- `CityName`.

Właściwości te odpowiadają wartościom z powiązanych encji `Building` oraz, dalej, `Street`, `StreetSpecifier` i `City`. Ich spójność została zapewniona po stronie bazy danych poprzez zastosowanie wyzwalaczy (*triggers*), które aktualizują te właściwości przy każdej zmianie powiązanych encji. Dzięki temu aplikacja może szybko uzyskać dostęp do tych wartości bez konieczności wykonywania złożonych zapytań z łączeniami (*joins*) do wielu tabel. Wyzwalacze te zostały manualnie umieszczone w jednej z migracji *EF Core*, aby zapewnić ich obecność w bazie danych od samego początku istnienia aplikacji.

Obecność właściwości *cache*'owanych nie tylko znacząco redukuje liczbę złączeń w zapytaniach do bazy danych, ale także pozwala na automatyczne obliczenie pola `FilterableString`, które jest wykorzystywane do szybkiego wyszukiwania zgłoszeń na podstawie adresu. Pole to jest indeksowane w bazie danych, co dodatkowo przyspiesza operacje wyszukiwania. Sama logika aktualizacji pola `FilterableString` również leży po stronie bazy danych, tym razem jednak na podstawie konfiguracji

encji w *EF Core*, która generuje odpowiednie polecenia SQL do aktualizacji tego pola przy każdej zmianie zgłoszenia.

Rozdział 4.

Porównanie z innymi implementacjami

Obecnie rynek aplikacji do organizacji wizyt duszpasterskich nie jest wcale rozwinięty. Parafie, które decydują się na przeprowadzanie wizyty duszpasterskiej w modelu kolędy na zaproszenie, najczęściej wybierają rozwiązania albo prowizoryczne, albo prywatne, tworzone i nadzorowane przez parafian na własną rękę. Nie istnieją jeszcze żadne aplikacje komercyjne dedykowane do tego celu.

4.1. System manualny

Najmniej technicznym rozwiązaniem jest podejście manualne, polegające na prowadzeniu listy wizyt w formie papierowej. Często dopuszcza również zgłoszenia internetowe (np. poprzez formularz Google), jednak ostateczna lista wizyt jest tworzona ręcznie przez duszpasterza lub inną osobę odpowiedzialną za organizację kolędy na papierze, czy to poprzez układanie karteczek z nazwiskami na stole (w dużych parafiach po całych pomieszczeniach), czy też poprzez wypisywanie listy na tablicy. Takie podejście jest jednak bardzo niewygodne i podatne na błędy. W przypadku dużej liczby zgłoszeń łatwo o przeoczenie lub podwójne zapisanie wizyty. Ponadto, w przypadku konieczności zmiany terminu wizyty, cała lista musi zostać zmodyfikowana ręcznie, co prowadzi do dodatkowych komplikacji. Ostatecznie oprócz czasu i wysiłku zajmuje to dużo przestrzeni fizycznej, a lista papierowa może zostać łatwo zgubiona lub uszkodzona.

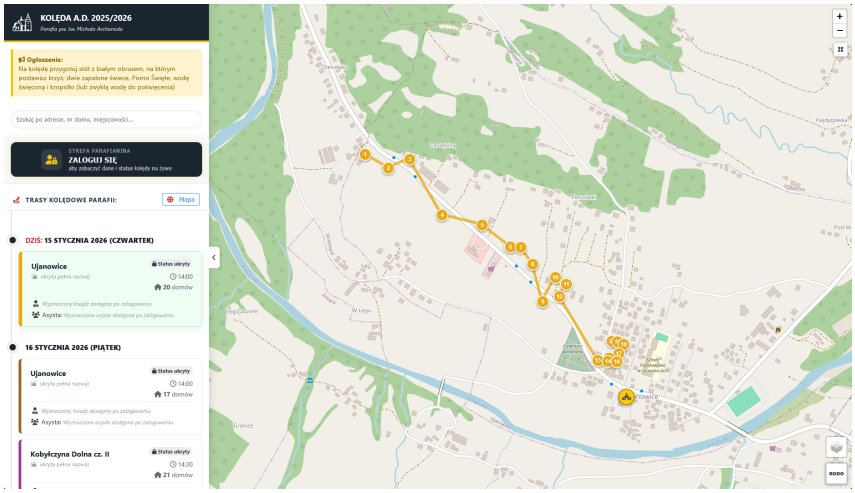
4.2. System półautomatyczny

Niektóre parafie decydują się na wykorzystanie arkuszy kalkulacyjnych (np. Microsoft Excel lub Google Sheets) do zarządzania listą wizyt duszpasterskich. W tym

podejściu lista wizyt jest tworzona i modyfikowana cyfrowo, co pozwala na łatwiejsze wprowadzanie zmian i zmniejsza ryzyko błędów związanych z ręcznym zapisywaniem. Arkusze kalkulacyjne oferują funkcje sortowania i filtrowania danych, co ułatwia organizację wizyt według różnych kryteriów, takich jak data czy ulica. Jednakże, mimo że podejście to jest bardziej zaawansowane niż prowadzenie listy na papierze, nadal wymaga ręcznego wprowadzania danych i aktualizacji listy. Ponadto, arkusze kalkulacyjne nie oferują dedykowanych funkcji do zarządzania wizytami duszpasterskimi, co może prowadzić do nieefektywności i komplikacji w organizacji kolędy.

4.3. *Adventus*

W bieżącym roku (2025) w parafii w Ujanowicach (archidiecezja krakowska) została wdrożona aplikacja webowa *Adventus* ([23]), stworzona przez zespół młodzieżowych programistów z tej parafii. Aplikacja ta umożliwia parafianom śledzenie zaplanowanej trasy kolędowej na interaktywnej mapie. Jest ona publicznie dostępna, jednak w celu sprawdzenia szczegółów wizyty (takich jak data, wyznaczony duszpasterz itp.) wymagana jest autentykacja parafianina poprzez podanie adresu i nazwiska rodziny.



Rysunek 4.1: Mapa trasy kolędowej w aplikacji *Adventus*

Ze względu na zamknięty charakter aplikacji oraz brak dostępu do jej kodu źródłowego, nie jest wiadome, czy posiada ona funkcje takie jak planowanie wizyt w sposób w pełni cyfrowy, czy też wymaga ręcznego przenoszenia danych do innego systemu przez administratora. Nie jest też jasne, czy aplikacja oferuje takie funkcjonalności jak:

- Przewidywanie godzin wizyty,
- Powiadamianie mailowe,

- Zarządzanie harmonogramami.

W systemie funkcjonuje jednak oznaczanie statusów wizyt na bieżąco przez ministrantów podczas trwania kolędy, co jest źródłem aktualnych informacji dla parafian ([24]).

Główną różnicą między systemem *Adventus* a opisaną w tej pracy aplikacją jest to, że *Adventus* skupia się głównie na udostępnianiu informacji parafianom, podczas gdy aplikacja opisana w tej pracy ma na celu ułatwienie organizacji i zarządzania wizytami duszpasterskimi od strony administracyjnej, co w efekcie poprawia również doświadczenie parafian.

Warto też zwrócić uwagę na różnicę w podejściu do autentykacji użytkowników. W aplikacji *Adventus* parafianie muszą podać swoje dane (adres i nazwisko) w celu uzyskania dostępu do szczegółów wizyty, co może budzić obawy dotyczące prywatności i bezpieczeństwa danych, zwłaszcza w małych miejscowościach, gdzie wszyscy się znają. Dodatkowo publicznie udostępnione są informacje o trasie kolędowej, co może prowadzić do różnych niepożądanych, a czasem i niebezpiecznych sytuacji (szczególnie w obszarach o większym zagęszczeniu ludności).

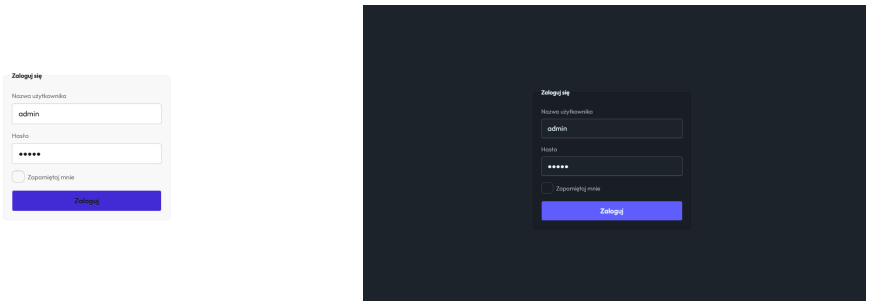
Rozdział 5.

Instrukcja użytkownika

Instrukcja opisuje podstawowe funkcje aplikacji oraz sposób jej obsługi przez administratora. Wszystkie zrzuty ekranów przedstawione w instrukcji z aplikacji uruchomionej w środowisku deweloperskim z domyślnymi zmiennymi środowiskowymi (w szczególności z przykładowymi danymi bazy danych).

Logowanie

Po uruchomieniu aplikacji użytkownik zostaje przekierowany na stronę logowania (rys. 5.1). W celu zalogowania się należy podać nazwę użytkownika oraz hasło i kliknąć przycisk *Zaloguj się*.



(a) Wariant jasny (b) Wariant ciemny

Rysunek 5.1: Strona logowania

Domyślnie zarówno nazwa użytkownika, jak i hasło, to `admin`. Te wartości można (i zaleca się) skonfigurować przed pierwszym uruchomieniem aplikacji poprzez plik zmiennych środowiskowych (por. sekcja 6.1.1.).

W zależności od ustawień przeglądarki, aplikacja może być wyświetlana w wariancie jasnym (rys. 5.1a) lub ciemnym (rys. 5.1b). W dalszej części instrukcji pokazywany jest wariant jasny. W celu zmiany wariantu kolorystycznego można skorzystać z przycisku zmiany motywu dostępnego w rozwijanym menu użytkownika w

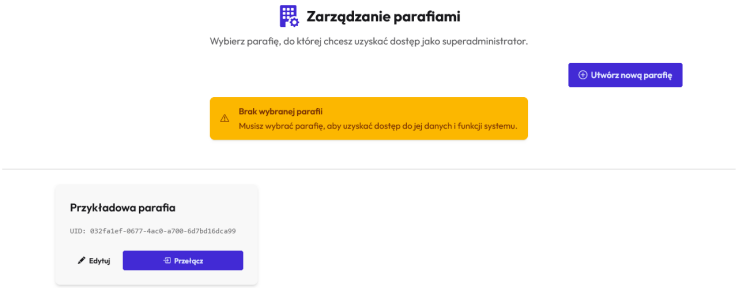
prawym górnym rogu ekranu po zalogowaniu (należy kliknąć na nazwę użytkownika).

Przy pierwszym uruchomieniu aplikacji dostępne jest tylko konto administratora o danych logowania zgodnych z podanymi w pliku konfiguracyjnym `.env`. Domyślnie są to:

- Nazwa użytkownika: `admin`
- Hasło: `admin`

Zarządzanie parafiami

Z racji logowania na konto superadministratora w pierwotnym stanie aplikacji po zalogowaniu zostanie wyświetlona strona zarządzania parafiami (rys. 5.2). Superadministrator nie jest przypisany do żadnej parafii, lecz może załadować się do dowolnej parafii w tym widoku.



Rysunek 5.2: Zarządzanie parafiami

Dostępna jest lista wszystkich parafii zarejestrowanych w systemie. Domyślnie utworzona została przykładowa parafia z przypisanym do niej jednym użytkownikiem — administratorem. Aby załadować się do danej parafii, należy kliknąć przycisk *Przełącz* w jej kafelku[1].

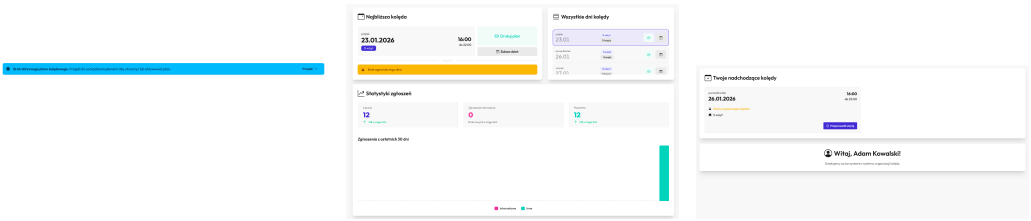
W tym widoku dostępny jest też przycisk *Utwórz nową parafię*, który przekierowuje do formularza tworzenia nowej parafii. Jego przesłanie spowoduje dodanie wpisu do listy parafii oraz utworzenie nowej bazy danych dla niej.

Strona główna

Normalny użytkownik po zalogowaniu zostaje przekierowany na stronę główną aplikacji. Jako superadministrator można również do niej wejść przez opcję *Przegląd* w menu bocznym (oznaczoną przez ikonkę domu), pod warunkiem uprzedniego

[1]Spowoduje to wydanie ciasteczka z identyfikatorem parafii, co pozwoli na dostęp do zasobów tej parafii po zalogowaniu. Po usunięciu danych aplikacji i restarcie może być potrzebne usunięcie ciasteczek w przeglądarce dla zapewnienia poprawnego działania.

wybrania parafii. W zależności od roli użytkownika, na stronie głównej dostępne są różne opcje menu.



(a) Pierwsze wejście (b) Widok administratora (c) Widok ministranta

Rysunek 5.3: Strona główna

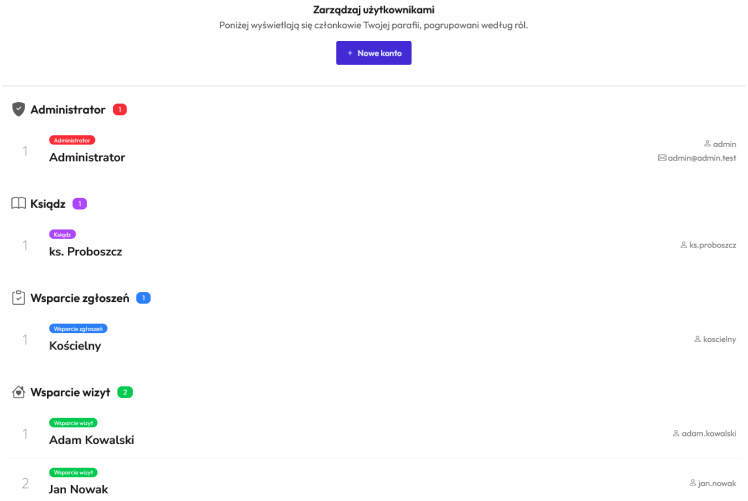
Przy pierwszym wejściu nie ma żadnego aktywnego planu, a więc na stronie głównej widoczny jest tylko komunikat z przekierowaniem (rys. 5.3a).

Po utworzeniu i aktywowaniu planu na stronie głównej pojawią się kafelki z podsumowaniem najbliższego dnia kolędowego, listą wszystkich dni kolędowych oraz ze statystykami zgłoszeń (rys. 5.3b).

W widoku ministranta (rys. 5.3c) zamiast tych kafelków wyświetlają się kafelki z listą przydzielonych mu nadchodzących agend oraz możliwością szybkiego przejścia do przeprowadzania wizyt.

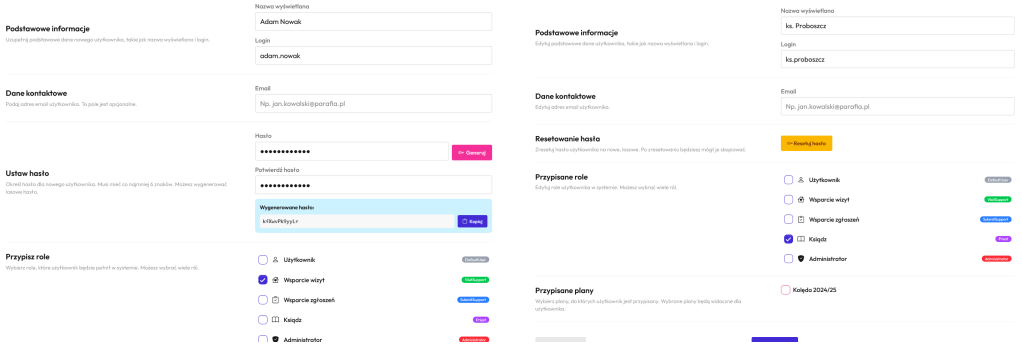
Zarządzanie użytkownikami

Aby przejść do zarządzania użytkownikami, należy w bocznym menu wybrać opcję *Ustawienia* (oznaczoną przez ikonę koła zębatego), a następnie zakładkę *Użytkownicy*. Wyświetli się wtedy strona z listą wszystkich użytkowników (rys. 5.4 pokazuje stan aplikacji z większą liczbą użytkowników).



Rysunek 5.4: Lista użytkowników

Po kliknięciu przycisku *Nowe konto* zostanie wyświetlony formularz tworzenia nowego użytkownika (rys. 5.5a). W celu edycji istniejącego użytkownika należy kliknąć na niego. Spowoduje to wyświetlenie formularza edycji użytkownika (rys. 5.5b).

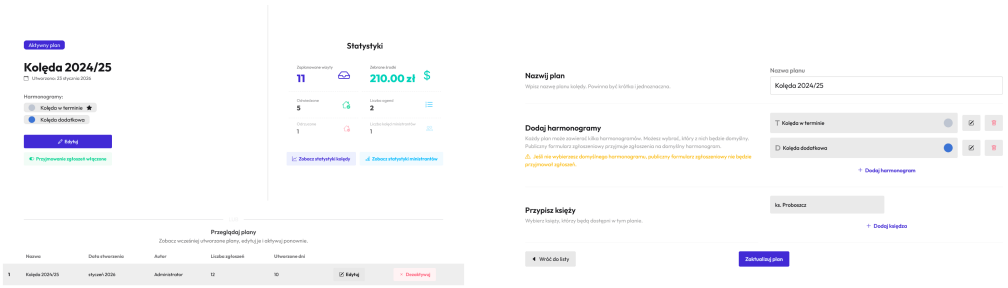


(a) Tworzenie użytkownika (b) Edycja użytkownika

Rysunek 5.5: Zarządzanie użytkownikami

Zarządzanie planami

Aby przejść do zarządzania planami, należy w bocznym menu wybrać opcję *Plany* (oznaczoną przez ikonę trzech poziomych kresek kaskadowych). Wyświetli się wtedy strona z możliwością utworzenia nowego planu oraz listą wszystkich planów (rys. 5.6a).



(a) Lista planów (b) Edycja planu

Rysunek 5.6: Zarządzanie planami

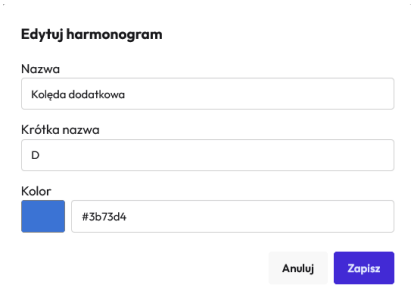
Aby aktywować dany plan, należy kliknąć przycisk *Aktywuj* w jego wierszu na liście planów. Spowoduje to, że plan stanie się aktualnym planem używanym przez aplikację, a wszelkie zgłoszenia będą wpływać na niego.

W głównej części widoku dostępne są szczegóły aktywnego planu, przycisk edycji oraz możliwość przełączania przyjmowania zgłoszeń przez formularz publiczny. Jest on zablokowany, jeśli plan nie ma wybranego domyślnego harmonogramu (wówczas i tak nie przyjmuje zgłoszeń). Po prawej stronie prezentowane są podstawowe statystyki planu oraz przyciski z odnośnikami do widoków z bardziej szczegółowymi statystykami.

Aby utworzyć nowy plan, należy kliknąć przycisk *Utwórz* (dostępny gdy nie ma aktywnego planu), co spowoduje wyświetlenie formularza tworzenia planu. W celu edycji istniejącego planu należy kliknąć na jego nazwę na liście. Spowoduje to wyświetlenie formularza edycji planu (rys. 5.6b).

Formularze tworzenia i edycji planu wyglądają identycznie. W formularzu należy podać nazwę planu oraz dostępne harmonogramy i wybrać harmonogram domyślny (rys. 5.7a) i przypisanych księży. Jedynie w miejscu można zarządzać harmonogramami planu (dodawać, edytować i usuwać harmonogramy — rys. 5.7b).

(a) Wybór harmonogramu domyślnego



(b) Edycja harmonogramu

Rysunek 5.7: Zarządzanie planami — szczegóły

Przyjmowanie zgłoszeń

W aplikacji istnieją dwa sposoby rejestracji zgłoszeń:

- poprzez publiczny formularz zgłoszeniowy,
- poprzez panel administratora.

Publiczny formularz zgłoszeniowy dostępny jest pod adresem:

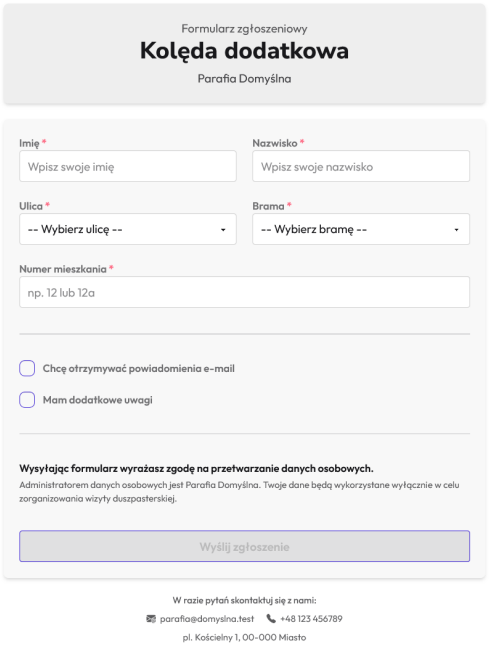
```
https://<adres_aplikacji>/<uid_parafii>/submissions/new
```

W powyższym `<adres_aplikacji>` to adres, pod którym dostępna jest aplikacja, a `<uid_parafii>` to unikalny identyfikator parafii, który jest generowany losowo przy tworzeniu parafii, a dostępny w *Ustawieniach* w zakładce *Ustawienia ogólne*.

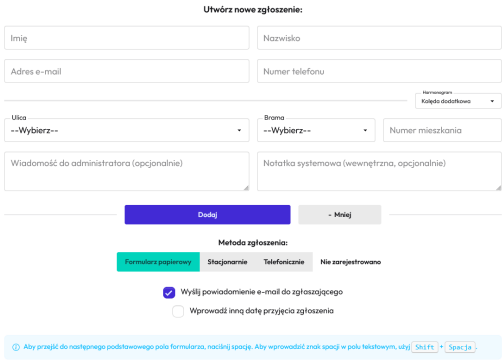
Aby ułatwić dostęp do formularza dla personelu parafialnego, można przejść do niego przechodząc do opcji menu bocznego *Ustawienia*, a następnie wybierając zakładkę *Formularz zgłoszeniowy*.

W formularzu zgłoszeniowym (rys. 5.8a) należy podać dane zgłaszającego — imię, nazwisko, adres e-mail (opcjonalnie) oraz adres. Jeśli został skonfigurowany automatyczny zapis, to wyświetli się tam informacja o dacie kolędy, na którą zgłaszający zostanie zapisany po wysłaniu zgłoszenia (jeśli jest dostępna dla danego adresu).

W formularzu dostępne są do wyboru tylko bramy, które zostały wcześniej utworzone w systemie (patrz sekcja *Zarządzanie adresami*) i są bramami widocznymi.



(a) Publiczny formularz zgłoszeniowy



(b) Wewnętrzny formularz zgłoszeniowy

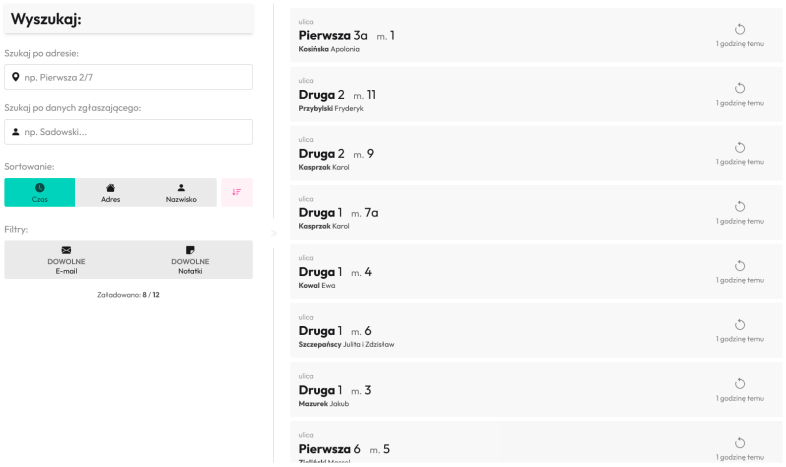
Rysunek 5.8: Formularze zgłoszeniowe

Wewnętrzny formularz zgłoszeniowy dostępny jest w opcji menu bocznego *Utwórz zgłoszenie* (oznaczoną przez ikonę plusa w przerywanym kwadracie). W nim oprócz podstawowych informacji można od razu uzupełnić dodatkowe dane zgłoszenia, takie wewnętrzne notatki, numer telefonu, sposób złożenia zgłoszenia itp. (rys. 5.8b).

Edycja zgłoszeń

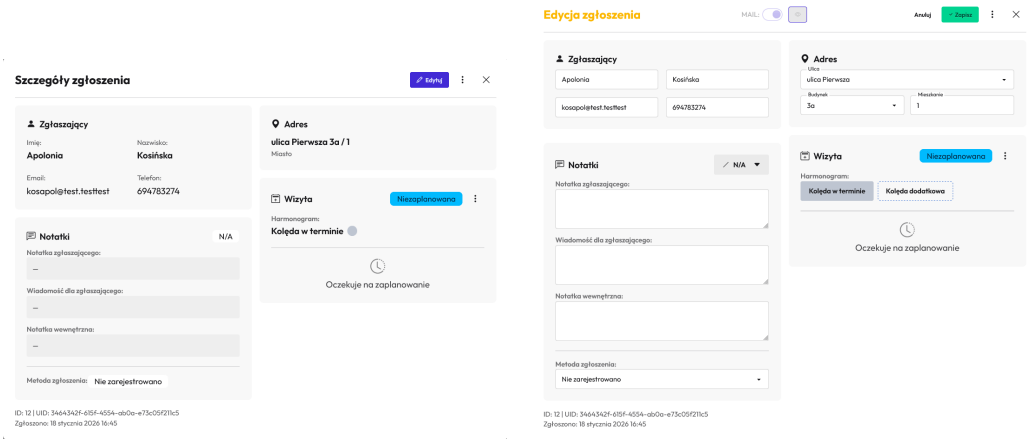
Po rejestracji zgłoszenia można je znaleźć na liście zgłoszeń dostępnej w opcji menu bocznego *Zgłoszenia* (oznaczoną przez ikonę koperty z wystającą kartką). W tym widoku (rys. 5.9) po lewej stronie ekranu dostępne są filtry umożliwiające zawężenie wyników wyszukiwania, a po prawej stronie znajduje się lista zgłoszeń spełniających kryteria wyszukiwania i sortowania.

Kliknięcie na zgłoszenie spowoduje wyświetlenie szczegółów zgłoszenia (rys. 5.10a). W tym widoku można zapoznać się ze wszystkimi danymi zgłoszenia oraz je edytować poprzez kliknięcie przycisku *Edytuj* w prawym górnym rogu okienka. W trybie edycji (rys. 5.10b) można zmieniać wszystkie dane zgłoszenia oraz zapisać zmiany poprzez kliknięcie przycisku *Zapisz*. Jeśli wówczas podany jest adres e-mail, to na górnym pasku wyświetla się również przełącznik wysyłania powiadomienia e-mail o zmianie danych zgłoszenia do zgłaszającego. Podgląd wysyłanej wiadomości można zobaczyć klikając przycisk obok przełącznika z ikonką oka (lub w menu rozwijanym w prawym górnym rogu okienka) — o ile wprowadzone zostały zmiany, kwalifikujące



Rysunek 5.9: Lista zgłoszeń

się do wysłania powiadomienia.



(a) Szczegóły zgłoszenia (b) Edycja zgłoszenia

Rysunek 5.10: Zarządzanie zgłoszeniami

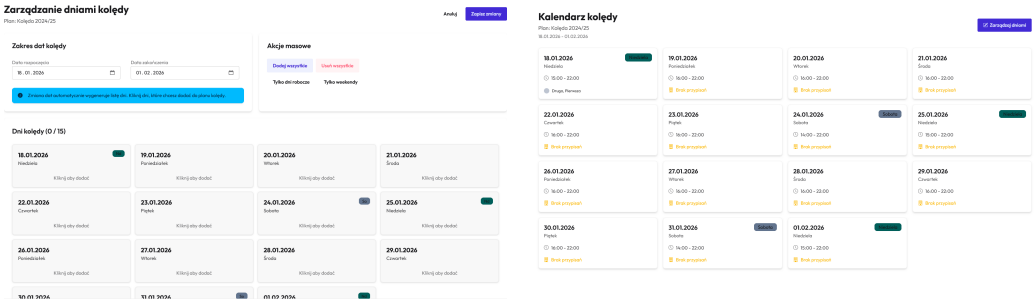
W menu rozwijanym w prawym górnym rogu okienka są dostępne także inne opcje, takie jak wysyłanie powiadomień e-mail do zgłaszającego (jeśli podany jest adres e-mail), przejście do panelu zgłoszenia (dostępnego dla zgłaszającego) lub dostęp do oryginalnych danych zgłoszenia (przechwyconych w momencie rejestracji zgłoszenia).

W kafelku wizyty dostępne jest również rozwijane menu (w prawym górnym rogu), w którym jest opcja anulowania wizyty/zgłoszenia (wówczas zgłoszenie staje się niezaplanowane, przestaje być wyświetlane w większości miejsc, a na liście zgłoszeń ma odpowiednie oznaczenie), lub przywrócenia anulowanego zgłoszenia.

Jeśli zgłoszenie jest zaplanowane w jakiejś agendzie, to w kafelku wizyty dostępny jest przycisk *Przejdź do agendy*, który przenosi do edytora tej agendy.

Zarządzanie dniami kolędowymi

Aby przejść do zarządzania dniami kolędowymi, należy w bocznym menu wybrać opcję *Kalendarz* (oznaczoną przez ikonę kalendarza). Jeśli w danym planie nie są jeszcze skonfigurowane daty rozpoczęcia i zakończenia kolędy, to zostanie wyświetlony formularz ich dodania (rys. 5.11a). W przeciwnym wypadku pokazana zostanie lista wszystkich dni kolędowych (rys. 5.11b).



(a) Tworzenie i edycja dni kolędowych (b) Lista dni kolędowych

Rysunek 5.11: Zarządzanie dniami kolędowymi

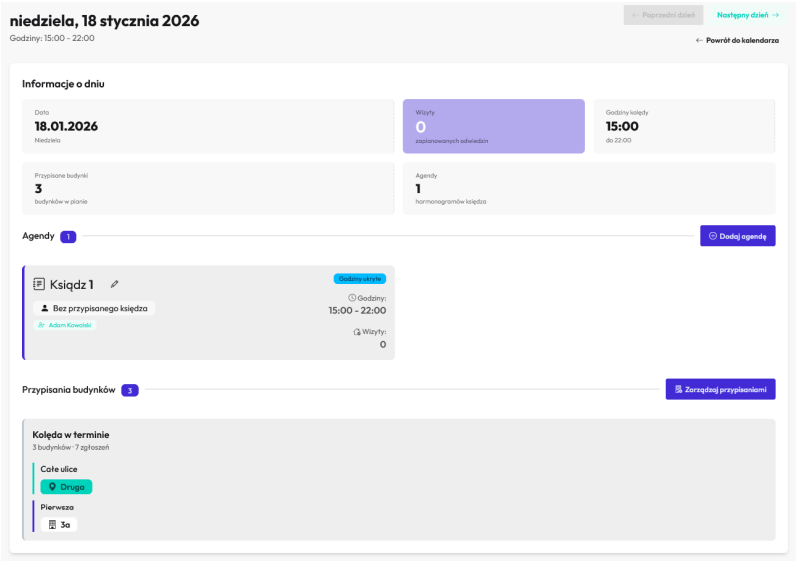
Podczas dodawania lub edycji dni kolędowych można ustawić daty rozpoczęcia i zakończenia kolędy, które potem będą umożliwiały dodanie wybranych dni z ich zakresu (często będzie to wiele dni, są to jednak daty graniczne). Dla każdego dnia ponadto można ustawić domyślne godziny rozpoczęcia i zakończenia wizyt, które potem mogą być nadpisywane na poziomie poszczególnych agend.

W widoku wyświetlania dni kolędowych można przejść do widoku danego dnia przez kliknięcie na niego. Spowoduje to wyświetlenie szczegółów dnia kolędowego (rys. 5.12), w którym dostępna jest statystyka dnia, lista wszystkich agend oraz lista wszystkich przypisań.

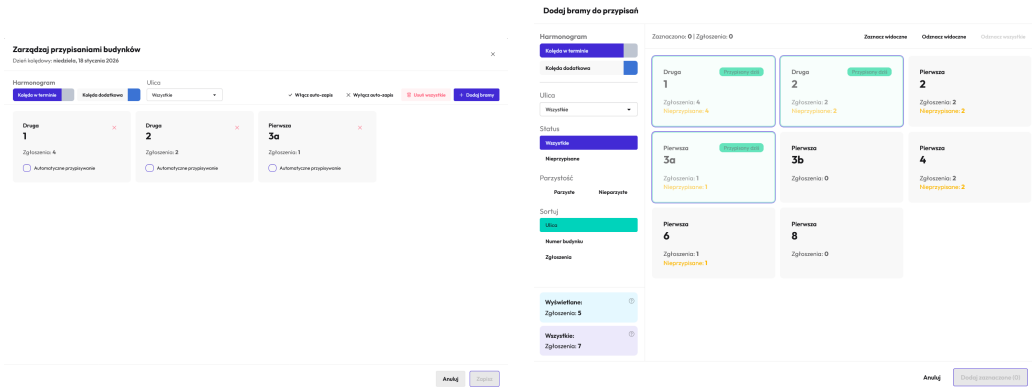
Zarządzanie przypisaniami

Sekcja przypisań prezentuje listę wszystkich przypisanych bram dla danego dnia. Jeśli wszystkie bramy na danej ulicy są przypisane, to zamiast wyświetlać je wszystkie system pokazuje nazwę ulicy w sekcji przypisań całych ulic.

Tworzenie, edycja i usuwanie przypisań jest możliwe po kliknięciu przycisku *Zarządzaj przypisaniami* w sekcji przypisań. Spowoduje to wyświetlenie okienka zarządzania przypisań (rys. 5.13a). Dostępne są w nim opcje filtrowania istniejących przypisań, usuwania ich, a także włączania i wyłączania autozapisu dla poszczególnych przypisań. Aby dodać nowe przypisanie, należy kliknąć przycisk *Dodaj bramy*, co spowoduje wyświetlenie okienka dodawania przypisań (rys. 5.13b). W nim po lewej stronie dostępna jest lista filtrów umożliwiających zawężenie listy dostępnych bram oraz podsumowanie aktualnie wybranych bram. Po prawej stronie znajduje się



Rysunek 5.12: Szczegóły dnia kolędowego



(a) Lista przypisań (b) Dodawanie przypisań

Rysunek 5.13: Zarządzanie przypisaniami

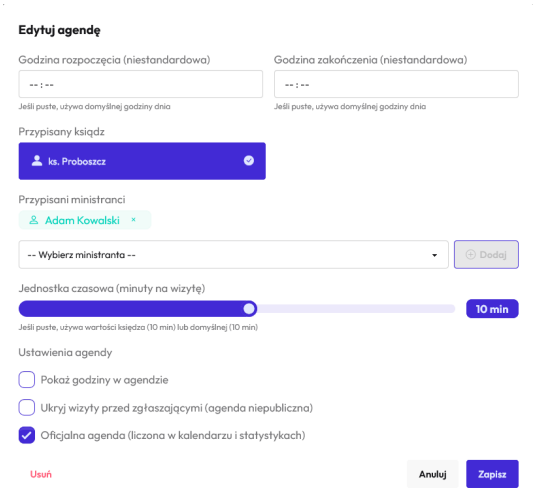
lista bram, spełniających kryteria wyszukiwania. Bramy, które są już przypisane w danym dniu, są oznaczone i nie można ich odznaczyć.

Przypisanie danej bramy jest rozpatrywane w zasięgu jednego harmonogramu, tzn. brama może być przypisana do tego samego dnia wiele razy w różnych harmonogramach, ale tylko jeden raz w ramach jednego harmonogramu. Dodatkowo w ramach jednego harmonogramu może być włączony autozapis, a w ramach drugiego już nie.

Zarządzanie agendami

W sekcji agend dostępna jest lista wszystkich agend danego dnia oraz możliwość ich tworzenia i edycji. Kliknięcie przycisku *Nowa agenda* spowoduje wyświetlenie formularza tworzenia agendy (rys. 5.14), który jest identyczny jak formularz edycji

agendy (dostępny po kliknięciu na ikonkę ołówka w kafelku agendy).



Rysunek 5.14: Tworzenie i edycja agendy

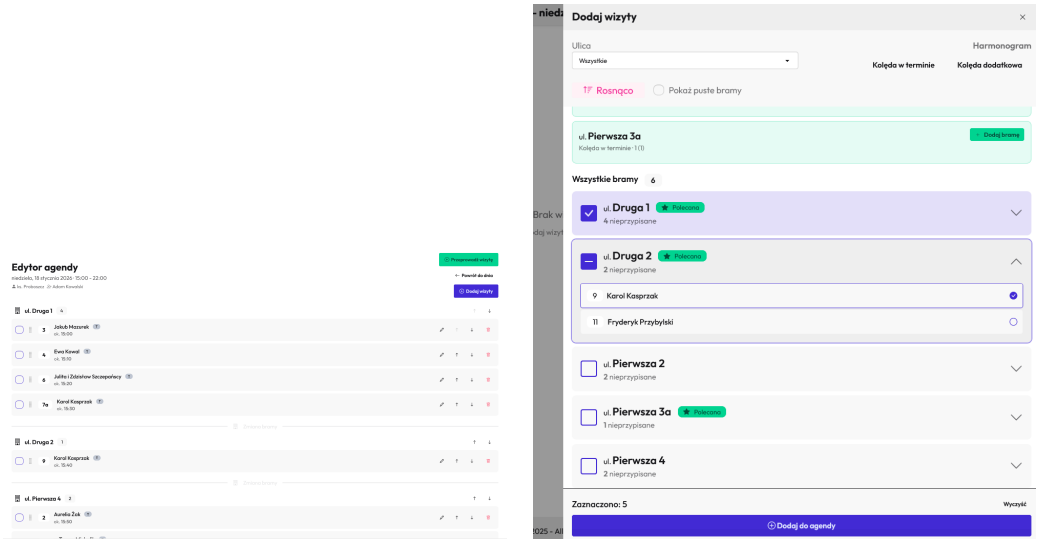
Każda agenda ma kilka istotnych opcji konfiguracyjnych:

- Godzina rozpoczęcia/zakończenia — nadpisanie opcji z dnia,
- Przypisany ksiądz — ksiądz, który będzie realizował listę wizyt,
- Przypisani ministranci — ministranci, którzy będą wspomagać księdza podczas wizyt,
- Jednostka czasowa — czas trwania pojedynczej wizyty w minutach (wliczając przejścia między mieszkaniami),
- Pokaż godziny — czy udostępniać zgłaszającym w panelu zgłoszenia przewidywany czas wizyty,
- Ukryj wizyty — czy ukryć agendę przed zgłaszającymi (wyświetlać zapisanym na nią informację o oczekiwaniu na zapis),
- Oficjalna agenda — czy agenda jest liczona do statystyk i pokazywana w kalendarzu (jej wizyty nadal są liczone; opcja przydatna do wizyt indywidualnych).

Planowanie wizyt

Po kliknięciu na kafelek agendy (poza przyciskiem edycji) zostanie wyświetlony edytor agendy (rys. 5.15a), w którym dostępna jest lista wszystkich wizyt zaplanowanych w danej agendzie oraz możliwość zarządzania nimi.

Aby dodać wizyty, należy kliknąć przycisk *Dodaj wizyty*, co spowoduje rozwinięcie od boku okienka z listą wszystkich niezaplanowanych zgłoszeń (rys. 5.15b). W nim można skorzystać z filtrów po lewej stronie, aby zawęzić (lub rozszerzyć) listę



(a) Edytor agendy (b) Dodawanie wizyt do agendy

Rysunek 5.15: Zarządzanie agendami

zgłoszeń, a następnie zaznaczyć te, które chce się dodać do agendy. Po kliknięciu przycisku *Dodaj do agendy* zostaną one dodane do agendy jako wizyty. Domyślnie zostaną umieszczone na końcu listy wizyt, ale jeśli system stwierdzi, że istnieje lepsze miejsce (np. występuje już dana brama), to wstawi je tam.

W edytorze można zmieniać kolejność wizyt za pomocą:

- przeciągania i upuszczania (*drag & drop*),
- przycisków przesuwania wizyty w górę lub w dół listy,
- przycisków przesuwania bramy w górę lub w dół listy,
- zaznaczania wielu sąsiadujących wizyt i przesuwania ich grupowo za pomocą dolnego menu.

Dla każdej wizyty dostępny jest także przycisk wyświetlania szczegółów zgłoszenia, otwierający okno szczegółów i edycji zgłoszenia (jak w sekcji *Edycja zgłoszeń*).

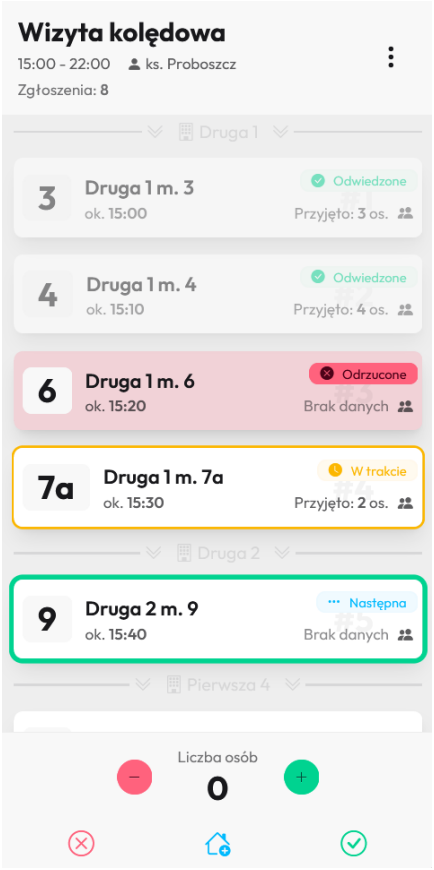
Przeprowadzanie wizyty

W edytorze agendy dostępny jest przycisk *Przeprowadź wizytę*, który przenosi do trybu przeprowadzania wizyty (rys. 5.16). Do tego widoku mają dostęp także przypisani do danej agendy ministranci poprzez kafelek z listą nadchodzących agend na stronie głównej.

W trybie przeprowadzania wizyty dostępna jest lista wszystkich wizyt w agendzie, dostosowana do widoku mobilnego (większe przyciski, układ pionowy) i umożliwiająca szybkie poruszanie się po liście wizyt i oznaczanie ich statusów. Przeprowadzanie wizyty można rozpocząć na kwadrans przed ustawionym w agendzie (bądź w dniu) czasie rozpoczęcia wizyty. Dopiero wtedy wyświetlany jest przycisk *Rozpocznij wizytę*. Po jego kliknięciu należy wybrać księdza (o ile nie został on wcześniej wybrany w agendzie), a następnie oznaczone zostaje pierwsze mieszkanie.

Kafelek każdej wizyty ma różne oznaczenia kolorystyczne w zależności od jej statusu:

- zwykły — wizyta zaplanowana,
- zielone obramowanie — następna wizyta,
- żółte obramowanie — trwająca wizyta (ksiądz jest w mieszkaniu),
- wyszarzony kafelek — zgłoszenie odwiedzone (wizyta zakończona),
- czerwone tło — wizyta nieodbyta (odrzucona),
- żółte tło — wizyta wstrzymana (tymczasowy status).



Rysunek 5.16: Przeprowadzanie wizyty

Status wizyty wyświetlany jest także zgłaszającemu w jego panelu zgłoszenia. Dodatkowo po rozpoczęciu wizyty przewidywane godziny są aktualizowane na bieżąco i wyświetlane zgłaszającemu (jeśli w agendzie jest włączona opcja pokazywania godzin).

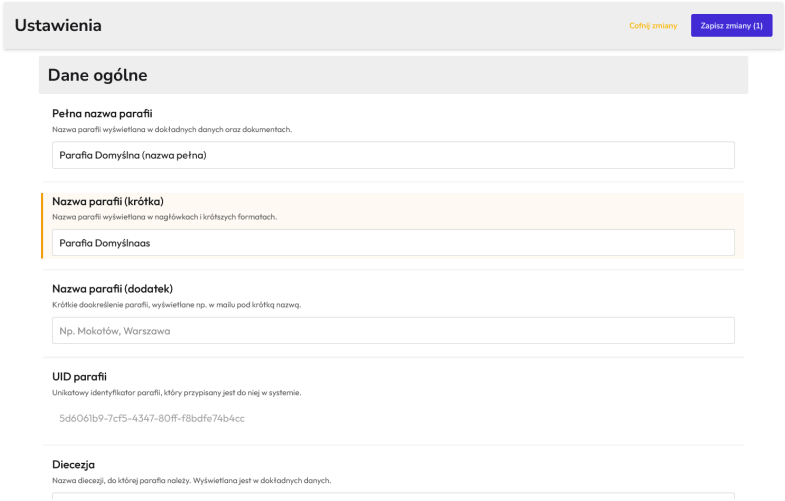
Dolny pasek zawiera przyciski do oznaczania liczby domowników w danym mieszkaniu oraz przyciski do kontroli całości — od lewej: oznaczenie wizyty jako odrzuconej, dodanie mieszkania (gdy ktoś niezaplanowany poprosi o kolędę) oraz oznaczenie wizyty jako zakończonej (odwiedzonej). Wszystkie te przyciski odnoszą się do wizyty, oznaczonej zieloną ramką (następnej). Po oznaczeniu wizyty jako zakończonej lub odrzuconej, automatycznie zaznaczana jest kolejna wizyta.

Przycisk do dodawania mieszkania otwiera okienko dodawania wizyty do agendy, w którym należy podać numer mieszkania (można dodawać tylko mieszkania z aktualnie odwiedzanej bramy). Po dodaniu mieszkania zostaje ono od razu oznaczone jako następne.

Więcej specjalistycznych opcji dostępnych jest w menu rozwijanym w prawym górnym rogu ekranu (ikona trzech pionowych kropek).

Ustawienia aplikacji

Aby przejść do ustawień aplikacji, należy w bocznym menu wybrać opcję *Ustawienia* (oznaczoną przez ikonę koła zębatego). Wyświetli się wtedy strona z kilkoma zakładkami ustawień. Należy wybrać zakładkę *Ustawienia ogólne*, aby wyświetlić listę ustawień ogólnych aplikacji dla tej parafii (rys. 5.17).



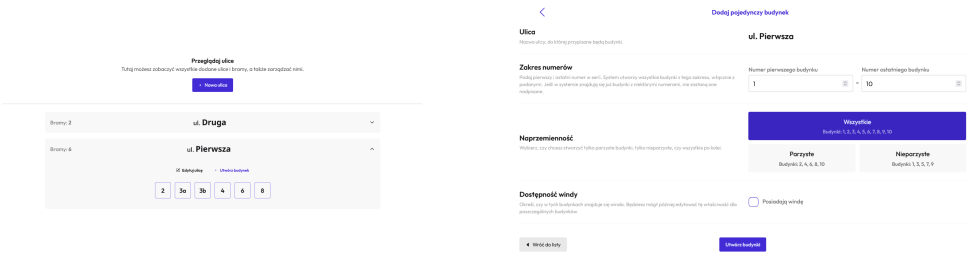
Rysunek 5.17: Ustawienia ogólne aplikacji

W tym miejscu można dowolnie je wszystkie zmieniać, a następnie zapisać zmiany przyciskiem *Zapisz zmiany* w przyklejonym nagłówku bądź je odrzucić przyciskiem *Cofnij zmiany* obok.

Zarządzanie adresami

Po przejściu do ustawień ogólnych i wejściu w zakładkę *Adresy* dostępna jest lista wszystkich wprowadzonych do systemu bram i ulic oraz możliwość ich tworzenia, edycji i usuwania (rys. 5.18a). Formularze zgłoszeniowe (publiczny i wewnętrzny) pozwalają na wybór adresu jedynie spośród wprowadzonych do systemu adresów, zatem istotne jest, aby przed rozpoczęciem przyjmowania zgłoszeń utworzyć wszystkie potrzebne bramy w parafii.

Z racji, że może to być dość czasochłonne zadanie, aplikacja umożliwia tworzenie serii bram przy danej ulicy na podstawie podanego zakresu numerów (rys. 5.18b). Na przykład, aby utworzyć bramy dla ulicy *Kwiatowej* z numerami od 1 do 20, należy najpierw utworzyć ulicę *Kwiatową*, uzupełniając formularz tworzenia ulicy dostępny po kliknięciu przycisku *Nowa ulica*, a następnie rozwinąć kafelek tej ulicy



(a) Lista adresów (b) Edycja bramy

Rysunek 5.18: Zarządzanie adresami

na liście ulic i kliknąć przycisk *Utwórz budynek*. Spowoduje to wyświetlenie formularza tworzenia budynku, z którego można przejść do tworzenia bram dla podanego zakresu numerów, klikając przycisk *Dodaj wiele budynków na raz*.

Przy tworzeniu i edycji bram dostępne są także opcje zaawansowane, przede wszystkim określenie dostępności windy. W obecnej wersji aplikacji opcja ta determinuje kolejność automatycznego układania mieszkań w danej bramie podczas planowania wizyt (w bramach z windą są porządkowane malejąco).

Rozdział 6.

Dla programistów

6.1. Instrukcja instalacji

6.1.1. Środowisko deweloperskie

Wymagania

Do uruchomienia aplikacji w środowisku deweloperskim potrzebne jest zainstalowanie następujących narzędzi:

- `docker` w wersji co najmniej 28.2.2,
- `docker compose` w wersji co najmniej 2.37.1,
- `npm` w wersji co najmniej 11.6.0 (*opcjonalnie*)

Instalacja i uruchamianie

Aby uruchomić aplikację w środowisku deweloperskim, należy wykonać następujące kroki:

1. Sklonować repozytorium z kodem źródłowym aplikacji:

   ```
   git clone https://github.com/michalchawar/SOK.NET.git
   ```

2. Przejść do katalogu z kodem źródłowym:

   ```
   cd SOK.NET
   ```

3. Stworzyć plik ze zmiennymi środowiskowymi na podstawie dostarczonego szablonu:

```
cp .env.sample .env
```

4. Wypełnić plik `.env` odpowiednimi wartościami lub pozostawić domyślne dla środowiska deweloperskiego,
5. Zbudować i uruchomić aplikację za pomocą narzędzia *docker-compose*:

   ```
   docker compose up --build
   ```

6. Przejść pod adres `http://localhost:8060`, gdzie dostępna będzie aplikacja.

Rozwój

Aby wprowadzać zmiany w kodzie źródłowym aplikacji, można użyć dowolnego edytora kodu, należy jednak pamiętać o każdorazowym przebudowaniu obrazu Dockera po wprowadzeniu zmian. Można to zrobić za pomocą polecenia:

```
docker compose up --build
```

W celu przyspieszenia tego procesu w repozytorium dostępny jest dodatkowy plik konfiguracyjny `docker-compose.vs-code.yml`, który umożliwia automatyczne wykrywanie zmian w kodzie źródłowym przy pomocy polecenia `dotnet watch`, do użytku w edytorach takich jak Visual Studio Code, które nie obsługują technologii *Hot Reload* w kontenerach Docker. Aby uruchomić aplikację w tej konfiguracji, należy użyć polecenia:

```
docker compose -f docker-compose.yml
               -f docker-compose.vs-code.yml up --build
```

Wówczas zmiany w kodzie będą automatycznie wykrywane i aplikacja będzie odświeżana bez konieczności ponownego budowania obrazu Dockera. Aby ponadto umożliwić automatyczne przebudowanie arkusza stylów CSS przy zmianach w plikach źródłowych, można uruchomić dodatkowo polecenie:

```
cd SOK.Web
npm run watch:css
```

Uruchamia to usługę Tailwind CSS w trybie obserwacji zmian w plikach źródłowych i automatycznie przebudowuje arkusz stylów CSS przy każdej zmianie. Należy również pamiętać, że za pierwszym razem trzeba uprzednio zainstalować zależności projektu przy pomocy polecenia:

```
npm install
```

Wszystkie powyższe kroki opisane są również w pliku `README.md` w repozytorium aplikacji.

Środowisko produkcyjne

Aby uruchomić aplikację w środowisku produkcyjnym, należy wykonać podobne kroki jak w przypadku środowiska deweloperskiego, z tą różnicą, że należy użyć pliku konfiguracyjnego `docker-compose.prod.yml` zamiast `docker-compose.yml`. Polecenie uruchamiające aplikację w trybie produkcyjnym wygląda następująco:

```
docker compose -f docker-compose.prod.yml up --build
```

Należy również pamiętać o odpowiednim skonfigurowaniu zmiennych środowiskowych w pliku `.env` pod kątem środowiska produkcyjnego.

6.2. Testy jednostkowe

Tak jak wspomniano w rozdziale **??**, aplikacja posiada testy jednostkowe pokrywające kluczowe funkcjonalności logiki biznesowej. Testy te znajdują się w katalogu `SOK.Tests` w repozytorium i można je uruchomić za pomocą narzędzia `dotnet test`. Aby to zrobić, należy przejść do katalogu z kodem źródłowym aplikacji i wykonać polecenie:

```
dotnet test SOK.Tests
```

Uruchomi ono wszystkie testy jednostkowe i wyświetli wyniki w konsoli. Testy te obejmują wybrane kluczowe scenariusze użycia aplikacji, weryfikując poprawność zarówno najważniejszych encji modelu dziedzinowego w warstwie domeny, jak i najistotniejszych serwisów w warstwie aplikacji.

6.3. Statystyki kodu

Tabela 6.1 przedstawia statystyki linii kodu projektu wygenerowane za pomocą narzędzia `cloc`. Zademonstrowano podział na języki programowania oraz typy plików, z wyszczególnieniem liczby plików, pustych wierszy, komentarzy oraz właściwego kodu źródłowego.

Analizę przeprowadzono na kodzie źródłowym aplikacji bezpośrednio po sklonowaniu repozytorium, bez wprowadzania jakichkolwiek zmian. Wykluczono przy tym pliki bibliotek zewnętrznych oraz pliki binarne i tymczasowe, generowane przez środowisko uruchomieniowe.

Język	**Pliki**	**Puste wiersze**	**Komentarze**	**Kod**
C#	305	11 625	4 995	41 126
Razor	58	881	426	10 086
JSON	9	0	0	4 275
HTML	8	205	133	1 140
JavaScript	4	41	31	231
YAML	3	8	2	147
CSS	2	34	8	125
Visual Studio Solution	1	1	1	106
MSBuild script	5	20	0	98
Markdown	4	34	0	81
Dockerfile	1	13	11	32
XML	1	0	0	19
SUMA	**401**	**12 862**	**5 607**	**57 466**

Tabela 6.1: Statystyki projektu (na podstawie narzędzia `cloc`)

Rozdział 7.

Podsumowanie

7.1. Osiągnięcia i wnioski

W niniejszej pracy przedstawiono kompleksowe podejście do analizy i implementacji systemu zarządzania danymi. Główne osiągnięcia obejmują:

- szczegółową analizę wymagań i wdrożenie systemu zarządzania danymi, uwzględniającą aspekty skalowalności, bezpieczeństwa i wydajności,
- projekt architektury systemu, który umożliwia efektywne przechowywanie i przetwarzanie danych,
- implementację całej aplikacji z wykorzystaniem nowoczesnych technologii,
- zastosowanie najlepszych praktyk programistycznych i wzorców projektowych,
- opakowanie całości w prosty do utrzymania i rozwijania system.

Praca nad tym projektem pozwoliła na zdobycie cennego doświadczenia w zakresie projektowania i implementacji systemów informatycznych, a także na zrozumienie kluczowych aspektów zarządzania danymi w kontekście współczesnych wyzwań technologicznych.

Pełna integracja wszystkich komponentów systemu wymagała dużego nakładu pracy i zaangażowania, a także specjalistycznej wiedzy z różnych dziedzin informatyki. Głównym wyzwaniem było zapewnienie, że system będzie nie tylko funkcjonalny, ale także przystępny zarówno dla użytkowników końcowych, o często przeciętnych zdolnościach technicznych, jak i dla administratorów odpowiedzialnych za jego utrzymanie. Przy wypełnianiu tego celu kluczowe okazało się poznanie wymienionych w rozdziale 3. technologii i narzędzi oraz ich efektywne zastosowanie w praktyce.

Aplikacja została wdrożona i przetestowana w rzeczywistych warunkach, co pozwoliło na zweryfikowanie jej funkcjonalności i wydajności. Uzyskane wyniki potwierdziły skuteczność zastosowanych rozwiązań i wskazały kierunki dalszego rozwoju systemu. Należy przy tym pamiętać, że w kontekście obecnej luki na rynku oprogramowania przeznaczonego do organizacji wizyt duszpasterskich, nawet najprostsze rozwiązania mogą przynieść znaczące korzyści użytkownikom końcowym. Szczególnie więc pierwsze w pełni zintegrowane, wszechstronne i ogólnodostępne narzędzie może okazać się przełomowe w tym obszarze, wyznaczając nowe standardy i otwierając drogę do dalszych innowacji.

7.2. Dalsza praca

7.2.1. Rozwój interfejsu

Choć aplikacja spełnia wszystkie założenia funkcjonalne, w samym modelu obiektowym pozostawiony został potencjał do dalszej rozbudowy i optymalizacji. W przyszłości planowane jest wprowadzenie kolejnycch funkcji, takich jak wersjonowanie niektórych danych, integracja z innymi systemami zarządzania danymi, czy też implementacja bardziej rozbudowanych opcji analizy i statystyk.

Uprości to jeszcze bardziej organizację kolędy oraz pozwoli na lepsze dostosowanie systemu do indywidualnych potrzeb użytkowników. Aplikacja zyska przez to kolejną realną przewagę nad metodami bardziej manualnymi.

7.2.2. Bezpieczeństwo danych

W trakcie implementacji aplikacji stworzono podstawowe mechanizmy zabezpieczające dane użytkowników, takie jak uwierzytelnianie i autoryzacja, a także szyfrowanie niektórych danych w bazie. W przyszłości należy jednak rozważyć wdrożenie bardziej zaawansowanych mechanizmów bezpieczeństwa, takich jak monitorowanie dostępu do danych, audyt zmian czy też implementacja mechanizmów, umożliwiających rotację kluczy szyfrowania.

7.2.3. Wzbogacenie planowania

W przyszłych wersjach aplikacji planowane jest wprowadzenie bardziej zaawansowanych funkcji planowania wizyt duszpasterskich. Jedną z nich będzie możliwość interaktywnego planowania tras wizyt na mapie obszaru, co pozwoli na optymalizację czasu i zasobów potrzebnych do realizacji wizyt. Dodatkowo, na tej podstawie możliwe będzie również zaproponowanie algorytmów automatycznego generowania planów wizyt, uwzględniających różne kryteria, takie jak preferencje duszpasterzy

czy specyficzne potrzeby parafian, jednocześnie optymalizując trasę pod kątem odległości i czasu podróży.

7.2.4. Kartoteki osobowe

Potencjalnym kierunkiem rozwoju aplikacji jest wzbogacenie jej o funkcjonalności związane z zarządzaniem kartotekami osobowymi parafian. Umożliwiłoby to przechowywanie szczegółowych informacji o mieszkańcach parafii, takich jak dane kontaktowe, historia wizyt duszpasterskich, preferencje czy specjalne potrzeby, a w dalszym etapie również pełnych danych, potrzebnych do prowadzenia dokumentacji sakramentalnej.

Jest to szczególnie obiecujący kierunek rozwoju, ponieważ okres wizyt kolędowych w praktyce duszpasterskiej często wiąże się z aktualizacjami i uzupełnianiem danych osobowych parafian, a współcześnie wiele parafii nie posiada spójnego i mobilnego systemu do zarządzania tymi informacjami, a duszpasterze często muszą polegać na papierowych kartotekach lub rozproszonych notatkach. Integracja takiej funkcjonalności z istniejącym systemem zarządzania wizytami duspasterskimi mogłaby znacząco usprawnić pracę duszpasterzy, umożliwiając im łatwy dostęp do aktualnych informacji o parafianach podczas wizyt, co z kolei przyczyniłoby się do bardziej efektywnego i spersonalizowanego podejścia do duszpasterstwa.

7.2.5. Aplikacja mobilna

Kolejnym, najbardziej obecnie odległym krokiem w rozwoju aplikacji mogłoby być stworzenie dedykowanej aplikacji mobilnej, która umożliwiłaby duszpasterzom i ministrantom dostęp do systemu zarządzania wizytami duszpasterskimi bezpośrednio z ich smartfonów lub tabletów. Taka aplikacja mogłaby oferować funkcje podobne do tych dostępnych w wersji webowej, ale zoptymalizowane pod kątem urządzeń mobilnych, co pozwoliłoby na jeszcze większą wygodę i elastyczność w zarządzaniu wizytami duszpasterskimi w terenie.

Podczas gdy aplikacja webowa, dostępna przez przeglądarkę internetową, oferuje w zupełności wystarczający zakres funkcjonalności i jest łatwo dostępna z różnych urządzeń, aplikacja mobilna mogłaby dodać dodatkową warstwę użytkowości, zwłaszcza w kontekście pracy duszpasterzy w terenie. Funkcje takie jak powiadomienia push, możliwość pracy offline czy integracja z funkcjami urządzenia mobilnego (np. GPS, aparat fotograficzny) mogłyby znacząco ulepszyć doświadczenie użytkowników i zwiększyć efektywność zarządzania wizytami duszpasterskimi, szczególnie w kontekście poprzedniego punktu, dotyczącego przechowywania kartotek osobowych parafian. Z racji większej kontroli nad urządzeniem i jego zasobami, aplikacja mobilna przede wszystkim oferowałaby lepsze bezpieczeństwo danych, co jest kluczowe w kontekście przechowywania wrażliwych informacji osobowych.

Bibliografia

[1] Microsoft. ASP.NET Core documentation. https://docs.microsoft.com/aspnet/core/. dostęp: 28.01.2026.

[2] Microsoft. .NET 8 documentation. https://docs.microsoft.com/dotnet/, 2023. dostęp: 28.01.2026.

[3] Microsoft. SQL Server 2022 documentation. https://docs.microsoft.com/sql/sql-server/, 2022. dostęp: 28.01.2026.

[4] Microsoft. Razor syntax reference for ASP.NET Core. https://docs.microsoft.com/aspnet/core/mvc/views/razor. dostęp: 28.01.2026.

[5] Tailwind Labs. Tailwind CSS documentation. https://tailwindcss.com/docs. dostęp: 28.01.2026.

[6] Evan You. Vue.js documentation. https://vuejs.org/guide/introduction.html. dostęp: 28.01.2026.

[7] Docker Inc. Docker documentation. https://docs.docker.com/. dostęp: 28.01.2026.

[8] Mozilla Developer Network. Fetch API. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API. dostęp: 28.01.2026.

[9] jQuery Foundation. jQuery API documentation. https://api.jquery.com/. dostęp: 28.01.2026.

[10] DaisyUI. DaisyUI documentation. https://daisyui.com/docs/. dostęp: 28.01.2026.

[11] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[12] Microsoft. Entity Framework Core documentation. https://docs.microsoft.com/ef/core/. dostęp: 28.01.2026.

[13] Microsoft. Introduction to Identity on ASP.NET Core. https://docs.microsoft.com/aspnet/core/security/authentication/identity. dostęp: 28.01.2026.

[14] Jeffrey Stedfast. MailKit documentation. https://github.com/jstedfast/MailKit. dostęp: 28.01.2026.

[15] QuestPDF. QuestPDF documentation. https://www.questpdf.com/. dostęp: 28.01.2026.

[16] QuestPDF. QuestPDF Community MIT License. https://www.questpdf.com/license/. dostęp: 28.01.2026.

[17] Traefik Labs. Traefik Proxy documentation. https://doc.traefik.io/traefik/. dostęp: 28.01.2026.

[18] Internet Security Research Group. Let's Encrypt. https://letsencrypt.org/. dostęp: 28.01.2026.

[19] Docker Inc. Docker Compose documentation. https://docs.docker.com/compose/. dostęp: 28.01.2026.

[20] xUnit.net. xUnit.net documentation. https://xunit.net/. dostęp: 28.01.2026.

[21] Moq. Moq: The most popular and friendly mocking framework for .NET. https://github.com/devlooped/moq. dostęp: 28.01.2026.

[22] Roy Osherove. *The Art of Unit Testing: With Examples in C#*. Manning Publications, 2013.

[23] Parafia pw. św. Michała Archanioła w Ujanowicach. Adventus. https://koleda.parafiaujanowice.pl/, 2025. dostęp: 15.01.2026.

[24] Ania Lewandowska and Łukasz Kadziewicz. Pytanie na śniadanie, część 4. https://vod.tvp.pl/programy,88/pytanie-na-sniadanie-odcinki,2542729/odcinek-154,S01E154,2622884, 2026. Sezon 1, odcinek 154; dostęp: 15.01.2026.