UNIWERSYTET WSB MERITO POZNAŃ

Wydział Finansów i Bankowości

PipeQ - elastyczne oprogramowanie usprawniające obsługę studentów na uniwersytecie WSB Merito

PROJEKT DYPLOMOWY

Poznań 2025

DANE PARTNERÓW

A1. Dane Promotora

Imię i nazwisko	**Mariusz Nogala**
Stopień / Tytuł naukowy	Doktor
Data i podpis	

A2. Dane członków Zespołu projektu

Imię i nazwisko	**Krzysztof Pacyna**
Kierunek studiów	Informatyka
Tryb studiów	Niestacjonarny
Data i podpis	

Imię i nazwisko	**Mateusz Popielarz**
Kierunek studiów	Informatyka
Tryb studiów	Niestacjonarny
Podpis	

Imię i nazwisko	**Mateusz Rój**
Kierunek studiów	Informatyka
Tryb studiów	Niestacjonarny
Data i podpis	

Przygotowane przez nas oprogramowanie jest dostępne na stronie GitHub pod następującym linkiem:
https://github.com/Kennene/pipeq

UNIWERSYTET WSB MERITO POZNAŃ

ZAŁOŻENIA PROJEKTU

B1. Opis projektu

1. Uzasadnienie wyboru tematu

Świat w zawrotnym tempie zmierza w stronę coraz większej cyfryzacji. Nasze życie jest przepełnione technologią, która wspiera nas w codziennych czynnościach. Istnieją jednak działania na tyle intuicyjne, że potrzeba ich cyfryzacji wydaje się znikoma. Mimo to, niektóre systemy – choć z pozoru proste – bywają projektowane bez uwzględnienia realnych potrzeb użytkowników, a jedynie z myślą o samej idei cyfryzacji. Takie podejście prowadzi do frustracji i ostatecznie sprawia, że użytkownicy omijają niepraktyczne rozwiązania na rzecz tradycyjnych interakcji. To właśnie w tym kontekście zrodziła się idea stworzenia naszego programu - PipeQ.

Program PipeQ to błyskawicznie szybki i wszechstronny system kolejkowy do zarządzania przepływem ludzi w miejscach, gdzie wymagane jest zachowanie porządku wejścia. Projekt, który początkowo zrodził się z frustracji studentów związanej z trudnościami w korzystaniu z obecnego rozwiązania, szybko przekształcił się w realną koncepcję usprawnienia procesów administracyjnych uniwersytetu. Obecny system kolejkowania studentów ma wiele wad, które przyczyniają się do negatywnego odczucia z załatwiania spraw związanych z tokiem nauczania studentów. Stworzona przez nas aplikacja oferuje eleganckie i efektywne rozwiązanie tego problemu.

Podstawą wyboru tematu była zauważona przez nas możliwość wykorzystania nowoczesnych, efektywnych rozwiązań informatycznych. Tworząc aplikację działającą w czasie rzeczywistym, zdecydowaliśmy się na zastosowanie relatywnie nowego protokołu komunikacyjnego WebSocket. Dzięki naszej determinacji i chęci nauki nowych technologii opracowaliśmy system, który błyskawicznie informuje użytkownika o zmianach dotyczących jego zgłoszenia. Dzięki modularnej budowie naszego rozwiązania byliśmy w stanie przystosować nasz program do każdego środowiska.
Praktyczny charakter naszego programu, okazja do nauki nowych technologii oraz możliwość realnego wpływu na nasze własne środowisko, jakim jest uniwersytet, jednoznacznie zapieczętował nasz wybór.

2. Problem badawczy

2.1 Identyfikacja obecnego problemu

Obecny system kolejkowy dla studentów opiera się na narzędziach Microsoft, zintegrowanych za pomocą Power Automate. Choć rozwiązania te szczycą się uniwersalnością i szerokim zakresem zastosowań, ich ogólny charakter sprawia, że mogą okazać się niewystarczające do realizacji specyficznych i złożonych zadań, tak jak w tym przypadku. W kontekście omawianego zagadnienia, znacznie lepszym rozwiązaniem jest opracowanie dedykowanego narzędzia dostosowanego do konkretnych wymagań.

Analiza obecnego systemu pozwoliła nam jednoznacznie zidentyfikować kluczowe problemy i wyeliminować je już na etapie projektowania. Wiele z tych trudności wynika z ograniczeń narzuconych przez wykorzystane technologie, które ze względu na swoją zamkniętą architekturę nie oferują swobody w dostosowywaniu do indywidualnych potrzeb. Wynika to bezpośrednio z polityki Microsoftu, który – choć dostarcza potężne narzędzia – nie zapewnia pełnej elastyczności w zakresie ich modyfikacji.

2.2 Formułowanie problemu badawczego

Problem badawczy naszego projektu PipeQ koncentruje się na wyzwaniach związanych z organizacją obsługi studentów na Uniwersytecie WSB Merito w Poznaniu. Aplikacja internetowa w intuicyjny i wygodny sposób umożliwia rejestrację użytkownika do kolejki w celu załatwienia sprawy związanej ze swoim tokiem nauczania.

Naszym celem było zaprojektowanie oraz wdrożenie dynamicznego i przejrzystego systemu kolejkowego, który:

- Umożliwi błyskawiczną aktualizację zgłoszeń w czasie rzeczywistym
- Wyeliminuje możliwość rejestracji użytkowników poza godzinami pracy dziekanatu
- Zapewni możliwość rejestracji osobom bez konta uczelnianego, zachowując przy tym pełną poufność informacji
- Będzie bardzo prosty do obsługi zarówno dla studentów, jak i dla pracowników dziekanatu

3. Cel główny i cele szczegółowe projektu

3.1 Cel główny projektu

Naszym głównym celem było zaprojektowanie i wdrożenie przejrzystego i efektywnego zarządzania przepływem ludzi w Biurze Obsługi Studenta na Uniwersytecie WSB Merito w Poznaniu, który udrożni proces obsługi studentów. Nasz program PipeQ ma zminimalizować czas oczekiwania, wyeliminować frustrację z konieczności logowania oraz zapewnić zorganizowany i płynny rodzaj komunikacji między pracownikami dziekanatu a studentami.

3.2 Cele szczegółowe

Aby skutecznie wdrożyć nowy system, konieczne jest przejście przez kilka kluczowych etapów. Cele, które umożliwią płynne przeprowadzenie tego procesu to:

- Szczegółowa analiza obecnego systemu kolejkowania
- Zaprojektowanie nowej architektury systemu
- Przygotowanie infrastruktury
- Implementacja systemu
- Pełne wdrożenie i oddanie Programu do użytku

4. Zakres podmiotowy, przedmiotowy, czasowy i przestrzenny

4.1 Zakres podmiotowy

Nasz program PipeQ wyróżnia trzy główne grupy aktorów korzystających z naszego systemu.

I. Użytkownicy

   Główna grupa docelowa korzystająca z programu. Z racji, że nasz system będzie funkcjonował w prywatnej placówce edukacyjnej, naszymi użytkownikami systemu będą studenci. Ze względu na szeroką grupę użytkowników, nasz system musi być uniwersalny, z tłumaczeniem na języki obce, prostym interfejsem, skalowalnością do urządzeń mobilnych oraz dostępnością poprzez Wi-Fi, kod QR i linki.

II. Koordynatorzy

Osoby odpowiedzialne za zarządzanie kolejnością i obsługę użytkowników – w naszym przypadku pracownicy dziekanatu – zyskają narzędzie, które usprawni ich codzienną pracę. Dzięki unowocześnionemu interfejsowi koordynatora oraz holistycznemu widokowi przestrzeni roboczej proces zarządzania kolejką stanie się znacznie szybszy.

III. Administratorzy

Wybrani aktorzy systemu, posiadający najwyższe uprawnienia i odpowiedzialni za konfigurację oraz administrowanie systemem PipeQ. Pełnią rolę koordynatorów o rozszerzonych możliwościach, z dostępem do konsoli administratorskiej. Konsola ta umożliwia im regulowanie godzin otwarcia poszczególnych działów, dodawanie i awansowanie koordynatorów oraz monitorowanie logów z przepływu biletów.

4.2 Zakres przedmiotowy

Zakres przedmiotowy naszego oprogramowania obejmuje

- Zaprojektowanie i wdrożenie systemu kolejkowego do Biura Obsługi Studenta na Uniwersytecie WSB Merito w Poznaniu
- Zapewnienie szybkiej i płynnej aktualizacji statusów zgłoszeń
- Zapewnienie pełnej poufności i bezpieczeństwo danych przesyłanych drogą elektroniczną
- Skalowalność i możliwość rozbudowy systemu do wykorzystania w innych warunkach lub innych działach, gdzie kolejność obsługi jest istotna

4.3 Zakres czasowy

Zakres czasowy obejmuje wszystkie kluczowe etapy projektowania aplikacji – od dopracowania konceptu, aż po utrzymanie systemu.

W pierwszych etapach tworzenia projektu przeprowadziliśmy szczegółową analizę wadliwego, istniejącego systemu. Pozwoliło nam to uniknąć powielenia jego błędów na etapie tworzenia koncepcji naszego rozwiązania.

Projektowanie systemu rozpoczęliśmy od szerokiego spojrzenia na problem i określenia, które narzędzia informatyczne najlepiej sprawdzą się w naszym przypadku. Na tym etapie dobraliśmy najodpowiedniejsze technologie oraz opracowaliśmy ogólny rozkład bazy danych.

Następnie przeszliśmy do programowania szkieletu aplikacji i jej mechanizmu działania. Skupiliśmy się przede wszystkim na zapewnieniu pełnej funkcjonalności kluczowego założenia - nieprzerwanej i błyskawicznej komunikacji na linii koordynator -> serwer -> użytkownik. Następnie nadaliśmy naszemu szkieletowi atrakcyjną oprawę graficzną, płynne animacje oraz przygotowaliśmy tłumaczenia.

Ostatnim krokiem było wygenerowanie certyfikatów SSL Let's Encrypt, zabezpieczenie komunikacji na wszystkich kanałach (zarówno HTTPS, jak i WebSocket Secure) oraz wdrożenie aplikacji na serwer produkcyjny.

Proces tworzenia oprogramowania był czasochłonny, ponieważ zależało nam na uniknięciu sytuacji, w której niesprawny system utrudniłby dostęp do Biura Obsługi Studenta. Ze względu na strategiczne znaczenie projektu, prace rozpoczęliśmy 1 sierpnia 2024, a zakończyliśmy 20 stycznia 2025 wraz z wdrożeniem nowego systemu do użytku.

4.4 Zakres przestrzenny

System PipeQ został stworzony z myślą o wdrożeniu na Uniwersytecie WSB Merito w Poznaniu, jednak jego architektura umożliwia łatwą adaptację na innych uczelniach Merito w Polsce. Dzięki modularnej budowie system można bez trudu dostosować do różnych miejsc, w których kluczowe znaczenie ma zachowanie kolejności obsługi użytkowników. Przykładami takich miejsc są Dział Rekrutacji czy Biuro Karier i Praktyk.

Sam koncept sekwencjonowania przyjęć jest powszechnie stosowany w codziennym życiu i może zostać z powodzeniem zaadaptowany także w placówkach pozaoświatowych. Tworząc nasze oprogramowanie, czerpaliśmy inspirację z dobrze znanych systemów, którymi otaczamy się każdego dnia - od systemów obsługi klientów w restauracjach typu fast food po starannie przemyślany i zaawansowany system kolejkowy Regionalnego Centrum Krwiodawstwa i Krwiolecznictwa w Poznaniu.

5. Metody i techniki badawcze

5.1 Analiza porównawcza systemów podobnych

Przeprowadziliśmy analizę aktualnie używanych systemów kolejkowych, ze szczególnym uwzględnieniem systemów działających w innych szkołach, urzędach oraz placówkach medycznych. Celem było określenie najczęstszych problemów, które występują w tych systemach oraz ich wpływu na efektywność obsługi interesantów.

Na podstawie tej analizy zidentyfikowaliśmy główne wady istniejących rozwiązań. Wyniki tej analizy stały się podstawą do projektowania PipeQ jako systemu eliminującego najistotniejsze niedogodności.

5.2 Analiza heurystyczna interfejsu użytkownika

Do oceny interfejsu zastosowano zasady heurystyczne Jakobsa Nielsena[1], które pozwoliły nam na identyfikację i eliminację problemów związanych z użytecznością systemu. Między innymi sprawdziliśmy, czy interfejs jest intuicyjny i łatwy w obsłudze przez użytkownika. Testowano również, czy układ i kolory elementów systemu wspierają czytelność i ergonomię użytkowania.

5.3 Monitoring wydajności systemu

Podczas testów analizowaliśmy czasy odpowiedzi systemu, aby upewnić się, że działa on sprawnie i nie generuje opóźnień. Wykorzystaliśmy Google Lighthouse który posłużył nam do testowania wydajności aplikacji internetowej. Dzięki temu zoptymalizowano system pod kątem wydajności i szybkości działania.

B2. Zadania w projekcie

Cele szczegółowe projektu	Zadania w projekcie oraz termin rozpoczęcia i zakończenia realizacji zadania	Osoby zaangażowane w realizację zadania
Cel 1: Szczegółowa analiza obecnego systemu kolejkowania	Zadanie 1: Empiryczna diagnoza systemu kolejkowania	1. Mateusz Rój
	Zadanie 2: Konsultacja z pracownikami dziekanatu	1. Mateusz Rój 2. Krzysztof Pacyna
	Zadanie 3: Agregacja wniosków i analiza słabości	1. Mateusz Rój 2. Krzysztof Pacyna
Cel 2: Zaprojektowanie nowej architektury systemu	Zadanie 1: Wybór całościowego frameworka, wyspecjalizowanych narzędzi informatycznych i wybranie kanału komunikacji	1. Mateusz Popielarz 2. Krzysztof Pacyna
	Zadanie 2: Wstępne zaprojektowanies szkieletu bazy danych	1. Krzysztof Pacyna
	Zadanie 3: Podział ról i obowiązków przy tworzeniu nowego systemu	1. Mateusz Rój 2. Krzysztof Pacyna
Cel 3: Przygotowanie infrastruktury do programowania	Zadanie 1: Instalacja odpowiednich wtyczek i pakietów, przygotowanie Modeli z architektury MVC	1. Krzysztof Pacyna

	Zadanie 2: Podział projektu na regiony podmiotowe	1. Mateusz Rój 2. Mateusz Popielarz
	Zadanie 3: Konfiguracja serwera reverb, ustawienie kanałów WebSocket	1. Krzysztof Pacyna
Cel 4: Implementacja systemu	Zadanie 1: Przygotowanie kontrolerów do widoków	1. Mateusz Popielarz 2. Krzysztof Pacyna
	Zadanie 2: Wystawienie odpowiednich ednpointów do interakcji z systemem	1. Krzysztof Pacyna
	Zadanie 3: Wykonanie widoków we frameworku Vue.js	1. Mateusz Popielarz
	Zadanie 4: Dodanie możliwości powodów przyjścia na widoku użytkownika	1. Krzysztof Pacyna 2. Mateusz Popielarz 3. Mateusz Rój
	Zadanie 5: Wprowadzenie uwierzytelnienia do poszczególnych widoków i podział na role	1. Krzysztof Pacyna
	Zadanie 6: Intensywne testowanie aplikacji, identyfikacja i naprawa błędów	1. Mateusz Rój 2. Mateusz Popielarz

Cel 5: Pełne wdrożenie i oddanie do użytku	Zadanie 1: Przygotowanie środowiska na udostępnionym serwerze uniwersytetu	1. Krzysztof Pacyna 2. Mateusz Popielarz 3. Mateusz Rój
	Zadanie 2: Konfiguracja zmiennych środowiskych i zaszyfrowanie komunikacji certyfikatami Let's Encrypt	1. Krzysztof Pacyna
	Zadanie 3: Przedstawienie oprogramowania pracownikom dziekanatu i oddanie do użytku	1. Krzysztof Pacyna

REALIZACJA

C1. Opracowanie projektu

1. Założenia teoretyczne

Celem naszego projektu było stworzenie błyskawicznie szybkiego programu obsługującego studentów, którzy przyjdą do Biura Obsługi Studenta na Uniwersytecie WSB Merito w Poznaniu. Naszym kluczowym założeniem jest zapewnienie jak najmniej inwazyjnego, a zarazem jak najszybszego narzędzia pozwalającego na równoległą obsługę studentów.

1.1 Założenia teoretyczne użytkownika

Interfejs użytkownika naszego systemu został zaprojektowany z myślą o maksymalnej przejrzystości oraz szerokiej kompatybilności z urządzeniami mobilnymi. Dzięki czytelnemu układowi i dużym przyciskom, nasza aplikacja jest przyjazna nawet dla osób starszych oraz użytkowników z problemami ze wzrokiem.

Największy nacisk położyliśmy na ergonomię i przystępność interfejsu, aby korzystanie z systemu było płynne, naturalne i pozbawione zbędnych komplikacji. Naszym priorytetem jest zapewnienie użytkownikowi pełnej kontroli nad jego biletem oraz czytelnego, jednoznacznego komunikowania jakiejkolwiek zmiany dotyczącej jego zgłoszenia w systemie.

1.2 Założenia teoretyczne koordynatora

Podczas tworzenia naszego oprogramowania wzięliśmy pod uwagę najważniejsze czynniki istotne dla koordynatorów, czyli osób zajmujących się przetwarzaniem zgłoszeń użytkowników. Najważniejszym aspektem było zapewnienie przejrzystego podziału obowiązków, jednocześnie umożliwiając koordynatorom holistyczny wgląd w aktualną sytuację przerobową dziekanatu.

Aby osiągnąć ten cel, system powinien zapewniać koordynatorom podgląd na wszystkie stanowiska jednocześnie. Takie rozwiązanie pozwala na całościową ocenę sytuacji oraz szybkie reagowanie na zmiany w natężeniu ruchu. Przejrzysta wizualizacja stanowisk oraz statusu obsługi dostarcza wszystkich niezbędnych informacji, co pozwala dostosować dostępne zasoby do aktualnych wymagań.

Zadbaliśmy również o obsługę wyjątkowych sytuacji, takich jak błędny wybór miejsca docelowego przez użytkownika. W przypadku, gdy student omyłkowo skierował swoje zgłoszenie do niewłaściwego działu, system umożliwia jego sprawne przekierowanie bez konieczności ponownego zgłaszania. Dzięki temu utrzymujemy porządek zgłoszeń i zwiększamy płynność obsługi.

2. Opis sytuacji faktycznej

2.1 Ogólna charakterystyka obecnego systemu w Biurze Obsługi Studenta

Obecny system zarządzania kolejkami w dziekanacie Uniwersytetu WSB Merito w Poznaniu opiera się na wbudowanej infrastrukturze Microsoft. W teorii rozwiązanie to umożliwia studentom zgłoszenie sprawy online i oczekiwanie na jej obsłużenie przez pracownika administracyjnego. Jednak w praktyce system ten ma liczne ograniczenia, które negatywnie wpływają zarówno na organizację pracy dziekanatu jak i na satysfakcję studentów z obsługi.

2.2 Szczegółowa charakterystyka procesu

I. Użytkownik chcący zapisać się do kolejki w dziekanacie musi posiadać ze sobą telefon komórkowy za pomocą którego skanuje kod QR umieszczony na korytarzu. Rozwiązanie to eliminuje potrzebę drukowania jednorazowych biletów, wpływając pozytywnie na środowisko i minimalizację produkowanych śmieci, jednak osoby, które nie posiadają skanera kodów QR w swoim telefonie nie mają możliwości wejścia na stronę internetową.

II. Użytkownik jest przekierowywany do formularza Microsoft. Żeby go jednak zobaczyć musi zalogować się na swoje uczelniane konto Microsoft, do którego dostęp może uzyskać jedynie poprzez zalogowanie się do centralnego systemu uwierzytelniającego (CAS). Powoduje to frustrację, ponieważ wiele studentów nie pamięta swojego loginu i hasła, gdyż dane te mają zachowane na swoich komputerach osobistych w domu, a nie na komórce. Ponadto wyklucza to osoby, które nie mają kont w systemie CAS, na przykład zostali już skreśleni z listy studentów, lub też przyszła upoważniona osoba, która załatwia sprawy w imieniu studenta.

III. Po dotarcia do formularza, student wypełnia go, mając możliwość zmiany języka na angielski. Po wybraniu swojego celu (Dziekanat/Płatności), użytkownik może wprowadzić opcjonalnie swój e-mail i opis problemu. Interface użytkownika jest skonstruowany w taki sposób, jakoby były to pola obowiązkowe. Rejestracja zgłoszenia nie rozpocznie się bez zatwierdzenia formularza w całości.

IV. Po kliknięciu przycisku “Wyślij”, użytkownik otrzymuje statyczny komunikat o potwierdzeniu rejestracji zgłoszenia. Przez to, że komunikat jest statyczny to informacja o pozytywnym zarejestrowaniu pokaże się zawsze – nawet w przypadku błędu.

V. Silnik całego programu opiera się na narzędziu Power Automate, które jest częścią ekosystemu Microsoft i umożliwia automatyzację procesów oraz integrację wewnętrznych systemów. Po zatwierdzeniu formularza specjalny skrypt w Power Automate automatycznie dodaje wniosek studenta na Listę Microsoft.

VI. Z racji, że Lista Microsoftu odświeża się bardzo rzadko (raz na minutę) proces rejestracji może trwać relatywnie długo. Na tyle długo, że zniecierpliwieni użytkownicy, nie widząc efektów swojej rejestracji, ponownie wypełniają formularz otrzymania biletu. Skutkuje to pojawieniem się na liście biletów oczekujących dwóch wpisów, z czego jeden od razu już jest nieaktualny.

VII. Pracownik dziekanatu zmienia status zgłoszenia z "Oczekuje" na numer stanowiska. Przy założeniu pesymistycznym, użytkownicy muszą czekać pełną minutę zanim zobaczą aktualizację statusu na liście Microsoft. Jednak przez uprzednio zduplikowane bilety, najpierw wywoływany jest pierwszy, nieważny już bilet. Dopiero po zauważeniu przez pracownika dziekanatu braku niczyjej reakcji, wywoływane jest kolejne zgłoszenie z kolejki.

VIII. Użytkownik po zobaczeniu zmiany swojego statusu na Liście Microsoft wchodzi do dziekanatu na swoje odpowiednie miejsce.

Cały proces jest frustrujący – zanim student jeszcze zdąży przedstawić swój problem dotyczący toku studiów, jest już zniechęcony przez chaotyczny i niedopracowany system kolejkowy.

2.3 Przedstawienie głównych problemów i proponowane rozwiązanie

Problem	**Sugerowane rozwiązanie**
Konieczność skanera kodów QR w smartfonie	Obok kodu QR warto podać prosty link do ręcznego wpisania w przeglądarce.
Wymóg logowania się na konto uczelniane Microsoft	Możliwość anonimowej rejestracji użytkowników do Biura Obsługi Studenta

Niepotrzebne elementy interfejsu formularza	Dodatkowa informacja o powodzie wizyty może znacznie przyspieszyć pracę dziekanatu, ale powinna być opcjonalna i odbywać się na etapie oczekiwania – nie przed etapem rejestracji
Statyczny komunikat o potwierdzeniu zgłoszenia	Dynamiczny komunikat potwierdzenia zarejestrowania wraz z indywidualnym numerem do śledzenia postępu
Opóźnienie w aktualizacji listy Microsoft / Duplikowanie zgłoszeń wynikające z braku responsywności	Zastosowanie protokołu do bardzo szybkiej wymiany informacji, na przykład WebSocket
Nieprzejrzysty interfejs wyświetlający zgłoszenia z elementami interfejsu Microsoft List / Brak wyraźnej zmiany statusu biletu	Nowy, dedykowany widok specjalnie przystosowany do wyświetlania biletów z animacjami i opcją sygnału dźwiękowego

3. Badania własne / opis metod, technik i narzędzi badawczych / aparatura / oprogramowanie

3.1 Badania własne i analiza systemów kolejkowych

Przed przystąpieniem do realizacji projektu PipeQ, przeprowadziliśmy analizę systemów kolejkowych istniejących w uczelnianych dziekanatach, placówkach medycznych oraz instytucjach publicznych. To badanie umożliwiło nam zidentyfikowanie istotnych problemów związanych z obecnymi rozwiązaniami, takich jak brak dynamicznej aktualizacji zgłoszeń, co wiązało się z koniecznością ręcznego odświeżania stron przez użytkowników, a także niska przejrzystość systemu. Wyniki tych badań posłużyły jako podstawa do stworzenia dynamicznego i responsywnego systemu PipeQ, wykorzystującego WebSocket oraz intuicyjny interfejs użytkownika.

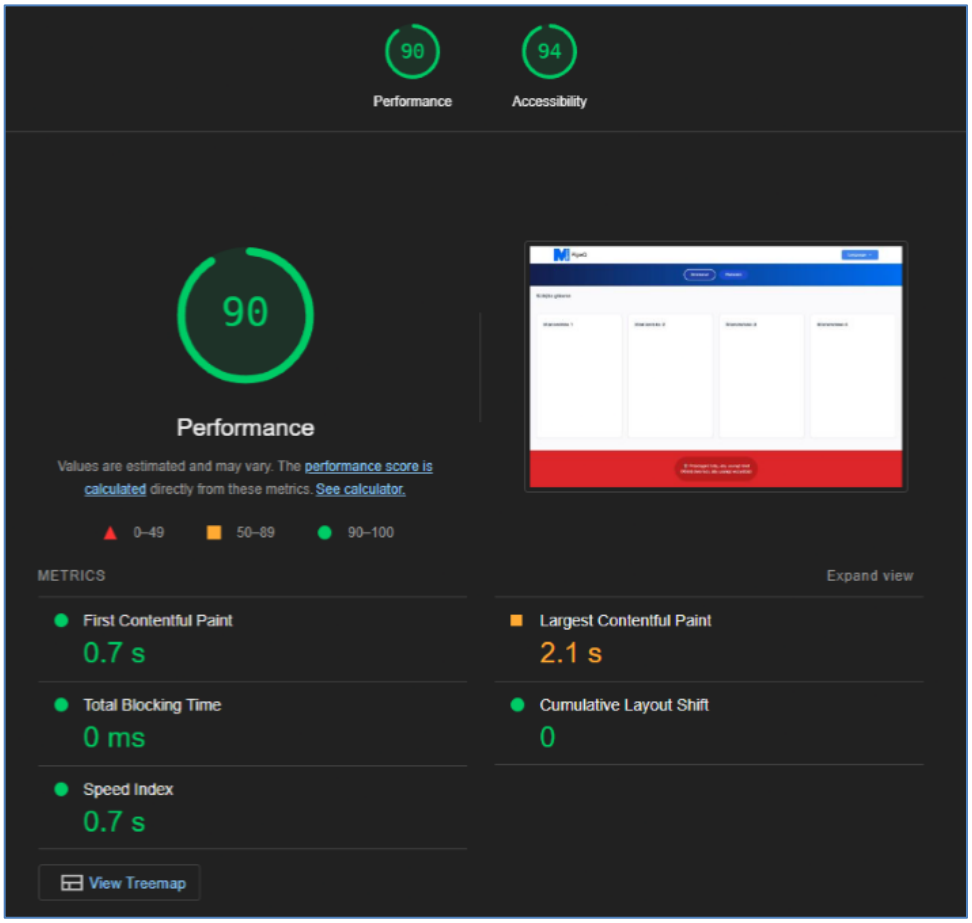
3.2 Metody badawcze i techniki projektowe

3.2.1 Analiza heurystyczna interfejsu użytkownika

Użyliśmy heurystycznych zasad Jakobsa Nielsena[1], by ocenić użyteczność systemu PipeQ. Testowanie obejmowało czytelność i intuicyjność interfejsu w czasie, w którym sprawdziliśmy, czy użytkownicy bez instrukcji potrafią korzystać z systemu. Responsywność naszego interfejsu, czy działa on na różnych urządzeniach (komputery tablety i smartfony). Oceniliśmy klarowność, spójność, estetykę oraz czytelność wyświetlanych informacji w naszym systemie.

3.2.2 Testy wydajnościowe

W celu sprawdzenia, jak system reaguje na obciążenie, przeprowadziliśmy symulacje ruchu użytkowników za pomocą Google Lighthouse co pozwoliło nam na analiza szybkości ładowania strony i optymalizacji kodu.



Rysunek 1. Wynik symulacji programu Google Lighthouse.

3.3 Testy integracyjne

System został testowany w kontekście integracji w celu sprawdzenia poprawności komunikacji między frontendem, backendem oraz bazą danych. Podczas testów sprawdzaliśmy między innymi, czy dane użytkowników są prawidłowo zapisywane i aktualizowane, czy komunikacja za pomocą WebSocket jest płynna bez widocznych opóźnień, a także czy pracownicy dziekanatu mogą efektywnie zarządzać zgłoszeniami.

3.4 Aparatura i środowisko testowe

Projekt był testowany w różnych środowiskach, aby zapewnić kompatybilność i wydajność. Do testów wykorzystaliśmy

- Komputery PC i laptopy z systemami Linux, Windows i macOS
- Smartfony i tablety (wbudowane przeglądarki Android, iOS)
- Sieć Wi-Fi oraz połączenia mobilne, aby sprawdzić działanie systemu w różnych warunkach sieciowych

C2. Efekty realizacji projektu

1. Przykładowy proces korzystania z oprogramowania PipeQ

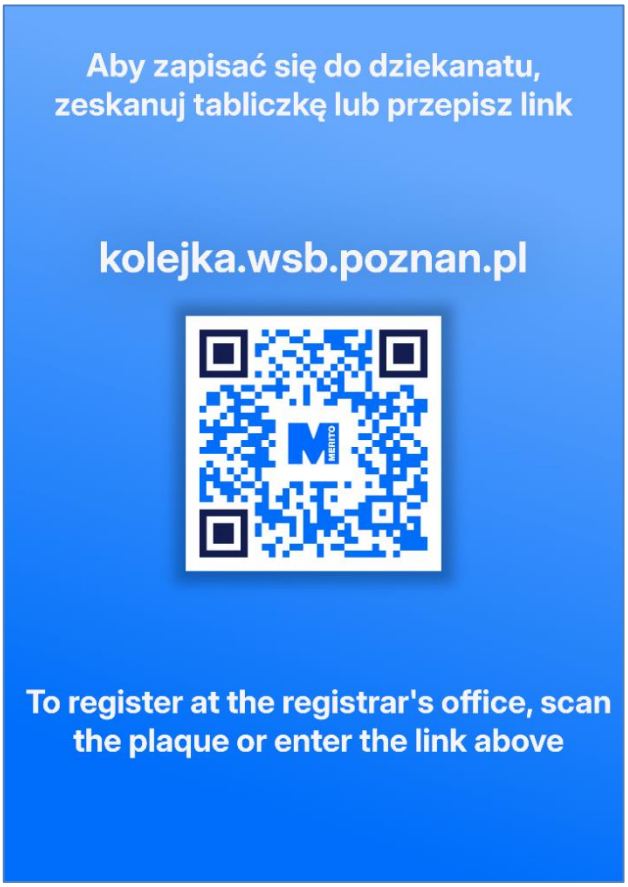
1.1 Dostęp do aplikacji

Nasz projekt całkowicie rezygnuje z papierowych biletów oraz wszelkich fizycznych form kolejkowania studentów. Cały proces odbywa się w pełni elektronicznie, za pośrednictwem smartfona. Uważamy, że w dzisiejszych czasach, gdy każdy posiada telefon komórkowy, takie rozwiązanie jest nie tylko wygodne, ale także bardziej przyjazne dla środowiska.

Aby zarejestrować swoje zgłoszenie, student udający się do dziekanatu musi znaleźć się na stronie kolejka.wsb.poznan.pl. Mając na uwadze kwestie dostępności, zapewniliśmy dwie metody uzyskania dostępu do systemu, aby każdy miał możliwość skorzystania z Biura Obsługi Studenta.

1.1.1 Zeskanowanie kodu QR

Przed dziekanatem umieszczona została tabliczka z kodem QR, który po zeskanowaniu przekierowuje użytkownika na stronę https://kolejka.wsb.poznan.pl. Kod QR został wygenerowany za pomocą darmowego narzędzia qrcodemonkey[2].



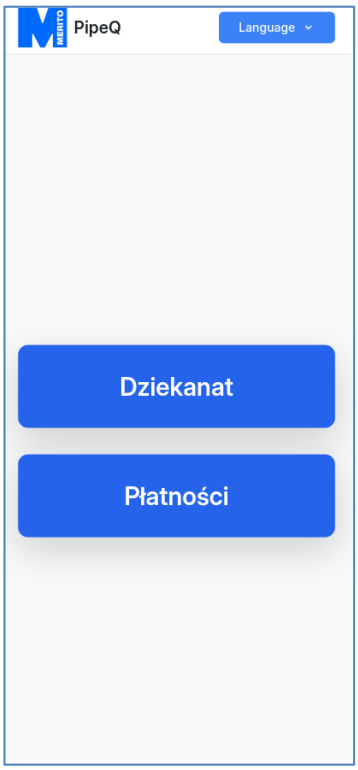
Rysunek 2. Tabliczka z kodem QR przed wejściem do dziekanatu

Jest to preferowana metoda uzyskania dostępu do naszej aplikacji, jednak z uwagi na możliwość braku odpowiedniego oprogramowania do skanowania kodów QR na urządzeniu użytkownika, konieczne było wdrożenie alternatywnej, manualnej ścieżki dostępu.

1.1.2 Ręczne wpisanie linku do przeglądarki

Wiele osób nie jest nawet świadomych, że ich aparat fotograficzny posiada wbudowaną funkcję skanowania kodów QR, co może stanowić barierę w korzystaniu z systemu. Aby uwzględnić możliwość problemów ze skanowaniem, na tabliczce umieszczono także adres URL prowadzący do strony internetowej. Użytkownik może go ręcznie wpisać w przeglądarce swojego smartfona, co zapewnia alternatywną drogę dostępu do aplikacji.

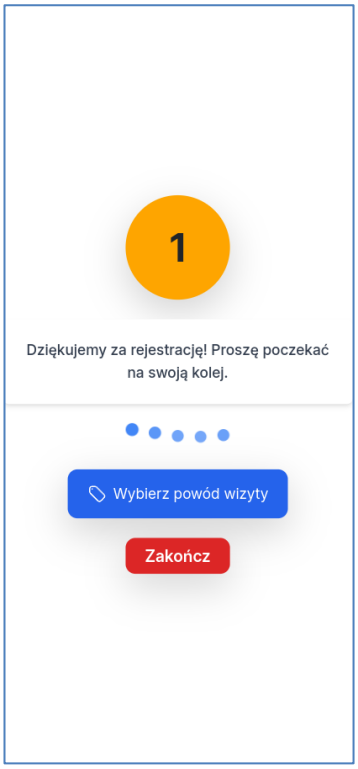
1.2 Wybór miejsca docelowego



Rysunek 3. Ekran powitalny użytkownika

Po wejściu na stronę naszej aplikacji użytkownik ma możliwość wyboru jednego z dwóch dostępnych miejsc docelowych. Zważywszy na fakt, że na Uniwersytecie studiuje wielu studentów nieposługujących się językiem polskim, nasza aplikacja oferuje pełne wsparcie wielojęzyczne. Po wybraniu opcji "Language" w menu, interfejs dostosowuje się do preferencji użytkownika, umożliwiając korzystanie z aplikacji w języku polskim oraz angielskim. Aby zapisać się do kolejki, wystarczy jedno kliknięcie odpowiedniego przycisku – system natychmiast rejestruje użytkownika i przypisuje mu unikalny numer w kolejce oczekujących.

Jeżeli student przypadkowo wybierze niewłaściwe miejsce docelowe, ma możliwość anulowania swojego biletu, co pozwala mu powrócić do ekranu początkowego i dokonać nowego wyboru. Podczas oczekiwania student może dodatkowo określić powód swojej wizyty w dziekanacie, co usprawnia obsługę i pozwala pracownikom na dostosowanie niektórych stanowisk do konkretnych akcji, na przykład przeznaczenie jednego stanowiska na przedłużanie legitymacji a drugiego na odbiór dokumentów. Szczegółowe omówienie tej funkcjonalności znajduje się w sekcji do dodatkowej funkcjonalności.



Rysunek 4. Ekran oczekiwania na wezwanie

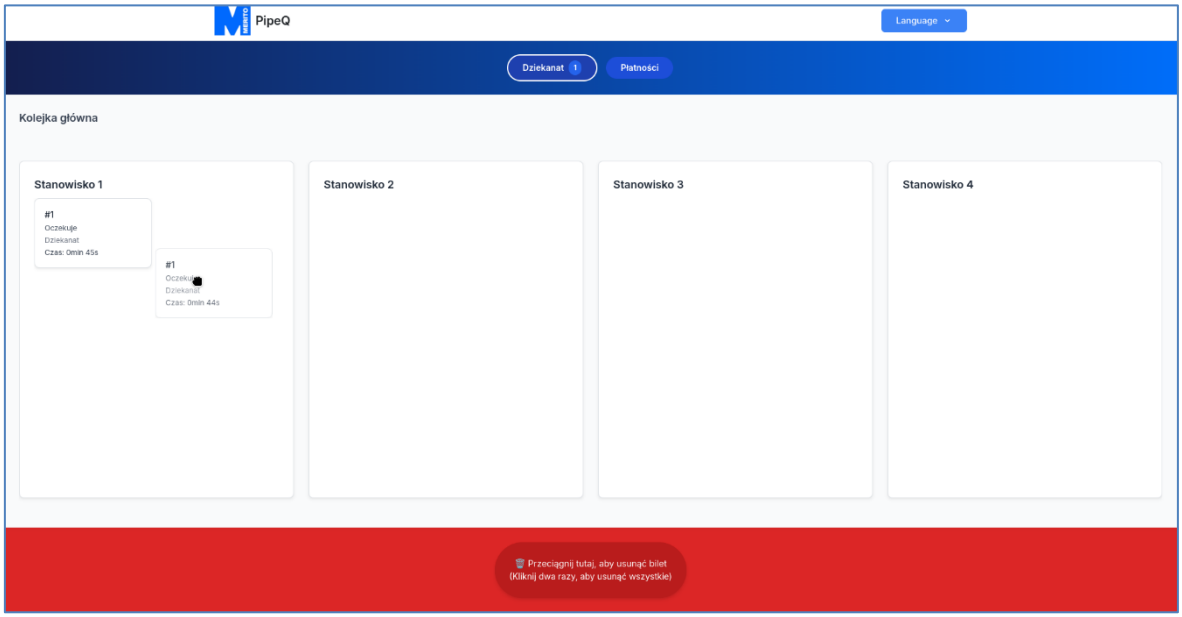
Opis techniczny:

Wybór miejsca docelowego inicjuje asynchroniczne żądanie HTTP POST na endpoint /register/{destination_id}. Odpowiedź zwraca status: zamknięcie dziekanatu, ponowną

rejestrację lub akceptację zgłoszenia, wraz z kodem HTTP i unikalnym tokenem biletu (UUID). Takie rozwiązanie eliminuje IDOR[3] (Insecure direct object references).
Token ten służy jako identyfikator kanału WebSocket, umożliwiając przeglądarce nawiązanie szyfrowanego połączenia z serwerem Reverb przez WebSocket Secure (WSS). Od tego momentu cała komunikacja serwer -> klient odbywa się przez ten protokół. Dzięki utrzymywanym połączeniu i małych ramkach jest to najszybszy sposób wymiany informacji w sieci web.

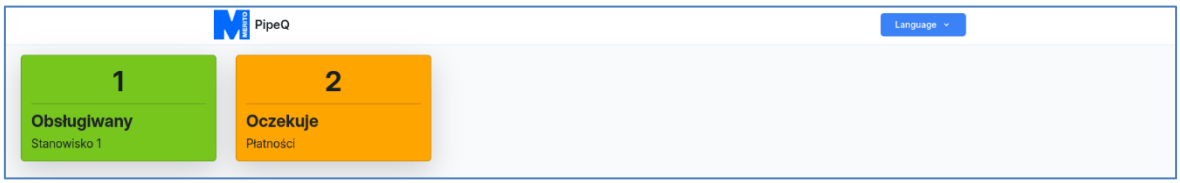
1.3 Wpuszczenie użytkownika na wybrane stanowisko

Natychmiast po wyborze miejsca docelowego przez użytkownika, w panelu koordynatora pojawia się nowy bilet. Przeciągnięcie go na wybrane stanowisko skutkuje natychmiastowym przydzieleniem użytkownika do obsługi, umożliwiając płynne i intuicyjne zarządzanie kolejką.



Rysunek 5. Koordynator przeciągający bilet na wybrane stanowisko

Bilet pojawia się na ekranie głównym, wzbogacony o płynną animację oraz krótki sygnał dźwiękowy. Połączenie efektu wizualnego i akustycznego skutecznie przyciąga uwagę użytkownika, sygnalizując zmianę w kolejce i skłaniając do weryfikacji, czy komunikat dotyczy właśnie jego biletu.



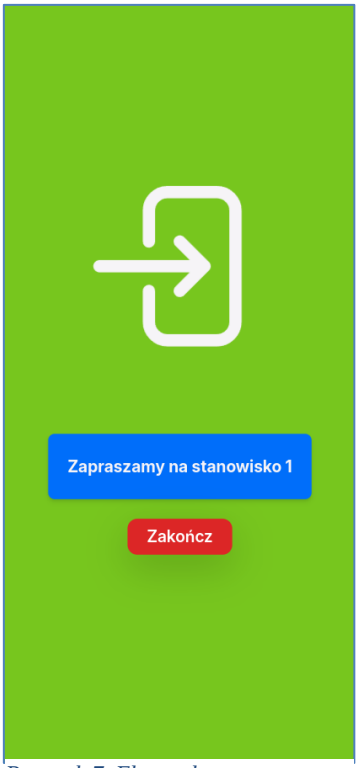
Rysunek 6. Natychmiastowa zmiana wizualna biletu na ekranie przed dziekanatem. Zmianie towarzyszy animacja oraz sygnał dźwiękowy

Opis techniczny:

Koordynator, otwierając swój widok, automatycznie dołącza do kanału WebSocket "display", znajdującego się za middlewarem uwierzytelniającym – brak potrzeby stosowania tokena. Każda akcja studenta, wykonana poprzez żądanie HTTP, wyzwala odpowiednią reakcję, wracającą do niego przez WebSocket oraz duplikowaną i przetwarzaną na kanale "display". Interakcje koordynatora z interfejsem przekładają się na odpowiednie zapytania API. Przykładowo, przeniesienie biletu na stanowisko wywołuje żądanie do /move/{ticket_id}/{workstation_id?}, które następnie poprzez WebSocket informuje studenta o aktualizacji statusu i numerze stanowiska. Ostatni argument w ścieżce żądania API jest opcjonalny. W przypadku przekazania wartości null, bilet zostaje zwrócony do kolejki z ustawionym statusem "oczekiwanie na wejście".

1.4 Sygnał wejścia użytkownika

Student natychmiast otrzymuje informację o możliwości wejścia na wyznaczone stanowisko. Oprócz wizualnej zmiany tła na jaskrawozielone, centralnie umieszczona animacja dynamicznie poruszającego się symbolu wejścia skutecznie przyciąga uwagę. Dodatkowo, jeśli urządzenie użytkownika obsługuje odpowiedni protokół, telefon wibruje, jeszcze bardziej intensyfikując sygnał wezwania do dziekanatu. Wszystkie te elementy razem sprawiają, że komunikat wejścia jest czytelny, trudny do przeoczenia i eliminuje zastoje w kolejce.



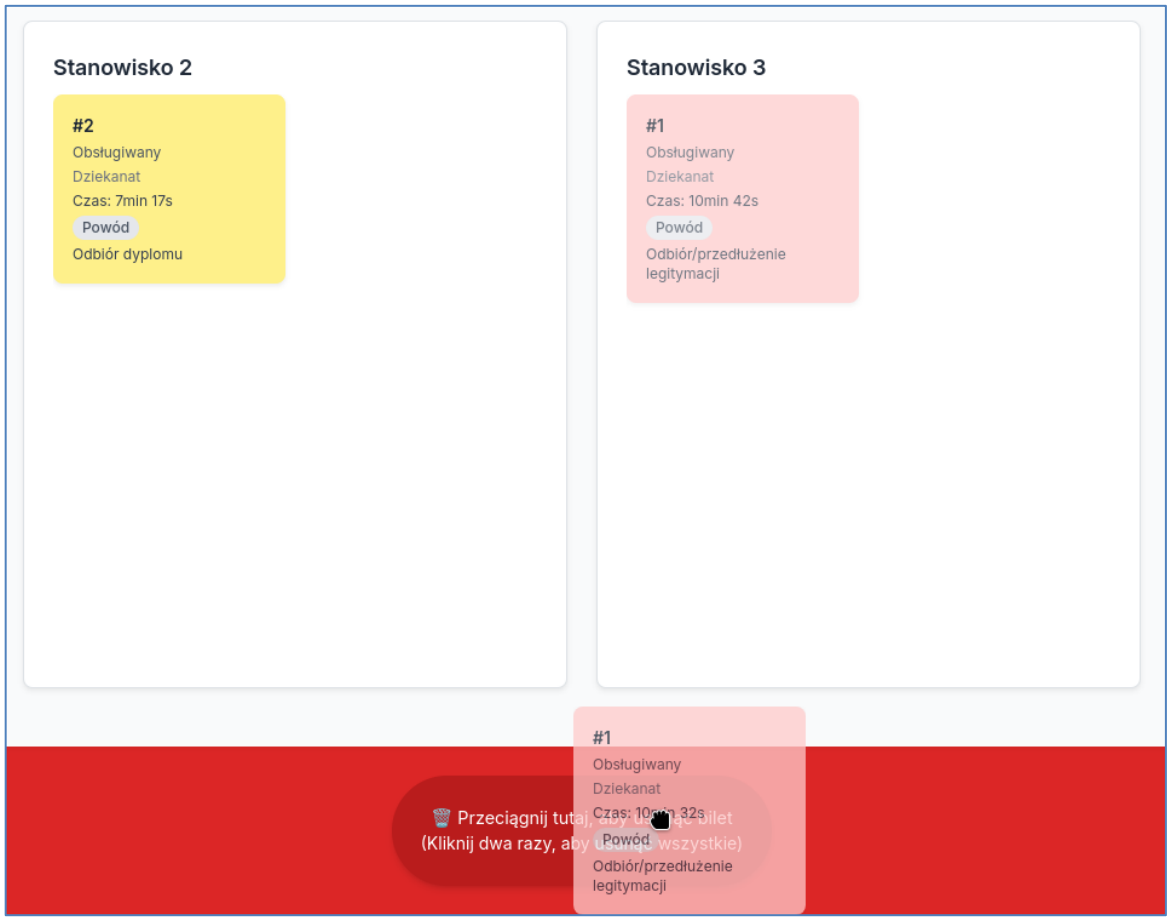
Rysunek 7. Ekran ekranu użytkownika po nadaniu sygnału wejścia. Zmianie wizualnej towarzyszy animacja

Opis techniczny:

Komponent odpowiedzialny za wibrację telefonu uruchamia się przy wywołaniu studenta zdarzeniem Laravela UpdateUserAboutHisTicket i zmianie statusu biletu na „in”. Jeżeli przeglądarka obsługuje API navigator.vibrate, urządzenie wykonuje sekwencję wibracji 200 ms, 100 ms przerwy, 200 ms. Obsługa tej funkcji jest warunkowa, co zapobiega ewentualnym błędom na niekompatybilnych urządzeniach.

1.5 Zakończenie cyklu

Koordynator przeciąga zakończony bilet na obszar usuwania, co inicjuje proces jego wycofania z systemu. Informacja o usunięciu jest natychmiast przekazywana użytkownikowi, który zostaje przeniesiony z powrotem do ekranu wyboru miejsca docelowego. Jednocześnie ekran nad wejściem do dziekanatu aktualizuje wyświetlaną listę, usuwając nieaktywny bilet, zapewniając przejrzystość i bieżącą synchronizację systemu.



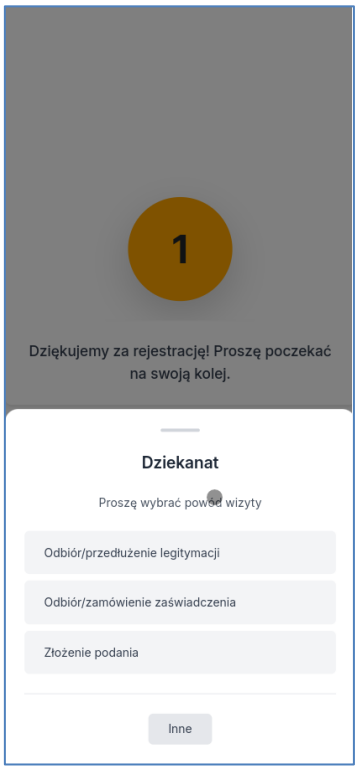
Rysunek 8. Usunięcie biletu z kolejki przez koordynatora

Opis techniczny:

Zdarzenie NotifyEndedTicketUser w Laravelu inicjuje asynchroniczne żądanie POST do zasobu /clearStorage, które usuwa token sesyjny z ciasteczek oraz resetuje sesję użytkownika, przygotowując system na ewentualne pobranie nowego numeru. NotifyEndedTicketDisplay rozgłasza zdarzenie do wszystkich widoków połączonych z protokołem WebSocket na kanale “Display”, sygnalizując zakończenie śledzenia biletu i usunięcie go z interfejsu.

2. Dodatkowe funkcjonalności programu PipeQ

2.1 Uzupełnianie powodu przyjścia



Rysunek 10. Wybranie powodu przyjścia przez użytkownika

Po zarejestrowaniu się do wybranego miejsca docelowego użytkownik otrzymuje możliwość podania powodu swojej wizyty w Biurze Obsługi Studenta. Nie jest to wymagane, lecz po kliknięciu odpowiedniego przycisku otwiera się lista najczęstszych powodów wizyt, z której można wybrać właściwą opcję. Powody są przypisane do miejsca docelowego, co oznacza, że powody przyjścia będą się różnić między dziekanatem i a płatnościami. Wybrany powód zostaje następnie przypisany do aktywnego biletu i przesłany na widok koordynatora.

Funkcjonalność ta znacząco poprawia efektywność pracy dziekanatu. Przykładowo, na początku semestru długie kolejki tworzą studenci chcący przedłużyć legitymację. Możliwość wcześniejszego określenia celu wizyty pozwala na dynamiczne zarządzanie stanowiskami, na przykład poprzez przeznaczenie jednego z nich wyłącznie do obsługi najczęściej wybieranego powodu, co bezpośrednio wpływa na szybkość przetwarzania zgłoszeń.

Rysunek 9. Powód przyjścia dodany do biletu na widoku koordynatora

Dodatkowo, na liście dostępna jest opcja „Inne”, która nie wykonuje żadnej akcji poza zamknięciem okna wyboru. To subtelne zagranie psychologiczne sprawia, że użytkownik nie czuje się zobowiązany do podania powodu wizyty, jednocześnie odnosząc wrażenie, że dodał dodatkową informację do swojego zgłoszenia.

2.2 Autoodzyskiwanie wcześniej pobranego biletu

Zakładamy, że studenci po zarejestrowaniu się w kolejce mogą przełączyć się na inną aplikację lub nawet całkowicie zamknąć przeglądarkę, co skutkuje zerwaniem połączenia WebSocket. Aby nie dopuścić do utracenia wcześniej zarejestrowanego biletu, system zapisuje unikalny token w ciasteczkach użytkownika. Po ponownym otwarciu strony aplikacja automatycznie odczytuje wartość ticket_token i na jej podstawie wznawia połączenie z odpowiednim kanałem WebSocket.

Dzięki temu, bez względu na wcześniejsze zamknięcie sesji, użytkownik zostaje przywrócony do aktualnego etapu obsługi biletu. Dodatkowo, mechanizm szyfrowania ciasteczek w Laravelu gwarantuje ich integralność – sesyjnie podpisane, nie mogą zostać sfałszowane, co zapobiega manipulacji tokenem biletu.

2.3 Harmonogram otwarcia miejsc docelowych

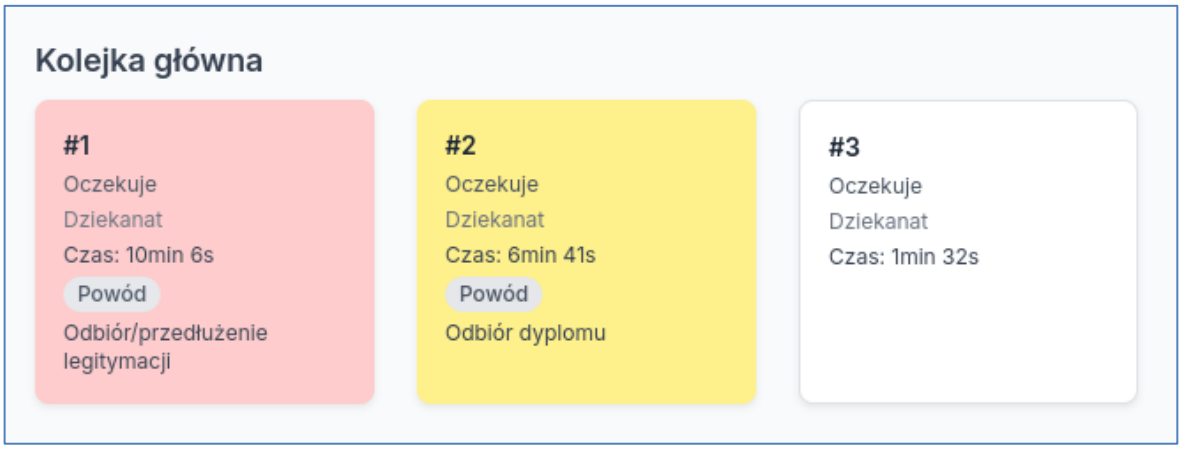
Biuro Obsługi Studenta działa według ustalonego harmonogramu, a w niektóre dni pozostaje całkowicie zamknięte. W związku z tym nasz system został zaprojektowany tak, aby umożliwiać dynamiczną edycję godzin otwarcia i automatyczne dostosowywanie interfejsu do bieżącej dostępności. Na ekranie użytkownika wyświetlane są jedynie przyciski prowadzące do aktualnie otwartych miejsc docelowych, co eliminuje możliwość wyboru z góry niedostępnych opcji. Dzięki temu interfejs pozostaje przejrzysty, a użytkownik nie jest wprowadzany w błąd.

Rysunek 11. Informacja o zamkniętym dziekanacie wraz z datą następnego otwarcia

W przypadku, gdy wszystkie stanowiska są zamknięte, system automatycznie przekierowuje go na stronę informacyjną, która nie tylko komunikuje brak dostępności, ale także wskazuje najbliższy termin otwarcia oraz alternatywne formy kontaktu, takie jak adres e-mail. Naturalnie tekst komunikatu dostępny jest również w innych językach.

2.4 Czas oczekiwania biletów

Na widoku koordynatora każdy bilet wyświetla czas oczekiwania na obsługę, a jego kolor dynamicznie zmienia się w zależności od momentu zarejestrowania. Po upływie 5 minut bilet przybiera żółty odcień, a po 10 minutach zaczyna pulsować na czerwono. Dzięki temu system w czytelny sposób wyróżnia bilety wymagające priorytetowej obsługi, umożliwiając koordynatorowi szybkie podejmowanie decyzji. Taka wizualna klarowność pozwala jednym spojrzeniem ocenić sytuację i, jeśli to konieczne, zwiększyć przepustowość kolejki poprzez otwarcie dodatkowego stanowiska.



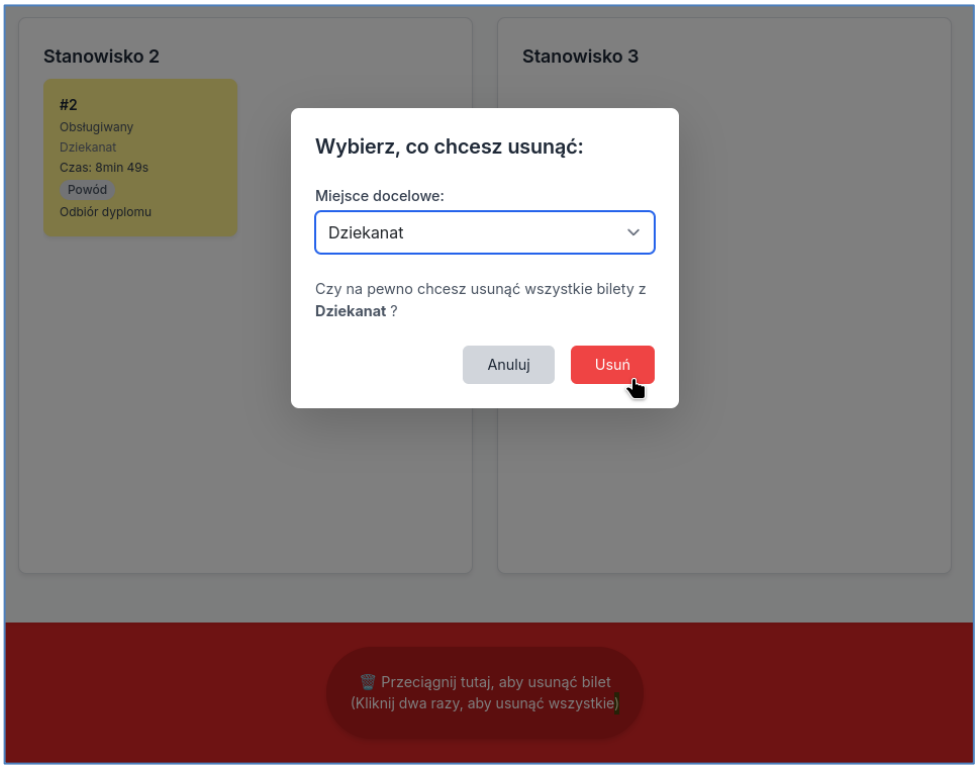
Rysunek 12. Kolor biletów w zależności od długości czasu oczekiwania

2.5 Ponowne wysłanie sygnału wejścia

Studenci spędzający dłuższy czas w oczekiwaniu przed dziekanatem mogą być pochłonięci korzystaniem z telefonu, co sprawia, że nie zawsze reagują na pierwsze wywołanie. W takiej sytuacji koordynator może ponowić wezwanie, klikając dwukrotnie na bilet przypisany do stanowiska. Akcja ta ponownie uruchamia funkcję powiadomienia – telefon użytkownika wibruje, emitowany jest sygnał dźwiękowy, a na ekranie wyświetlającym wszystkie bilety pojawia się animacja.

2.6 Masowe usuwanie biletów

Świadomi ograniczeń technologicznych każdego systemu informatycznego, wyposażyliśmy panel koordynatora w funkcję masowego usuwania biletów. Nie jest to jedynie operacja po stronie interfejsu użytkownika, lecz dedykowana funkcjonalność, która bezpośrednio czyści odpowiednią tabelę w bazie danych. Dzięki temu, nawet w przypadku, gdy na koniec dnia pozostaną niezamknięte bilety, które w nieoczekiwany sposób nadal widnieją na liście, funkcja ta gwarantuje ich całkowite usunięcie, eliminując wszelkie niepożądane wpisy i przywracając pełną integralność systemu.



Rysunek 13. Usuwanie wszystkich biletów zakolejkowanych do dziekanatu

Usunięcie biletów inicjowane jest poprzez dwukrotne kliknięcie w obszar, który standardowo służy do ich przesuwania. Po tej akcji system wyświetla okno potwierdzenia, eliminując przypadkowe uruchomienie tej funkcjonalności. Koordynator ma możliwość wyboru, czy usunąć wszystkie bilety, czy jedynie te przypisane do konkretnego miejsca docelowego.

3. Wykorzystane narzędzia i struktura danych

3.1 Narzędzia wykorzystane przy projekcie PipeQ

3.1.1 Technologie backendowe

- Laravel 11 - Nowoczesny framework PHP. Umożliwił nam wielką swobodę wykorzystania pozostałych narzędzi, oferując przy tym wiele ułatwień i wbudowanych funkcjonalności automatyzujących.

- Komponent Laravel Reverb - Jedna z wtyczek ekosystemu Laravela. To dzięki niemu mogliśmy użyć protokołu WebSocket i w błyskawiczny sposób wymieniać dane między serwerem a użytkownikiem. Protokół ten utrzymuje trwałe połączenie, eliminując koszt nawiązywania komunikacji protokołu HTTP za każdym razem.

- Supervisor - Proste narzędzie w systemie Linux do zarządzania aktywnymi usługami. W naszym projekcie wykorzystaliśmy go do automatycznego uruchamiania usługi serwerowej Laravel Reverb. W razie nagłego wyłączenia serwera, supervisor jest w stanie automatycznie ponownie rozpocząć działanie serwera Reverb.

- SQLite3 - Lekka i bardzo szybka baza danych. Z racji, że nasza aplikacja była nastawiona głównie na szybkość i nie potrzebuje wielowątkowej obsługi transakcji równoczesnych, wykorzystanie sekwencyjnej bazy danych do systemu kolejkowego jest idealnym rozwiązaniem.

- Pusher.js - Biblioteka umożliwiająca dynamiczną komunikację w czasie rzeczywistym, co pozwala na natychmiastowe aktualizacje interfejsu użytkownika w odpowiedzi na zmiany na serwerze.

- Google Lighthouse - Narzędzie do audytowania wydajności i jakości stron internetowych. Dzięki niemu mogliśmy przeprowadzić szczegółową analizę renderowania naszej aplikacji.

3.1.2 Technologie frontendowe

- Vue.js - Popularny framework języka JavaScript. Wykorzystanie go w projekcie było kluczowe i pozwoliło nam na dużą responsywność na widoku użytkownika i koordynatora, piękne animacje oraz możliwość dynamicznego skalowania naszej aplikacji według nowych potrzeb.

- Bootstrap - Framework CSS, który wykorzystaliśmy między innymi do zaprojektowania ekranu wyświetlającego wszystkie aktywne zgłoszenia przed wejściem do Biura Obsługi Studenta.

- Tailwind CSS - Framework CSS oparty na klasach narzędziowych (utility-first). Dzięki niemu mogliśmy szybko i efektywnie stylizować interfejs użytkownika, zapewniając pełną responsywność i estetyczny wygląd aplikacji.

- Sass - Preprocesor CSS, który pozwolił nam na bardziej strukturalne zarządzanie stylami w projekcie, ułatwiając pracę ze zmiennymi, zagnieżdżenia i inne zaawansowane funkcje ułatwiające rozwój interfejsu użytkownika.

- Pinia - Nowoczesna biblioteka do zarządzania stanem w aplikacjach Vue.js, pozwalająca na łatwe zarządzanie danymi globalnymi i synchronizowanie stanu aplikacji między komponentami.

- Vite - Narzędzie do budowania aplikacji Vue.js, oferujące błyskawiczny Hot Module Replacement (HMR), co znacznie przyspiesza proces budowania i testowania aplikacji.

- VueDraggable - Biblioteka oparta na Sortable.js, którą wykorzystaliśmy do implementacji funkcji przeciągania i upuszczania zgłoszeń w interfejsie użytkownika, umożliwiając intuicyjne zarządzanie zgłoszeniami.

- Alpine.js - lekki framework JavaScript, który wykorzystaliśmy do dodania prostych interakcji na widokach użytkownika bez konieczności użycia pełnowymiarowego frameworka JS.

- Axios - Biblioteka języka JavaScript do asynchronicznej komunikacji z backendem za pośrednictwem zapytań HTTP.

- HTML, CSS, JavaScript - Fundamentalne języki tworzenia stron internetowych.

3.1.3 Narzędzia wspomagające development

- Docker - Platforma konteneryzacji, dzięki której byliśmy w stanie w szybki sposób wdrożyć aplikację w środowisku produkcyjnym, bez konieczności manualnej konfiguracji serwera.

- Visual Studio Code - Zaawansowane środowisko programistyczne, którym posługiwaliśmy się przy tworzeniu naszego projektu.

- Git - Najpopularniejszy system kontroli wersji. Umożliwił równoległą pracę nad wieloma funkcjonalnościami systemu jednocześnie.

- GitHub - Platforma hostingowa repozytoriów Git, usprawniająca współpracę nad kodem.

3.2. Modele bazy danych

3.2.1 Wybór bazy danych

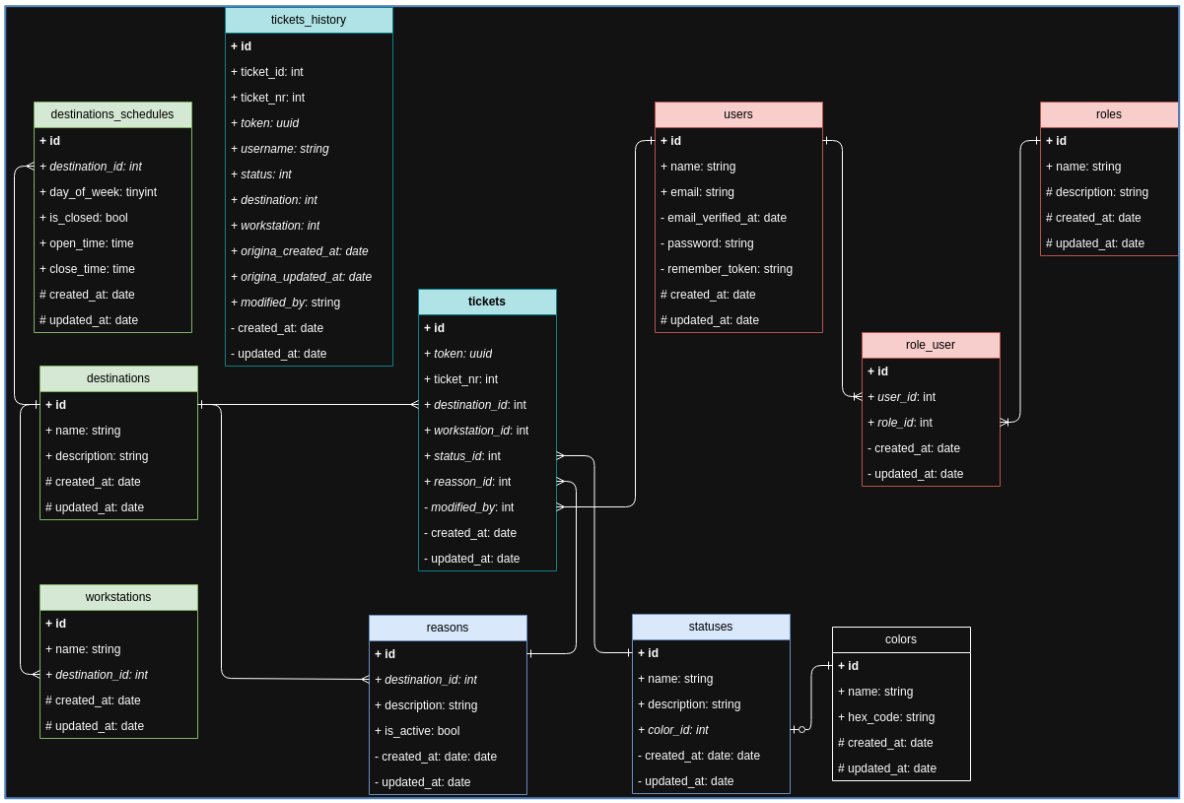
Jako model bazy danych wybraliśmy SQLite3, ze względu na jej wyjątkową szybkość, prostotę oraz łatwość obsługi. Choć jej głównym ograniczeniem jest brak wsparcia dla jednoczesnego, wielowątkowego zapisu, w kontekście systemu kolejkowego – który działa w sposób sekwencyjny, a nie równoległy – nie stanowi to żadnego problemu. Struktura bazy pozostaje lekka i efektywna, a jedynym bardziej zaawansowanym mechanizmem, jaki zastosowaliśmy, są dwa triggery na tabeli ticket, które automatycznie aktualizują historię zgłoszeń w tabeli ticket_history. Poza tymi mechanizmami nie korzystaliśmy z żadnych niestandardowych funkcji bazy danych.

3.2.2 Laravel Eloquent ORM

Nasza aplikacja opiera się głównie na Eloquent ORM, który pozwala na odwzorowanie tabel w bazie danych jako obiekty odpowiadające klasom w kodzie aplikacji. Dzięki wykorzystaniu plików migracyjnych Laravel, proces definiowania struktury bazy danych za pomocą poleceń DDL został w pełni zautomatyzowany. To podejście umożliwia łatwe i szybkie odtworzenie identycznej struktury bazy na dowolnym silniku bazodanowym obsługiwanym przez Laravel. Dzięki temu nasza aplikacja jest bardzo elastyczna i może dołączyć się do już wykorzystywanej bazy danych w przedsiębiorstwie, na przykład jako osobny schemat w bazie PostgreSQL.

3.2.3 Struktura modeli

Nasza baza danych została znormalizowana do trzeciej postaci normalnej, co zapewnia jej wysoką spójność. Dzięki starannej atomizacji tabel struktura pozostaje przejrzysta, łatwa w zarządzaniu oraz elastyczna, umożliwiając bezproblemowe rozszerzanie funkcjonalności w przyszłości.



Rysunek 14. iagram ERD oprogramowania PipeQ

- Zielony kolor odzwierciedla rzeczywistą infrastrukturę budynku, zapewniając użytkownikom czytelną wizualizację. System działa w pełni dynamicznie – każdorazowe wyświetlenie strony powoduje automatyczne pobranie aktualnych danych z bazy. Dzięki temu dodanie nowego rekordu, na przykład nowego destination i przypisanie do niego workstation, jest natychmiast uwzględniane w aplikacji i odpowiednio wyświetlane.

- Tabele oznaczone kolorem czerwonym odpowiadają za relację między użytkownikami a przypisanymi im rolami. System został zaprojektowany w sposób elastyczny, umożliwiając przypisanie wielu ról do jednego użytkownika. Przypisana rola odpowiada za dostęp do odpowiednich widoków, takich jak coordinator czy administrator.

- Kolorem niebieskim wyróżnione zostały tabele zawierające dodatkowe informacje dotyczące biletów. Stanowią one rozszerzenie podstawowych danych, wzbogacając je o szczegółowe opisy i parametry związane z procesem obsługi zgłoszeń. Dzięki temu system przechowuje nie tylko sam numer biletu, ale również kontekst jego przetwarzania.

3.2.4 Opis najważniejszych relacji

- Destinations ⇒ Workstations (1:N)
  Miejsca docelowe (destinations) mogą mieć wiele stanowisk (workstations). W naszym przypadku, destination "dziekanat" posiada trzy stanowiska, natomiast destination "płatności" posiada tylko jedno stanowisko.

- Destinations ⇒ Destinations_schedules (1:N)
  Każde miejsce docelowe (destination) może mieć wiele harmonogramów otwarcia (destinations_schedules). Relacja w naszym przypadku występuje w stosunku 1 do 7, ponieważ dla każdego miejsca można ustawić godziny otwarcia na każdy dzień tygodnia.

- Reasons ⇒ Destinations (N:1), Reasons ⇒ Tickets (1:N)
  Powody wizyty (reasons) są powiązane z miejscami docelowymi (destinations), co oznacza, że każdy powód musi być przypisany do konkretnego miejsca docelowego. Użytkownik, wybierając destination, może podać powód przyjścia. Powód jest następnie przypisany do zgłoszenia (ticket), dzięki czemu można śledzić motyw wizyty dla każdego biletu.

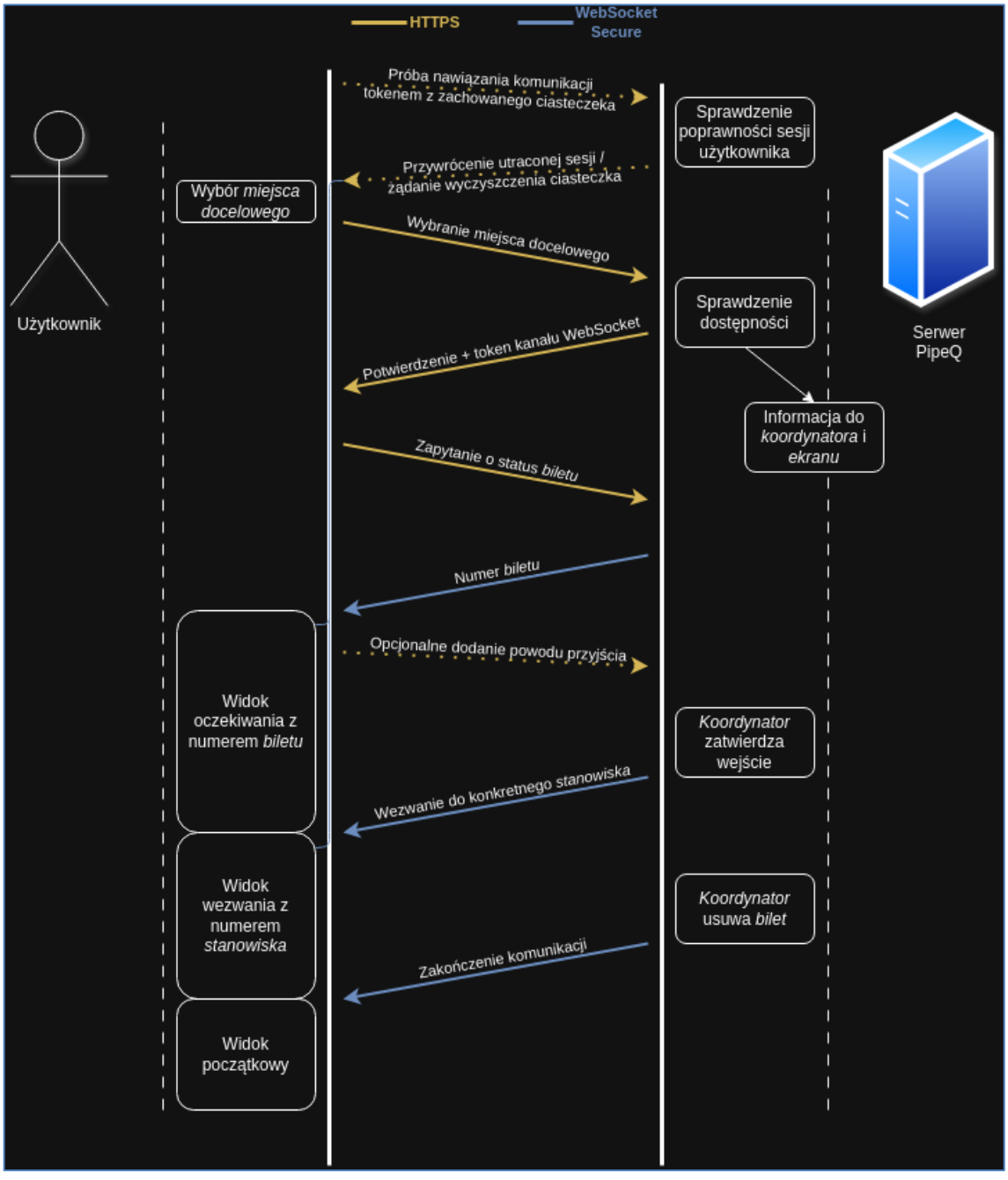
- Statuses ⇒ Colors (1:1), Statuses ⇒ Tickets (1:N)
  Każdy status (statuses) posiada swój własny kolor (colors), co pozwala na wizualne rozróżnienie etapów przetwarzania zgłoszeń. Przykładowo, status wejścia do miejsca docelowego jest oznaczony jaskrawozielonym kolorem dla lepszej czytelności. Status jest również nierozerwalnie przypisany do biletu (ticket), aby kontrolować jego aktualny etap.

- Role_user ⇒ Users (N:M), Role_user ⇒ Roles (N:M)
  Tabela role_user to tabela pośrednia (many-to-many), która agreguje użytkowników (users) i ich role (roles). Pozwala to na przypisanie wielu ról do jednego użytkownika oraz wielu użytkowników do jednej roli.

- Tickets_history
  Tabela tickets_history przechowuje historię wszystkich zmian, jakie zaszły w tabeli tickets. Nie posiada bezpośrednich relacji z innymi tabelami, ponieważ jej głównym celem jest archiwizacja danych. Jest aktualizowana przez dwa triggery na tabeli tickets, co zapewnia, że każda zmiana w zgłoszeniach (tickets) zostanie zapisana. Dzięki temu nawet jeśli klucze obce w głównej tabeli ulegną zmianie, administrator nadal będzie w stanie prześledzić pełną historię biletu, co umożliwia audyt wszystkich operacji.

4. Komunikacja klient ⇒ serwer

Ze względu na specyfikę komponentu Laravel Reverb, komunikacja WebSocket może przebiegać wyłącznie w kierunku serwer → klient. Zanim jednak nawiązane zostanie stałe połączenie, użytkownik musi wysłać asynchroniczne żądanie HTTPS, w którym zarejestruje się do kolejki. W odpowiedzi otrzymuje unikalny token, pełniący rolę identyfikatora kanału WebSocket. Od tego momentu cała dalsza komunikacja wychodząca z serwera odbywa się wyłącznie za pośrednictwem protokołu WebSocket, zapewniając błyskawicznie szybką aktualizację danych.



Rysunek 15. Schemat nawiązywania połączenia użytkownika z serwerem aplikacyjnym PipeQ

C3. Użyteczność projektu

1. Najistotniejsze zmiany

System kolejkowy PipeQ stworzyliśmy z myślą o usprawnieniu procesów obsługi interesantów w różnych instytucjach, gdzie zachowanie kolejności jest kluczowe. W założeniu system eliminuje problem długiego oczekiwania oraz chaosu organizacyjnego. Główną zaletą naszego systemu jest błyskawiczna aktualizacja statusu zgłoszeń. W przeciwieństwie do tradycyjnych systemów, my wykorzystaliśmy technologię WebSocket, co oznacza, że użytkownik otrzymuje natychmiastowe informacje o zmianie statusu swojego zgłoszenia bez konieczności odświeżania strony. Użytkownicy mają łatwy dostęp do systemu i mogą zarejestrować się do dziekanatu poprzez zeskanowanie kodu QR dostępnego w budynku. W przeciwieństwie do innych systemów kolejkowych, PipeQ nie wymaga logowania, co oznacza, że mogą z niego korzystać również osoby bez dostępu do systemów wewnętrznych uczelni (na przykład prawni opiekunowie załatwiający sprawy w imieniu studentów).

1.1 Użyteczność dla koordynatorów

Koordynatorzy, czyli pracownicy dziekanatu, zyskali nowoczesne narzędzie znacząco podnoszące komfort pracy. Od momentu wdrożenia systemu PipeQ w Biurze Obsługi Studenta szybkość obsługi studentów wyraźnie wzrosła. Wyeliminowanie wąskiego gardła, jakim w poprzednim systemie był długi czas odświeżania statusów biletów, bezpośrednio przyczyniło się do poprawy szybkości świadczonych usług na Uniwersytecie WSB Merito w Poznaniu.

System został poddany testom w okresie standardowej aktywności studentów odwiedzających dziekanat. Choć nie obejmowały one szczytowych momentów roku akademickiego, takich jak masowe przedłużanie legitymacji, przewidujemy, że dzięki możliwości podawania celu wizyty obciążenie poszczególnych stanowisk będzie mogło być lepiej zbalansowane. Odpowiednie rozłożenie zgłoszeń powinno przełożyć się na optymalizację pracy dziekanatu i zwiększenie ogólnej przepustowości systemu.

Oprócz przyspieszenia obsługi, przejrzysty interfejs koordynatora pozwala na dynamiczne dostosowywanie mocy przerobowej dziekanatu do aktualnych potrzeb. Dzięki jednostronicowej strukturze widoku wystarczy jedno spojrzenie, aby ocenić sytuację i w razie potrzeby uruchomić dodatkowe stanowisko obsługi.

1.2 Użyteczność dla użytkowników

Studenci korzystający z nowego systemu PipeQ są znacznie mniej sfrustrowani. Brak konieczności logowania oraz uproszczony proces rejestracji sprawiają, że PipeQ jest intuicyjny i nieinwazyjny w obsłudze. Zminimalizowanie liczby kroków wymaganych do dołączenia do kolejki sprawia, że system nie stanowi bariery, lecz naturalnie wspiera użytkownika w szybkim i bezproblemowym załatwieniu spraw administracyjnych.

Możliwość bieżącego monitorowania statusu zgłoszenia daje studentom poczucie kontroli nad procesem, jednocześnie redukując stres związany z niepewnością i dezorientacją. Dzięki wielokanałowemu systemowi powiadomień studenci są na bieżąco informowani o ich kolejności, co znacząco skraca czas ich reakcji i przyspiesza cały proces obsługi. W efekcie kolejka porusza się płynniej, eliminując zbędne przestoje i zwiększając ogólną wydajność systemu.

W istocie, nasz system zrealizował kluczowe założenie – dzięki minimalnej inwazyjności program, stał się niemal niewidoczny dla użytkowników. To, co wcześniej było chaotyczną i frustrującą próbą siłowej cyfryzacji, przekształciło się w elegancki, uporządkowany mechanizm samoorganizacji studentów. PipeQ nie tylko usprawnił proces kolejkowania, lecz stał się symbolem dobrze zaprojektowanej integracji technologii z codziennym życiem – tam, gdzie cyfryzacja przestaje być przeszkodą, a staje się naturalnym i niemal niewyczuwalnym elementem organizacji.

2. Środowisko wdrożeniowe

2.1 Wpływ wdrożenia oprogramowania PipeQ

Wdrożenie systemu PipeQ przyniosło realne korzyści dla Uniwersytetu WSB Merito w Poznaniu, podnosząc standardy obsługi administracyjnej i pozytywnie wpływając na całościowy obraz doświadczeń uczęszczania na uniwersytet. Usprawnienie procesów kolejkowania przełożyło się na znaczącą redukcję czasu oczekiwania studentów. Dostarczając nasze autorskie rozwiązanie mieliśmy realne przełożenie na nasze własne warunki studiowania.

2.2 Hipotetyczne alternatywne wdrożenia

Obecnie system został wdrożony wyłącznie w Biurze Obsługi Studenta na Uniwersytecie WSB Merito w Poznaniu. Jednak na uczelni funkcjonuje kilka innych działów, które mogłyby znacząco skorzystać z jego implementacji. Przykładem jest Biuro Karier i Praktyk, gdzie w okresach wzmożonej aktywności, związanej z rozliczaniem dokumentacji praktyk studenckich, PipeQ mógłby skutecznie usprawnić obsługę interesantów.

Największym beneficjentem systemu mógłby okazać się Dział Rekrutacji, przez który w okresie poprzedzającym rozpoczęcie roku akademickiego, przetaczają się setki studentów. W takich warunkach nasz system, dzięki przejrzystej strukturze i dopracowanemu interfejsowi, nie tylko usprawniłby proces kolejkowania, ale również podkreśliłby profesjonalny wizerunek uczelni.

C4. Autoewaluacja zespołu projektowego

I. Krzysztof Pacyna – backend engineer i lider techniczny

Zakres odpowiedzialności

W projekcie odpowiadałem za stworzenie fundamentów backendu, niezbędnych do implementacji kluczowych funkcji systemu. Byłem również odpowiedzialny za zarządzanie zespołem oraz koordynację komunikacji backend <=> frontend.

Do moich obowiązków należało również opracowanie i wdrożenie:

- zaplecza logicznego projektu *(backend)*
- ekranu wyświetlającego wszystkie bilety *(display)*
- panelu administratora

Realizowane zadania

- Zaprojektowanie architektury systemu oraz wybór narzędzi technologicznych
- Opracowanie struktury bazy danych i definicja relacji między modelami
- Implementacja protokołu WebSocket w aplikacji
- Przygotowanie backendu do efektywnej współpracy z frontendem
- Wdrożenie mechanizmu uwierzytelniania anonimowych użytkowników za pomocą sesji przeglądarkowych oraz automatyczne przywracanie połączenia po ponownym otwarciu aplikacji
- Implementacja systemu ról użytkowników i podział aplikacji na strefy dostępne publicznie oraz wymagające uwierzytelnienia lub specjalnych uprawnień
- Przygotowanie wersji Dockerowej projektu
- Archiwizacja logów dotyczących obsługiwanych biletów

Najważniejsze osiągnięcia

- ★ Uruchomienie serwera WebSocket Secure (WSS) ze wsparciem pełnego szyfrowania, wykorzystującego certyfikaty wygenerowane przez Let's Encrypt
- ★ Opracowanie solidnej architektury aplikacji, przewidującej nieszablonowe scenariusze zachowań użytkowników

Nabyte umiejętności

- Znaczna poprawa umiejętności korzystania z systemu kontroli wersji Git
- Znajomość nowych protokołów, w tym WebSocket, TLS oraz konfiguracji serwera webowego NGINX
- Poszerzenie doświadczenia w pracy z nowoczesnymi frameworkami takimi jak Laravel i Vue.js

II. Mateusz Popielarz – rozwój umiejętności i wkład w projekt

Podczas realizacji projektu PipeQ moim głównym zadaniem było opracowanie warstwy frontendowej systemu. Pracowałem z Vue.js, tworząc dynamiczne i responsywne interfejsy użytkownika, a także wdrażając obsługę WebSocketów, co pozwoliło na natychmiastowe aktualizacje zgłoszeń w systemie.

1. Frontend – Vue.js i Tailwind CSS

- Opanowałem Vue.js, w tym zarządzanie komponentami, stanem aplikacji (Pinia) i dynamicznymi interakcjami.
- Wykorzystałem Tailwind CSS do szybkiego i modułowego stylizowania interfejsu.
- Tworzyłem intuicyjne UI dla użytkowników i koordynatorów systemu.

2. WebSockety - aktualizacja zgłoszeń w czasie rzeczywistym

- Wdrożyłem Laravel Reverb i jego integrację z Vue.js.
- Zapewniłem natychmiastową aktualizację statusu zgłoszeń, eliminując konieczność ręcznego odświeżania strony.

3. Praca z GitHubem i zarządzanie projektem

- Korzystałem z Git do kontroli wersji, pracując na oddzielnych branchach i rozwiązując konflikty.
- Współpracowałem z zespołem poprzez Pull Requesty i Code Review.

4. Podstawy backendu – Laravel i SQLite

- Zapoznałem się z strukturą MVC w Laravelu i podstawami API do obsługi biletów.
- Testowałem i optymalizowałem komunikację frontend-backend.

5. Wyzwania i nauka

- Największym wyzwaniem była integracja WebSocketów, wymagająca testów i optymalizacji.
- Poprawiłem responsywność interfejsu, dostosowując system do urządzeń mobilnych.

6. Podsumowanie

Dzięki projektowi rozwinąłem się jako frontend developer, zdobywając praktyczne doświadczenie w Vue.js, WebSocketach i integracji frontendu z backendem. Projekt pozwolił mi lepiej zrozumieć procesy w aplikacjach webowych i skutecznie zarządzać kodem w zespole.

III. Mateusz Rój - Projektant Interfejsu Użytkownika i Tester

Zakres odpowiedzialności:
W projekcie byłem odpowiedzialny za analizę istniejących systemów oraz zaprojektowanie ogólnego wyglądu aplikacji, przeprowadzanie testów oraz utworzenie i aktualizowanie dokumentacji projektowej.

Realizowane zadania:

- Analiza istniejących systemów
- Projektowanie interfejsu użytkownika dostosowany do wymagań
- Menadżer spójności wizualnej

- Przeprowadzenie testów oprogramowania
- Tworzenie i aktualizacja dokumentacji projektowej

Najważniejsze osiągnięcia:

- Zaprojektowanie interfejsu użytkownika, który został pozytywnie oceniony przez użytkowników
- Responsywny design, co oznacza, że oprogramowanie działa poprawnie na różnych urządzeniach
- Utrzymanie spójności wizualnej w całym projekcie
- Przeprowadziłem kompleksowe testy oprogramowania, identyfikując i raportując błędy
- Utworzyłem kompletną i aktualną dokumentację projektową

Nabyte umiejętności

- Nabyłem umiejętność projektowania intuicyjnych i estetycznych interfejsów, które spełniają potrzeby użytkowników i są zgodne z wymaganiami projektu
- Rozwinąłem umiejętność przeprowadzania kompleksowych testów oprogramowania, identyfikowania i raportowania błędów oraz współpracy z programistami w celu ich naprawienia
- Nauczyłem się tworzyć kompletną i aktualną dokumentację projektową, która jest łatwa w czytaniu i nawigacji oraz ułatwia zrozumienie projektu przez zespół
- Nauczyłem się skutecznie komunikować z zespołem, klientami i użytkownikami.

C5. Wykorzystane materiały i bibliografia związana z realizacją projektu

1. Oficjalna dokumentacja technologii użytych w projekcie

- Vue.js - Oficjalna dokumentacja: https://vuejs.org/
- Laravel - Oficjalna dokumentacja: https://laravel.com/docs
- Laravel Reverb (WebSocket) - Oficjalna dokumentacja: https://laravel.com/docs/11.x/broadcasting
- Tailwind CSS – Oficjalna dokumentacja: https://tailwindcss.com/docs
- Pinia (zarządzanie stanem w Vue.js) – Oficjalna dokumentacja: https://pinia.vuejs.org/
- SQLite - Oficjalna dokumentacja: https://www.sqlite.org/docs.html
- Axios – Oficjalna dokumentacja: https://axios-http.com/

2. Narzędzia wykorzystywane do budowy aplikacji

- Visual Studio Code – dokumentacja i rozszerzenia: https://code.visualstudio.com/
- Docker – podręcznik użytkownika: https://docs.docker.com/
- GitHub – dokumentacja systemu kontroli wersji: https://docs.github.com/
- [2]qrcode-monkey – darmowy generator kodów QR: https://www.qrcode-monkey.com/

3. Artykuły i źródła internetowe

- Blog Vue.js - najlepsze praktyki w zarządzaniu stanem aplikacji: https://blog.vuejs.org/

- The WebSocket API - szczegółowy opis technologii WebSocket: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

- [1]Heurystyka Nielsena - dobre praktyki projektowania stron: https://www.damianrams.pl/heurystyki-nielsena/

- [3]IDOR - luka przy modyfikacji sekwencjonowanych identyfikatorów: https://portswigger.net/web-security/access-control/idor

- Markus Winand (2012). *SQL Performance Explained.*

- Robert C. Martin (2015). *Czysty kod. Podręcznik dobrego programisty*, tłum. Paweł Gonera. Wydawnictwo HELION.

- Chris Sanders (2017). *Praktyczna analiza pakietów. Wykorzystanie narzędzia Wireshark do rozwiązywania problemów związanych z siecią. Wydanie III*, tłum. Robert Górczyński. Wydawnictwo HELION.

4. Kursy i materiały multimedialne

- Codecourse - “Realtime with laravel reverb” kompletny poradnik implementacji technologii WebSocket: https://www.youtube.com/watch?v=WgirvczpvA8

- Azizdev - “Laravel reverb vps deployment” duża pomoc przy konfiguracji serwera Nginx + Reverb: https://www.youtube.com/watch?v=zuBTQOz8r-0

- Refactoring Guru - strona szczegółowo omawiająca najpopularniejsze wzorce projektowe i dobre praktyki: https://refactoring.guru/pl/design-patterns/catalog