SPRAWOZDANIE Z PROJEKTU PK4:

SYSTEM ZARZĄDZANIA TICKETAMI ZE ZINTEGROWANYM SERWEREM HTTP

AUTOR: JAKUB BODZIOCH, SEM.4 GR.2
JB305900@STUDENT.POLSL.PL

SPIS TREŚCI:

WPROWADZENIE I CEL PROJEKTU:

Projekt URWIS (User Request Wizard with Integrated Server) – System Zarządzania Ticketami został zaprojektowany jako aplikacja wspierająca zarządzanie zgłoszeniami w systemie helpdeskowym. W ramach projektu zaimplementowano architekturę opartą na kontrolerach i serwisach, umożliwiającą tworzenie, przeglądanie, filtrowanie oraz automatyczne obsługiwanie zgłoszeń użytkowników. System wspiera również uwierzytelnianie, sesje użytkowników, podstawowe logowanie działań oraz automatyczne odpowiedzi na bazie słów kluczowych.

Celem projektu było stworzenie modularnego systemu typu „ticket manager", który:

- umożliwia autoryzację i zarządzanie sesjami użytkowników (różne role),
- wspiera rejestrację i obsługę zgłoszeń (ticketów),
- automatyzuje odpowiedzi na podstawie słów kluczowych,
- rejestruje działania systemowe w prostym logu,
- umożliwia administratorom zarządzanie słownikami oraz zgłoszeniami,
- stanowi przykład poprawnie zaprojektowanego obiektowo systemu w C++.

Do realizacji warstwy interfejsu HTTP wykorzystano bibliotekę **Crow** – nowoczesny, lekki framework dla C++ inspirowany bibliotekami takimi jak Flask (Python) czy Express (Node.js). Crow umożliwia szybkie budowanie REST-owych (Representational State Transfer) API, wspiera routing, obsługę żądań, odpowiedzi oraz integrację z danymi JSON. Dzięki zastosowaniu tej biblioteki możliwe było stworzenie kompletnego backendu wyłącznie w języku C++, bez potrzeby integrowania go z innymi technologiami, takimi jak Python czy Node.js.

OPIS DZIAŁANIA PROGRAMU:

Program URWIS to aplikacja serwerowa działająca w środowisku HTTP, której głównym celem jest zarządzanie zgłoszeniami (ticketami) użytkowników. Użytkownik komunikuje się z systemem poprzez zapytania HTTP (np. z poziomu przeglądarki, Postmana lub innego klienta REST). Aplikacja reaguje na te żądania zgodnie z wcześniej zdefiniowanymi ścieżkami (endpointami), przetwarzając dane i zwracając odpowiedzi w formacie JSON.

Podstawowy przebieg działania wygląda następująco:

1. **Uruchomienie aplikacji (URWIS.cpp)** – główna funkcja konfiguruje router HTTP z pomocą biblioteki Crow i łączy ścieżki z odpowiednimi kontrolerami.
2. **Autoryzacja użytkownika (AuthController, AuthService)** – użytkownik loguje się przy użyciu loginu i hasła. Dane są weryfikowane, a po poprawnym logowaniu tworzona jest sesja z tokenem, zarządzana przez SessionManager.
3. **Obsługa zgłoszeń (TicketController, TicketService)** – użytkownicy mogą:
    a. dodawać nowe zgłoszenia,
    b. przeglądać własne zgłoszenia,
    c. filtrować je np. po statusie lub dacie.
4. **Odpowiedzi automatyczne (AutoResponder)** – jeśli w treści zgłoszenia występuje rozpoznane słowo kluczowe (zdefiniowane przez administratora), system może automatycznie odpowiedzieć na ticket.
5. **Zarządzanie słowami kluczowymi (KeywordController, KeywordService)** – administratorzy mogą tworzyć, edytować i usuwać reguły słów kluczowych.
6. **Logowanie (SimpleLogService)** – działania systemu są zapisywane do pliku logu.
7. **Zarządzanie użytkownikami i zgłoszeniami (AdminController)** – administrator może przeglądać wszystkich użytkowników, przypisywać ticketom statusy lub usuwać je.

Wszystkie dane użytkowników, zgłoszeń i sesji są zapisywane i ładowane z plików JSON (z pomocą *nlohmann::json*), co umożliwia prostą, plikową „bazę danych”.

DIAGRAM CZYNNOŚCI WYKONYWANYCH PO STRONIE UŻYTKOWNIKA:

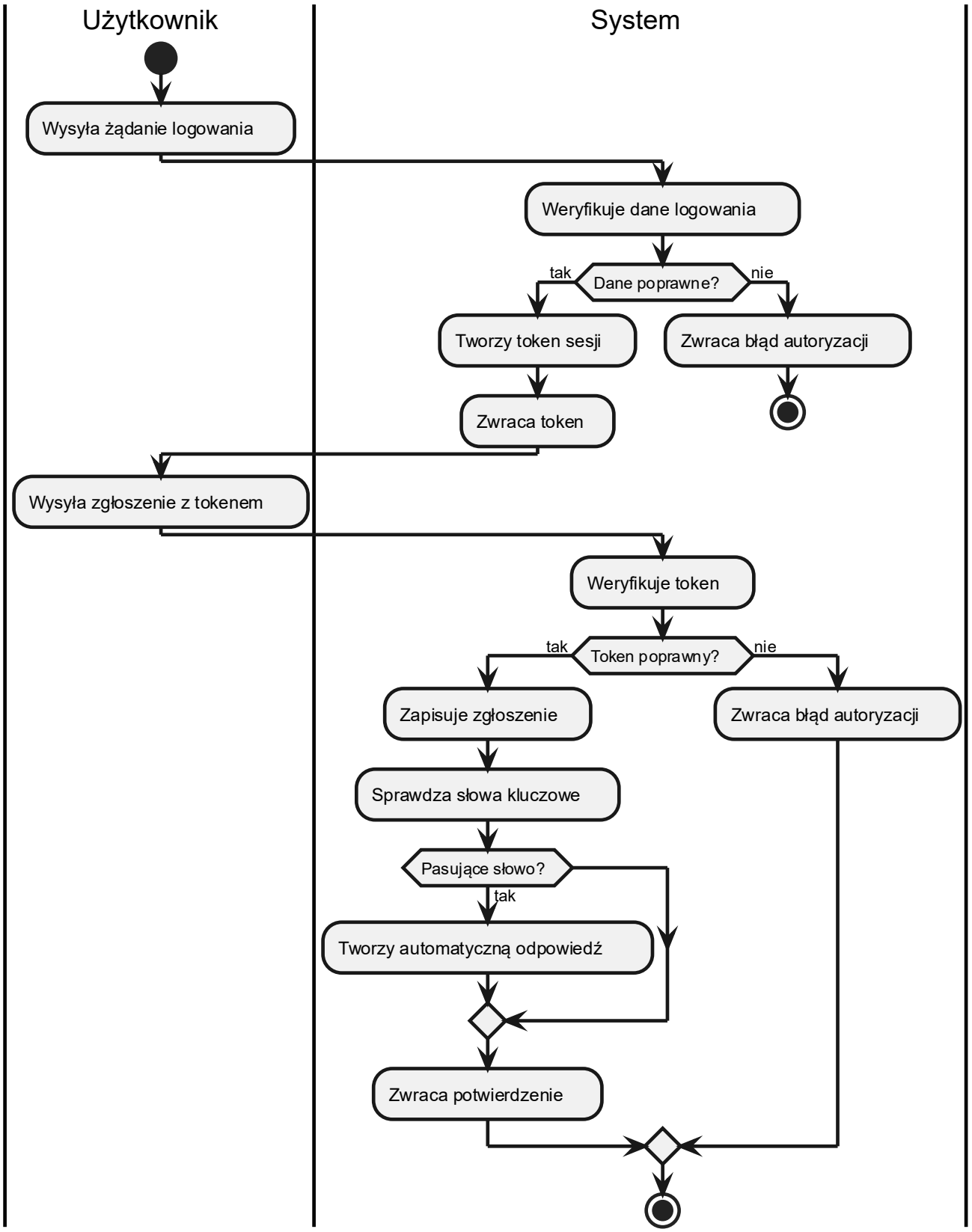


Diagram 1: czynności wykonywane po stronie użytkownika.

DIAGRAM CZYNNOŚCI WYKONYWANYCH PO STRONIE ADMINISTRATORA:

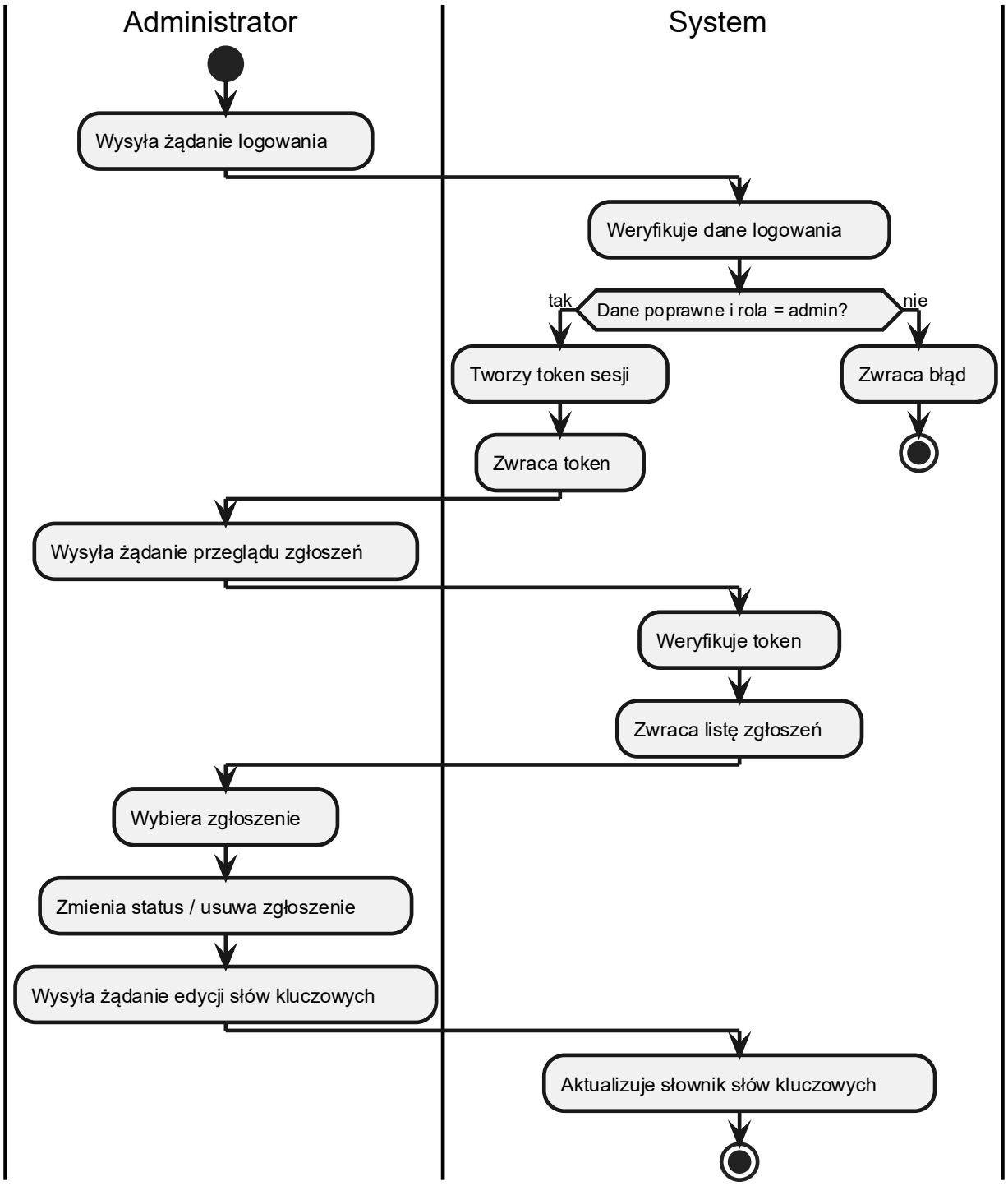
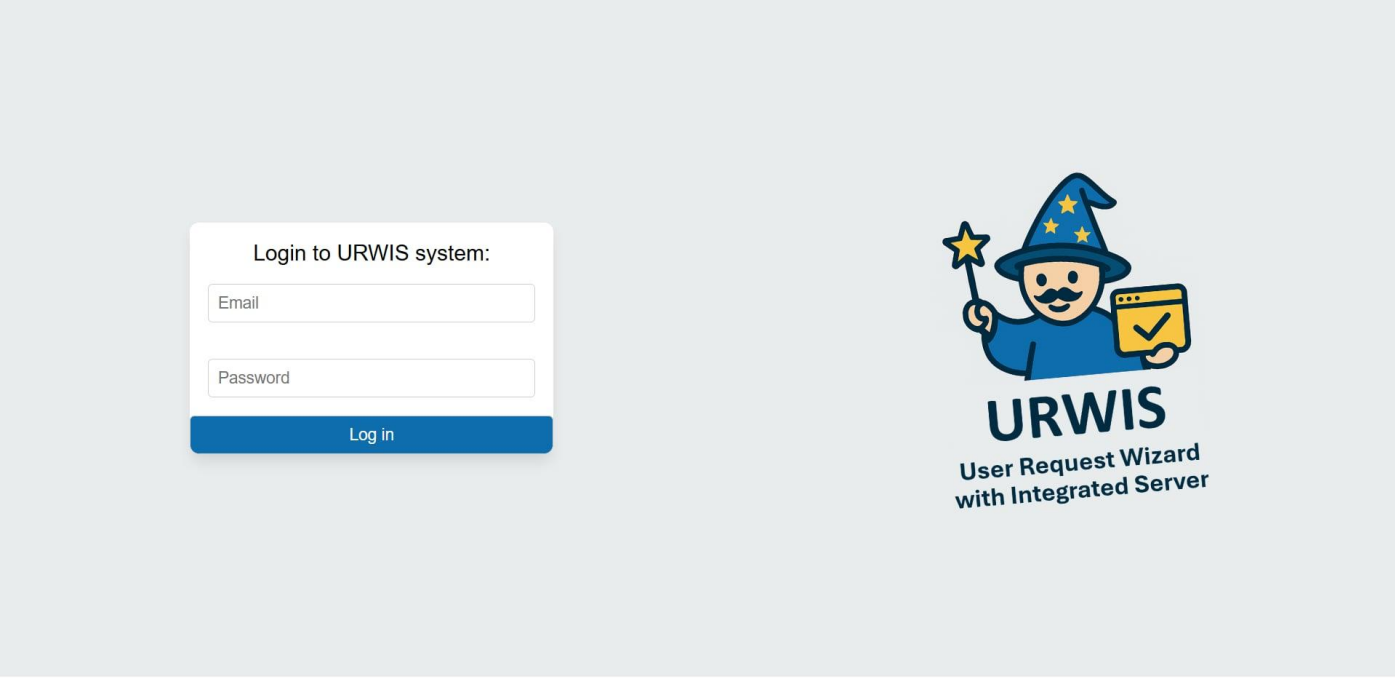


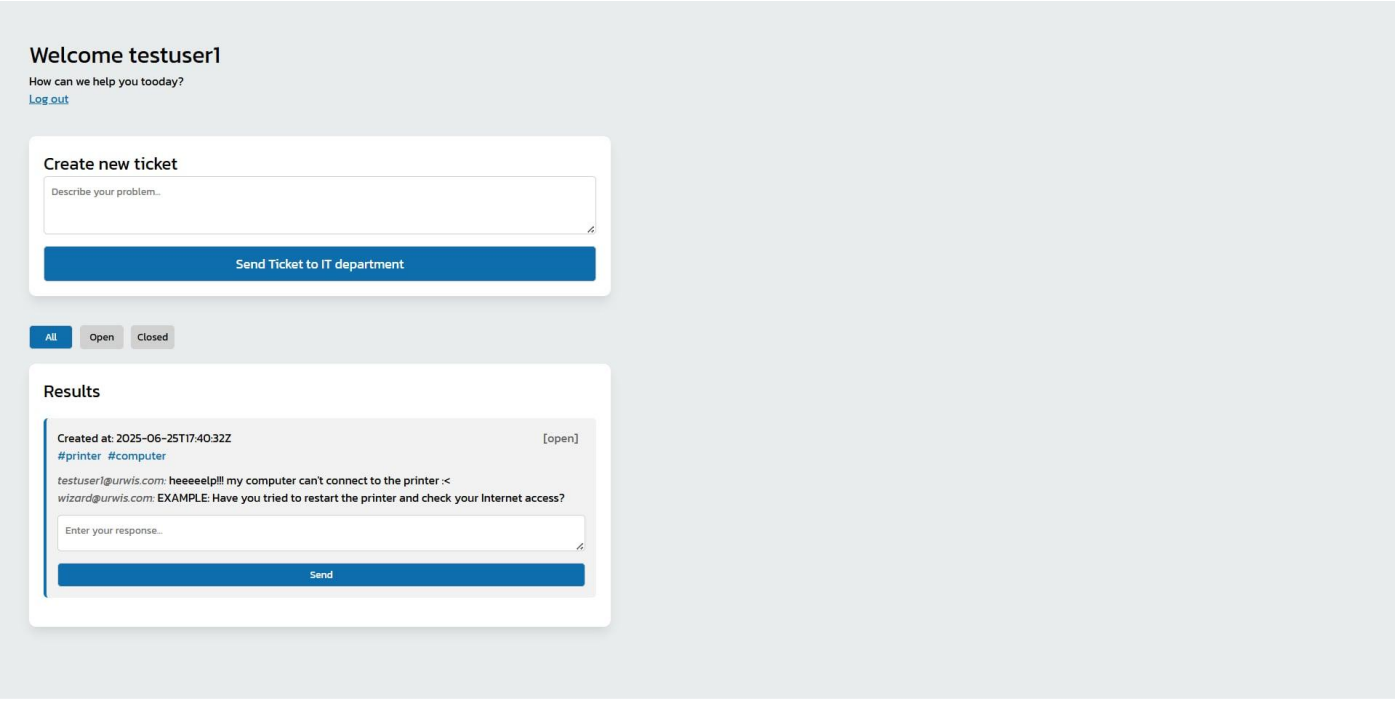
Diagram 2: czynności wykonywane po stronie administratora.

PREZENTACJA INTERFEJSU GRAFICZNEGO:

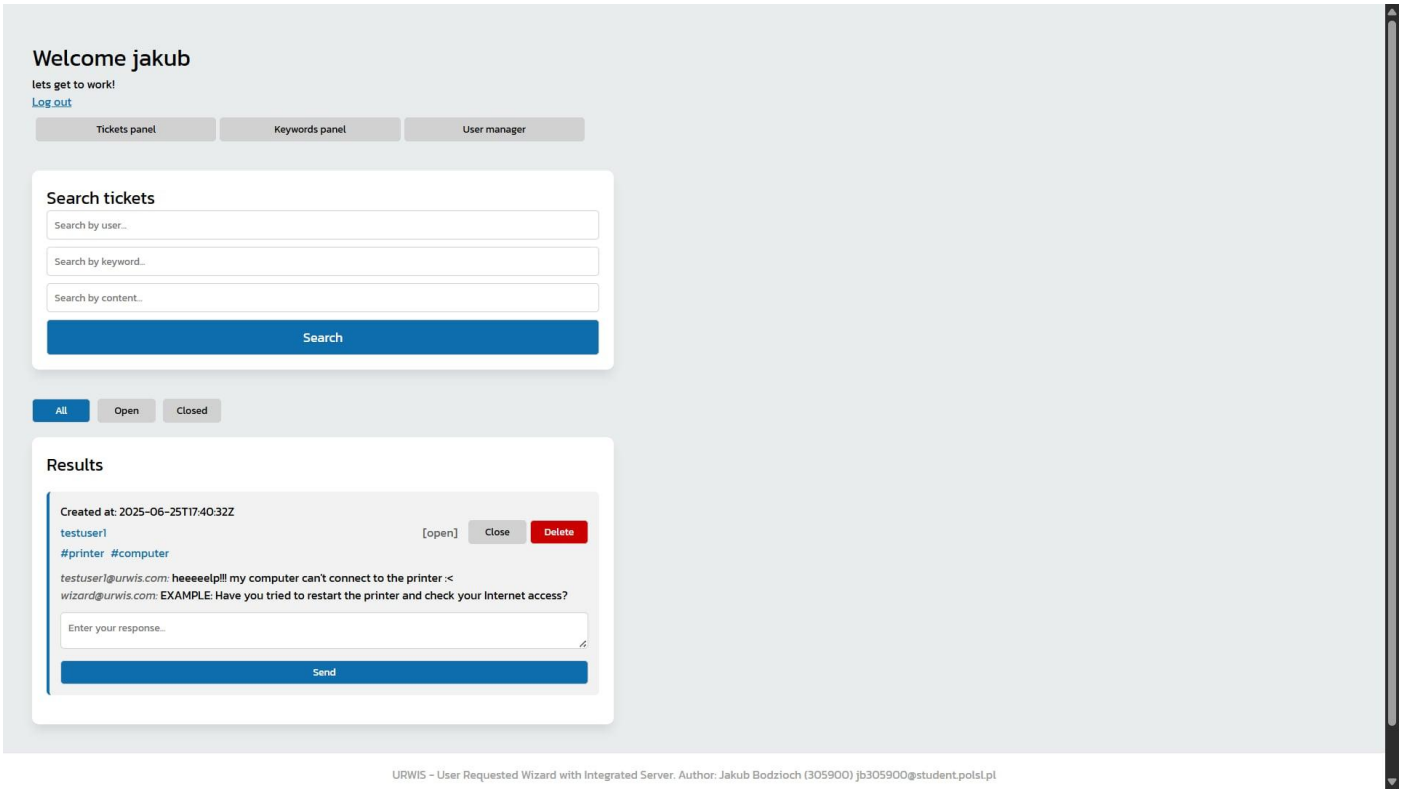
Interfejs użytkownika dostępny jest z poziomu przeglądarki internetowej. Został zaprojektowany w taki sposób, aby był skalowalny i dostosowany do urządzeń mobilnych, dzięki czemu system można wygodnie obsługiwać zarówno na komputerze, jak i na telefonie czy tablecie.



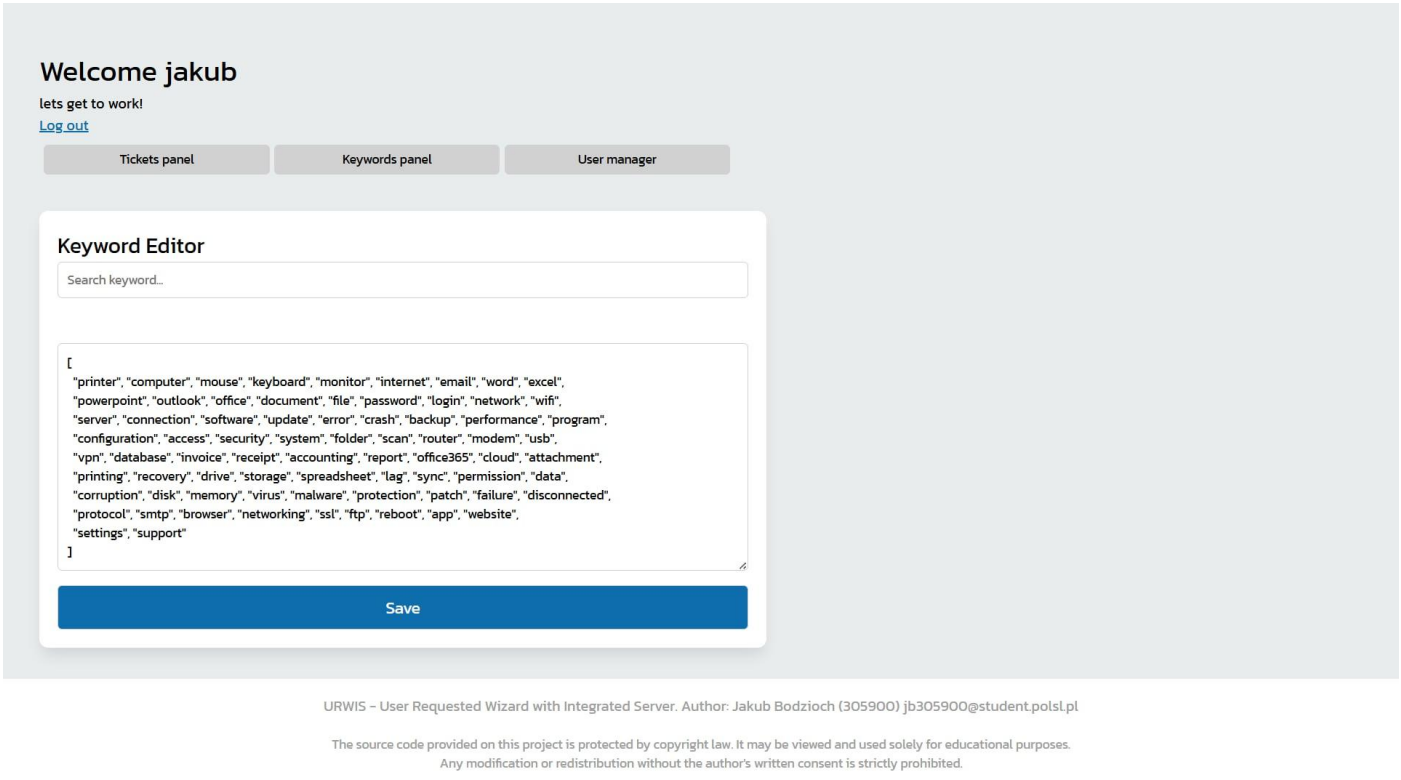
Interfejs 1: strona logowania do systemu.



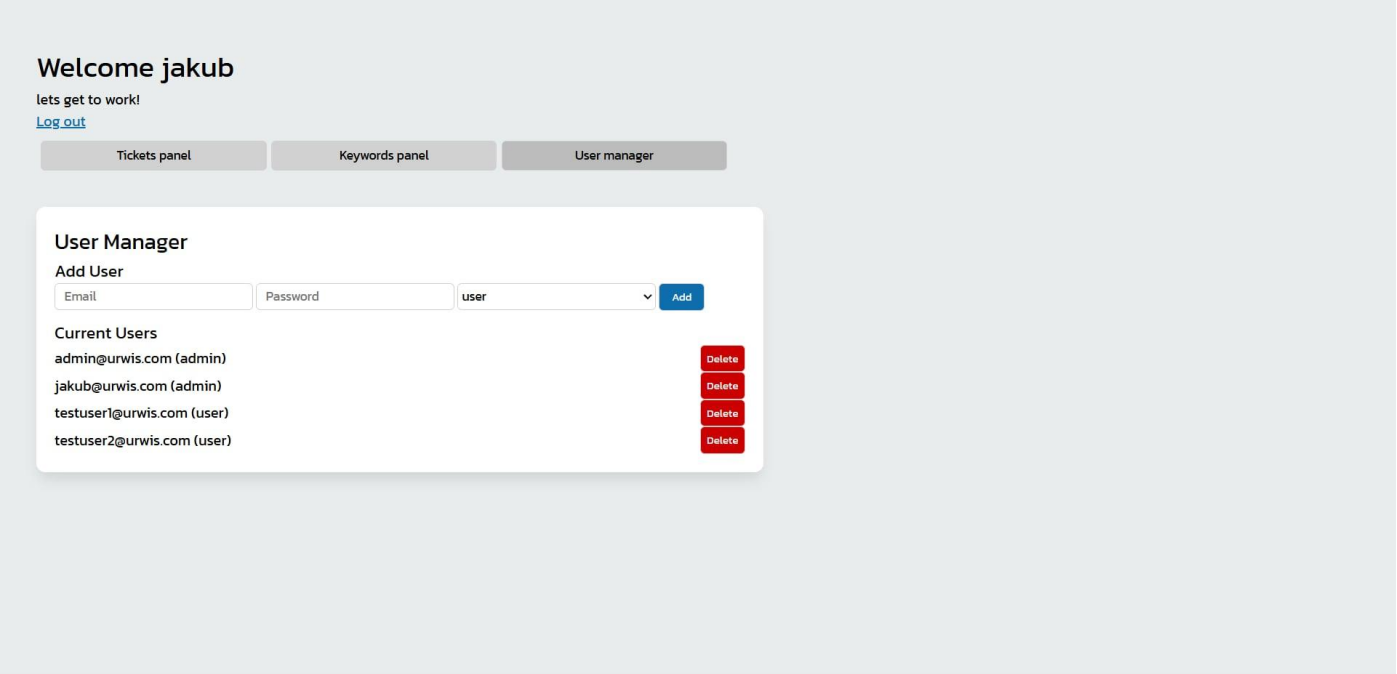
Interfejs 2: strona użytkownika roli user.



Interfejs 3: strona zarządzania zgłoszeniami użytkownika admin.



Interfejs 4: wbudowany edytor słów kluczowych.



Interfejs 5: strona zarządzania użytkownikami w systemie.

OMÓWIENIE WYBRANYCH FUNKCJI PORGRAMU:

AUTORYZACJA UŻYTKOWNIKÓW I SESJE:

System URWIS wykorzystuje mechanizm logowania oparty na autoryzacji przez ciasteczka (cookies). Po poprawnym zalogowaniu użytkownika, system generuje specjalny token sesji, który jest szyfrowany i odsyłany do klienta jako ciasteczko. Przy każdym kolejnym zapytaniu, klient automatycznie dołącza je do żądania, co pozwala systemowi rozpoznać użytkownika i jego uprawnienia.

Opis działania poszczególnych segmentów systemu logowania:

AuthController: Odbiera żądania logowania (/login), weryfikuje dane logowania i jeśli są poprawne, to ustawia ciasteczko z tokenem sesji.

AuthService: Odpowiada za: wczytanie danych użytkowników z pliku JSON, znalezienie użytkownika po loginie, sprawdzenie hasła (zwykłe porównanie ze zdeszyfrowaną wersją), utworzenie sesji.

CryptoService: Zamiast klasycznego hashowania jak SHA256, system używa: zakodowania danych sesji do JSON, następnie prostego szyfrowania z użyciem **XOR z kluczem** i kodowania Base64.

SessionManager: Obsługuje dekodowanie ciasteczek sesji: deszyfruje token (Base64 + XOR), odczytuje z niego login i rolę użytkownika, sprawdza, czy sesja jest poprawna.

Przebieg działania autoryzacji:

1. Użytkownik wysyła login i hasło (POST /login).
2. System sprawdza dane i tworzy zakodowany token sesji (login + rola).
3. Token trafia do ciasteczka w odpowiedzi HTTP.
4. Przy każdym zapytaniu, klient przesyła ciasteczko automatycznie.
5. System dekoduje token i wie, kim jest użytkownik i jaką ma rolę.

OBSŁUGA ZGŁOSZEŃ (TICKETY):

System URWIS umożliwia użytkownikom zgłaszanie problemów (ticketów), ich późniejsze przeglądanie, a administratorom, zarządzanie nimi. Cała logika podzielona jest na część kontrolerową (HTTP) i usługową (logika działania).

Opis działania poszczególnych segmentów systemu zgłoszeń:

TicketController odpowiada za obsługę żądań HTTP dotyczących zgłoszeń:

- **POST /tickets** – dodanie nowego zgłoszenia,
- **GET /tickets** – lista zgłoszeń użytkownika (lub wszystkich, jeśli admin),
- **DELETE /tickets/:id** – usuwanie zgłoszenia (admin),
- **POST /tickets/:id/status** – zmiana statusu (np. „otwarte”, „zamknięte”).

TicketService zajmuje się:

- walidacją danych ticketa,
- przypisaniem unikalnego ID,
- zapisem ticketa do pliku (tickets.json),
- filtrowaniem zgłoszeń dla konkretnego użytkownika lub roli,
- wczytywaniem ticketa po ID.

Przebieg dodawania zgłoszenia:

1. Użytkownik (z tokenem w ciasteczku) wysyła dane zgłoszenia.
2. System odczytuje token i identyfikuje użytkownika.
3. Tworzony jest nowy obiekt zgłoszenia z przypisanym ID.
4. Zgłoszenie zapisywane jest w pliku JSON.
5. Następuje automatyczne sprawdzenie treści przez AutoResponder.

AUTOMATYCZNE ODPOWIEDZI I SŁOWA KLUCZOWE:

Jedną z dodatkowych funkcji systemu URWIS jest możliwość **automatycznego odpowiadania na zgłoszenia** na podstawie słów kluczowych zawartych w treści. Jeśli treść zgłoszenia zawiera ustalony wcześniej wyraz/zwrot, system sam odsyła gotową odpowiedź.

AutoResponder to główna klasa realizująca mechanizm automatycznych odpowiedzi. Jej zadaniem jest:

- przeanalizowanie treści zgłoszenia (tytułu i opisu),
- wyszukanie pasujących słów kluczowych,
- jeśli znajdzie pasujące – wygenerowanie odpowiedzi.

KeywordService zarządza listą słów kluczowych i powiązanych odpowiedzi:

- odczytuje dane z pliku keywords.json,
- umożliwia dodawanie, edytowanie i usuwanie słów (np. przez admina),
- słowo kluczowe może zawierać kilka synonimów.

KeywordController udostępnia endpointy HTTP dla administratora do zarządzania słownikami:

- GET /keywords – pobiera listę wszystkich słów,
- POST /keywords – dodaje nowe słowo,
- PUT /keywords/:id – edycja,
- DELETE /keywords/:id – usunięcie.

Działanie systemu w praktyce:

1. Użytkownik dodaje zgłoszenie z treścią np. „Nie działa VPN”.
2. System po zapisaniu ticketa przekazuje go do AutoResponder.
3. AutoResponder analizuje tekst i wykrywa słowo „VPN”.
4. Znaleziona zostaje dopasowana odpowiedź i trafia do zgłoszenia

ELEMENTY Z ZAJĘĆ LABORATORYJNYCH I ICH WYKORZYSTANIE W PROJEKCIE:

BIBLIOTEKA <THREAD> – WIELOWĄTKOWOŚĆ:

Pozwala na uruchamianie funkcji w osobnych wątkach, niezależnie od głównego przepływu programu. Umożliwia to np. obsługę logów lub zadań tła bez blokowania serwera. Klasa SimpleLogService uruchamia osobny wątek do asynchronicznego zapisu logów, co poprawia wydajność i nie blokuje aplikacji.

Przykład użycia:

```
worker = std::thread(&SimpleLogService::run, this);
```

BIBLIOTEKA <FILESYSTEM> – OPERACJE NA PLIKACH I FOLDERACH:

Umożliwia sprawdzanie, tworzenie i zarządzanie strukturą plików/katalogów. Używana do upewnienia się, że katalogi z danymi istnieją przed zapisem. Biblioteka używana do operacji na plikach danych (np. tworzenie folderu logów).

Przykład użycia:

```
std::filesystem::create_directory("logs");
```

BIBLIOTEKA <REGEX> – WYRAŻENIA REGULARNE:

Służy do dopasowywania wzorców tekstowych. Umożliwia analizę treści zgłoszenia lub wyciąganie danych z formularzy. Wydobywanie wartości z treści POST (login, hasło itd.). Sprawdzanie, czy w zgłoszeniu występuje dane słowo kluczowe (ignorując wielkość liter).

Przykład użycia:

```
std::regex pattern(key + "=([^&]*)");
std::smatch match;
if (std::regex_search(body, match, pattern)) {
    return url_decode(match[1].str());
}
```

BIBLIOTEKA <RANGES> – PRZETWARZANIE KOLEKCJI W STYLU FUNKCYJNYM:

Biblioteka <ranges> pozwala na eleganczne i wydajne operacje na kolekcjach, np. transformacje i filtrowania bez jawnych pętli. Przekształcanie treści zgłoszenia do formy małymi literami, co ułatwia dopasowywanie słów kluczowych niezależnie od pisowni.

Przykład użycia:

```
ranges::transform(lower, lower.begin(), ::tolower);
```

PEŁNY DIAGRAM DZIAŁANIA PROGRAMU:

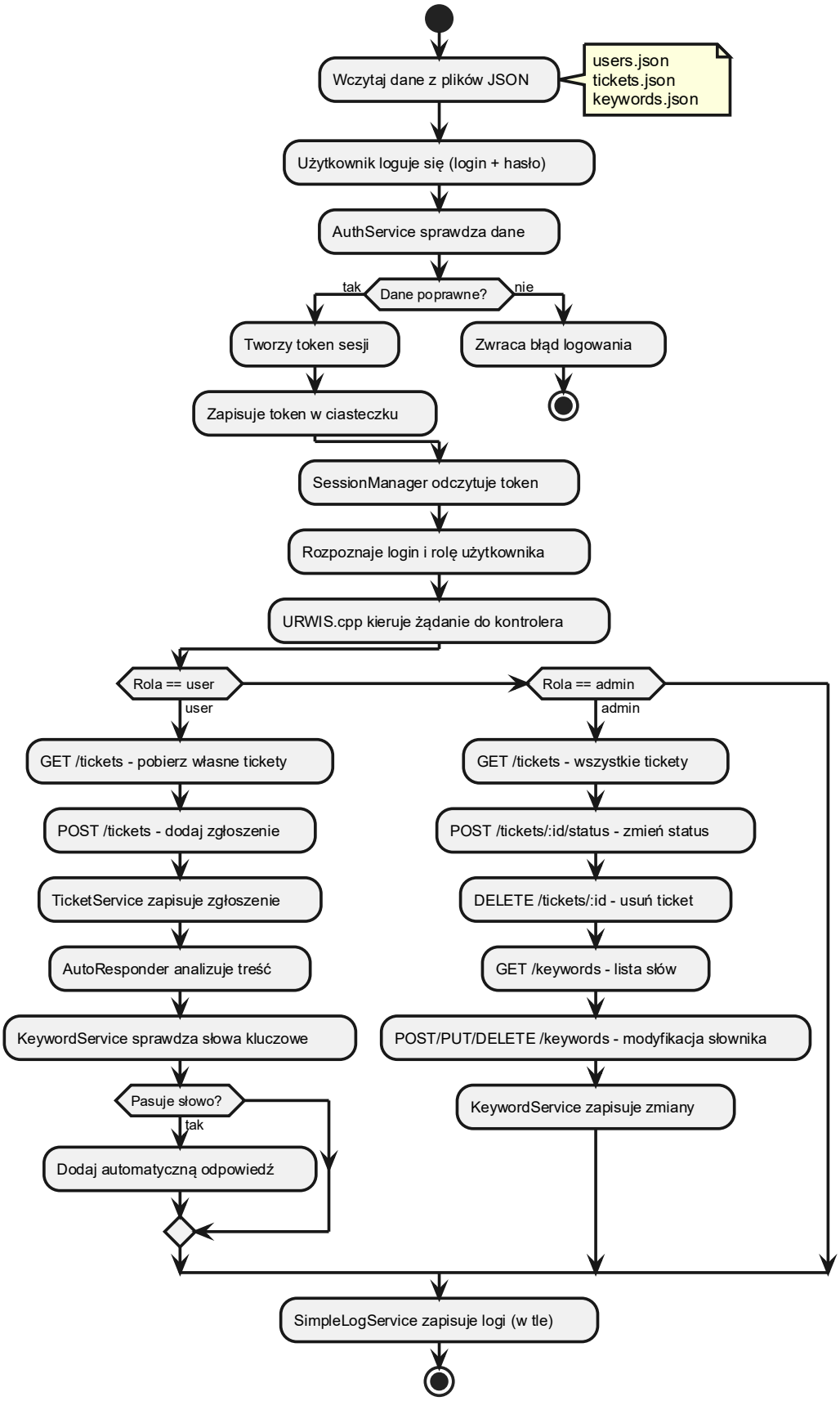


Diagram 3: uproszczony diagram działania całego programu.

DIAGRAM UML PREZENTUJĄCY KLASY PROJEKTU:

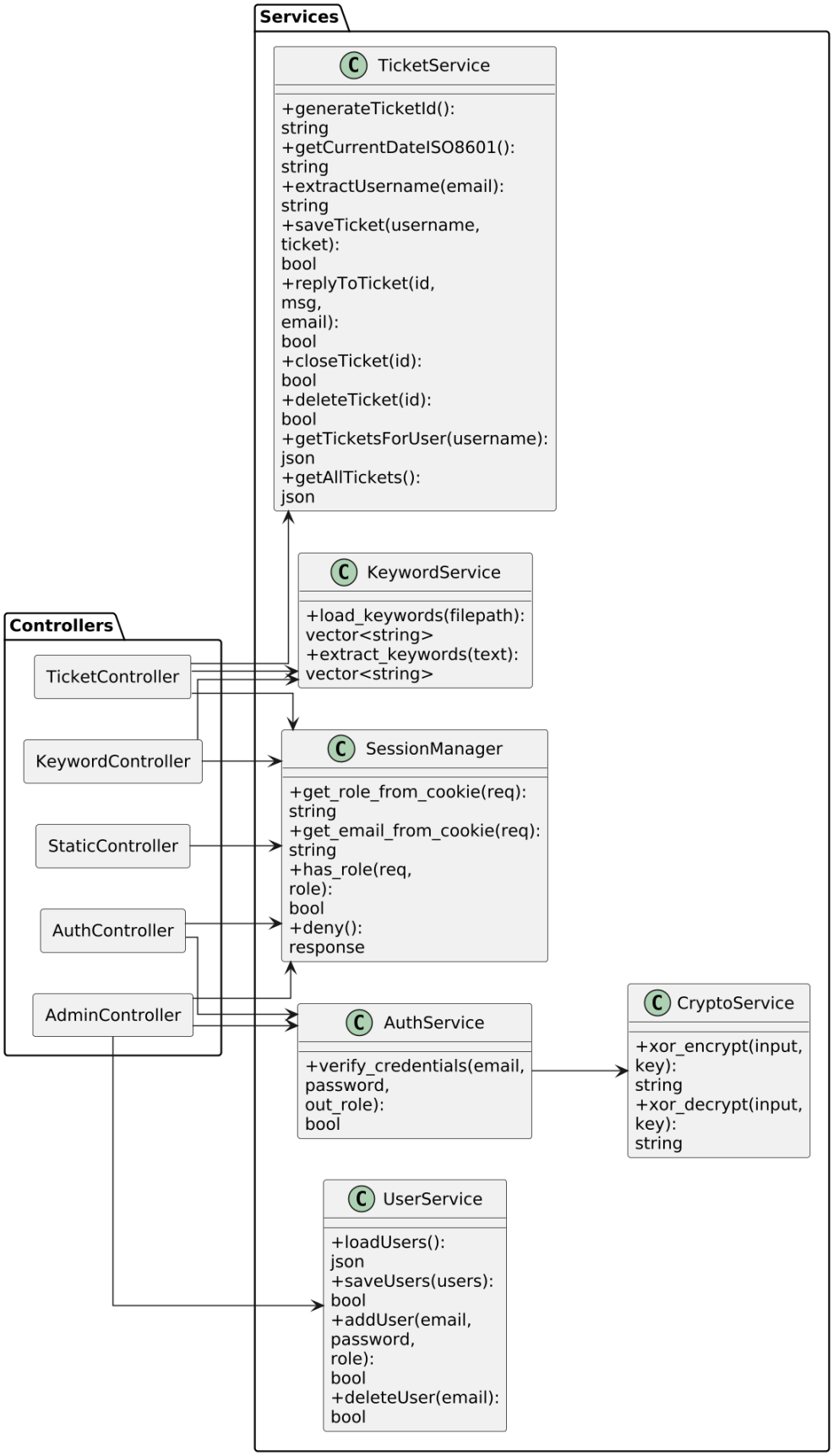


Diagram 4: diagram klas rozwinięty o połączenia kontrolerów.

TABELA KLAS PROJEKTU:

Klasa	**Opis**
AuthController	Obsługuje logowanie użytkowników, ustawianie ciasteczek i wylogowanie.
AdminController	Umożliwia administratorowi zarządzanie użytkownikami (dodawanie, usuwanie, listowanie).
TicketController	Obsługuje tworzenie, przeglądanie, aktualizowanie i usuwanie ticketów.
KeywordController	Udostępnia API do zarządzania słowami kluczowymi.
StaticController	Serwuje pliki statyczne i widoki frontendowe.
AuthService	Weryfikuje dane logowania na podstawie zaszyfrowanych danych z pliku.
SessionManager	Zarządza ciasteczkami sesji i dostępem na podstawie roli użytkownika.
UserService	Obsługuje operacje na użytkownikach w pliku auth_data.json.
CryptoService	Realizuje szyfrowanie i deszyfrowanie (XOR + Base64).
TicketService	Zajmuje się tworzeniem, zapisem i modyfikacją ticketów w systemie.
KeywordService	Wczytuje słowa kluczowe i dopasowuje je do treści zgłoszenia.
AutoResponder	Analizuje zgłoszenia i generuje automatyczne odpowiedzi na podstawie słów kluczowych.
SimpleLogService	Zapisuje zdarzenia systemowe do pliku logów w osobnym wątku.
URWIS	Punkt wejściowy aplikacji – uruchamia serwer HTTP i montuje wszystkie kontrolery.

TABELA WYKORZYSTANYCH BIBLIOTEK ZEWNĘTRZNYCH:

Biblioteka	**Opis**	**Link**
Crow	Lekki mikroframework HTTP dla C++ inspirowany Flaskiem – umożliwia tworzenie serwera i routingu REST.	https://github.com/CrowCpp/Crow
nlohmann/json	Biblioteka C++ do prostego przetwarzania danych JSON w stylu przypominającym język Python.	https://github.com/nlohmann/json

PROBLEMY NAPOTKANE PODCZAS REALIZACJI PORJEKTU:

Problem	**Opis i rozwiązanie**
Zabezpieczenie danych logowania	Zaprojektowanie mechanizmu szyfrowania haseł (XOR + Base64), zamiast klasycznego hashowania. Wymagało ręcznego testowania poprawności i bezpieczeństwa przechowywanych danych.
Routing w bibliotece Crow	Problemy z odpowiednim mapowaniem ścieżek HTTP do kontrolerów i metod. Wymagało spójnego projektowania endpointów i dostosowania do dokumentacji Crow.
Projekt interfejsu użytkownika	Zaprojektowanie prostego, przejrzystego i responsywnego UI działającego także na urządzeniach mobilnych – mimo braku rozbudowanego frontendu.
Obsługa znaków narodowych (kodowanie UTF-8)	Problemy z wyświetlaniem polskich znaków diakrytycznych w zgłoszeniach i loginach. Próbowano wymusić UTF-8 w plikach, nagłówkach HTTP i dyrektywach kompilatora, ale finalnie interfejs przestawiono na język angielski.
Brak natywnej ochrony sesji w Crow	Crow nie oferuje wbudowanego systemu sesji – konieczne było tymczasowe użycie localStorage po stronie przeglądarki i własne ciasteczkowe tokeny. System sprawdza dostęp przy każdym żądaniu.

Powyższe zestawienie przedstawia najważniejsze problemy napotkane podczas realizacji projektu. Część z nich – jak również wiele drobniejszych trudności – miała realny wpływ na kierunek rozwoju aplikacji oraz decyzje projektowe. To właśnie one zainspirowały nazwę systemu „URWIS”, która z czasem zyskała również swoje rozwinięcie: User Request Wizard with Integrated Server – trafne i nieco żartobliwe podsumowanie charakteru oraz funkcji całego projektu.

WNIOSKI:

Realizacja projektu URWIS pozwoliła nie tylko na stworzenie w pełni funkcjonalnego systemu do zarządzania zgłoszeniami, ale również na praktyczne przećwiczenie wielu zagadnień związanych z programowaniem obiektowym, pracą z serwerem HTTP oraz przetwarzaniem danych w formacie JSON. Zastosowanie architektury kontroler–serwis umożliwiło czytelny podział odpowiedzialności i ułatwiło dalszy rozwój kodu.

Podczas pracy nad projektem pojawiło się wiele wyzwań technicznych, od implementacji mechanizmu autoryzacji, przez obsługę tras w bibliotece Crow, aż po kwestie związane z bezpieczeństwem i kodowaniem znaków. Każdy z tych problemów wymagał indywidualnego podejścia, a ich rozwiązanie znacząco przyczyniło się do podniesienia jakości finalnej wersji aplikacji.

Na uwagę zasługuje również fakt, że projekt został zrealizowany w całości w języku C++, bez potrzeby używania dodatkowych technologii frontendowych, co pokazało, że z pomocą nowoczesnych bibliotek (takich jak Crow czy nlohmann/json) możliwe jest budowanie nowoczesnych, responsywnych systemów webowych nawet w tym języku.

URWIS to nie tylko skrót, to także odzwierciedlenie charakteru tego projektu: system nieoczywisty, nieco uporczywy w projektowaniu, ale ostatecznie skuteczny i gotowy do działania w realnych scenariuszach.