

AI2	Dokumentacja projektu
Autor	Dawid Olko 125148
Kierunek, rok	Informatyka, IV rok, st. stacjonarne (3,5-l)
Temat projektu	<i>Projekt i implementacja aplikacji „Systemu zarządzania zadaniami w firmie informatycznej”</i>

Spis treści

1. WSTĘP.....	3
1.1. Cel projektu	3
1.2. Zakres funkcjonalny.....	3
1.3. Fragment rzeczywistości.....	3
2. NARZĘDZIA I TECHNOLOGIE.....	4
2.1. Języki programowania	4
2.2. Frameworki	4
2.3. Baza danych	4
2.4. Biblioteki	4
3. BAZA DANYCH	5
3.1. Schemat ERD	5
3.2. Opis tabel	5
4. ENDPOINT'Y API.....	7
4.1. Lista endpoint'ów.....	7
4.2. Dokumentacja Swagger.....	8
4.3. Zgodność z poziomem 2 Richardsona	8
5. REALIZACJA SORTOWANIA	9
5.1. Kod backendu.....	9
5.2. Opis działania	9
5.3. Zrzuty ekranu	10
6. REALIZACJA FILTROWANIA	11
6.1. Kod backendu.....	11
6.2. Opis działania	12
6.3. Walidacja filtrów.....	13
6.4. Zrzuty ekranu	14
7. REALIZACJA WYSZUKIWANIA.....	15
7.1. Kod backendu.....	15
7.2. Opis działania	15

7.3. Zrzuty ekranu	16
8. REALIZACJA PAGINACJI.....	17
8.1. Kod backendu.....	17
8.2. Opis działania	17
8.3. Zrzuty ekranu	18
8.4. Kod filtrowania, sortowania, paginacji po stronie frontendu.....	20
9. PODSUMOWANIE	28

1. WSTĘP

1.1. Cel projektu

Celem mojego projektu było stworzenie systemu zarządzania zgłoszeniami IT (nazwa: IT Help Desk System), który umożliwiał będzie użytkownikom zgłaszać problemy techniczne oraz ich efektywne rozwiązywanie.

1.2. Zakres funkcjonalny

Aplikacja obejmuje następujące funkcjonalności:

- Tworzenie zgłoszeń IT przez użytkowników
- Przypisywanie zgłoszeń do techników
- Zmiana statusu zgłoszeń (Otwarte, W trakcie, Rozwiążane, Zamknięte)
- Ustawianie priorytetów zgłoszeń (Niski, Średni, Wysoki, Krytyczny)
- Kategoryzacja zgłoszeń (Sprzęt, Oprogramowanie, Sieć, Dostęp, Inne)
- Dodawanie komentarzy do zgłoszeń
- System historii zmian

1.3. Fragment rzeczywistości

System modeluje rzeczywisty proces obsługi zgłoszeń IT w firmie:

1. **Użytkownik** zgłasza problem techniczny (np. "Nie działa drukarka")
2. **Administrator** przypisuje zgłoszenie do odpowiedniego **technika**
3. **Technik** pracuje nad rozwiązyaniem problemu i zmienia status
4. **System** automatycznie śledzi czas.
5. Po rozwiązaniu zgłoszenie jest zamknięte

2. NARZĘDZIA I TECHNOLOGIE

2.1. Języki programowania

- **C# 12** - backend REST API
- **TypeScript 5** - frontend
- **SQL** - zapytania do bazy danych

2.2. Frameworki

Backend:

- **ASP.NET Core 9.0** - framework webowy
- **Entity Framework Core 9.0** - ORM do komunikacji z bazą danych

Frontend:

- **Vue.js 3.5** - framework JavaScript (SPA)
- **Vite 6.0** - bundler i dev server
- **Pinia** - zarządzanie stanem aplikacji
- **Vue Router** - routing w SPA

2.3. Baza danych

- **MySQL 8.0** - relacyjna baza danych
- **Entity Framework Core** - Code First approach

2.4. Biblioteki

Backend:

- **Swashbuckle.AspNetCore 7.2.0** - generowanie dokumentacji Swagger/OpenAPI
- **Microsoft.EntityFrameworkCore.Sqlite 9.0** - driver SQLite
- **System.ComponentModel.DataAnnotations** - walidacja danych

Frontend:

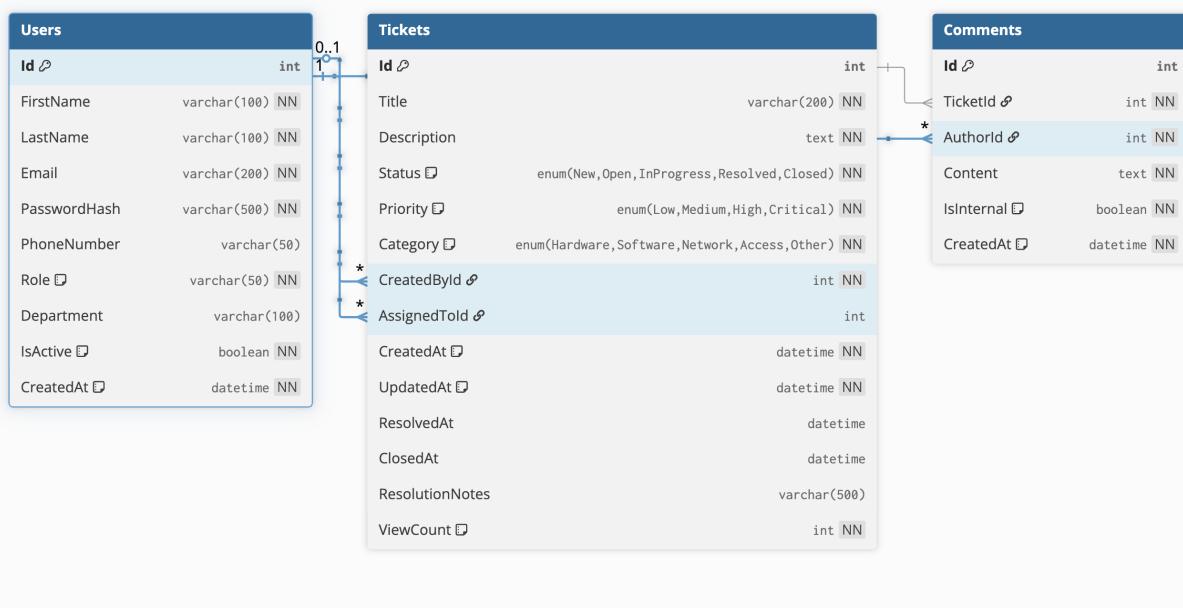
- **Axios 1.7.9** - komunikacja HTTP z API
- **Vue Router 4.5** - routing
- **Pinia 2.3** - state management
- **Heroicons** - ikony UI

2.5. Narzędzia deweloperskie

- **Visual Studio Code** - IDE
- **Swagger UI** - testowanie API
- **.NET CLI** - narzędzia .NET
- **npm** - menedżer pakietów

3. BAZA DANYCH

3.1. Schemat ERD



3.2. Opis tabel

Tabela: Users Przechowuje użytkowników systemu (pracownicy, technicy i administratorzy).

Pola: Id - unikalny identyfikator (Primary Key) FirstName - imię użytkownika (max 100 znaków, wymagane) LastName - nazwisko użytkownika (max 100 znaków, wymagane) Email - adres email (max 200 znaków, unikalny, wymagany) PasswordHash - zahaszowane hasło BCrypt (max 500 znaków, wymagane) PhoneNumber - numer telefonu (max 50 znaków, opcjonalny) Role - rola użytkownika: User, Technician, Admin (max 50 znaków, wymagane, domyślnie "User") Department - dział/wydział (max 100 znaków, opcjonalny) IsActive - czy konto aktywne (boolean, domyślnie true) CreatedAt - data utworzenia konta (DateTime, UTC)

Relacje: User → Tickets (1:N jako CreatedBy) - użytkownik może utworzyć wiele zgłoszeń
 User → Tickets (1:N jako AssignedTo) - technik może mieć przypisane wiele zgłoszeń
 User → Comments (1:N) - użytkownik może dodać wiele komentarzy

Tabela: Tickets Przechowuje zgłoszenia IT.

Pola: Id - unikalny identyfikator zgłoszenia (Primary Key) Title - tytuł zgłoszenia (max 200 znaków, wymagane) Description - szczegółowy opis problemu (text, wymagane) Status - status zgłoszenia: New, Open, InProgress, Resolved, Closed (enum, domyślnie "New") Priority - priorytet: Low, Medium, High, Critical (enum, domyślnie "Medium") Category - kategoria: Hardware, Software, Network, Access, Other (enum, domyślnie "Other") CreatedById - ID użytkownika który utworzył zgłoszenie (Foreign Key → Users.Id, wymagane) AssignedToId - ID przypisanego technika (Foreign Key → Users.Id, nullable) CreatedAt - data utworzenia (DateTime, UTC) UpdatedAt - data ostatniej aktualizacji (DateTime, UTC) ResolvedAt - data rozwiązania (DateTime, nullable) ClosedAt - data zamknięcia (DateTime, nullable) ResolutionNotes - notatki rozwiązania (max 500 znaków, opcjonalny) ViewCount - liczba wyświetleń (int, domyślnie 0)

Relacje: Ticket → User (N:1 jako CreatedBy) - zgłoszenie ma jednego twórcę Ticket → User (N:1 jako AssignedTo) - zgłoszenie może mieć przypisanego jednego technika Ticket → Comments (1:N) - zgłoszenie może mieć wiele komentarzy

SLA (Service Level Agreement): Critical: 4 godziny High: 24 godziny Medium: 72 godziny Low: 168 godzin (7 dni)

Tabela: Comments Przechowuje komentarze do zgłoszeń.

Pola: Id - unikalny identyfikator (Primary Key) TicketId - ID zgłoszenia (Foreign Key → Tickets.Id, wymagane) AuthorId - ID użytkownika który dodał komentarz (Foreign Key → Users.Id, wymagane) Content - treść komentarza (text, wymagane) IsInternal - czy komentarz wewnętrzny (boolean, domyślnie false) true = widoczny tylko dla techników/adminów false = widoczny dla wszystkich CreatedAt - data dodania (DateTime, UTC)

Relacje: Comment → Ticket (N:1) - komentarz należy do jednego zgłoszenia Comment → User (N:1) - komentarz ma jednego autora

4. ENDPOINT'Y API

4.1. Lista endpoint'ów

System implementuje 15 endpointów REST API zgodnych z poziomem 2 modelu dojrzałości Richardsona:

- POST /api/auth/register - rejestracja nowego użytkownika
- POST /api/auth/login - logowanie (zwraca JWT token)
- Tickets (Zgłoszenia) - Wymagają autoryzacji
- GET /api/tickets - pobranie listy zgłoszeń (z SFWP: Sort, Filter, Search, Pagination) Filtry: status, priority, category, assignedToId, createdById, isOverdue Sortowanie: id, title, status, priority, category, createdAt, updatedAt, viewCount Wyszukiwanie: title, description, creator name/email, assignee name Paginacja: page, pageSize
- GET /api/tickets/{id} - pobranie szczegółów zgłoszenia (z komentarzami)
- POST /api/tickets - utworzenie nowego zgłoszenia
- PUT /api/tickets/{id} - aktualizacja zgłoszenia
- DELETE /api/tickets/{id} - usunięcie zgłoszenia (tylko Admin)
- POST /api/tickets/{id}/comments - dodanie komentarza do zgłoszenia
- GET /api/tickets/statistics - statystyki dashboard (liczba zgłoszeń według statusu, priorytetu, kategorii)
- Users (Użytkownicy) - Wymagają autoryzacji
- GET /api/users - pobranie listy użytkowników (Admin/Technician) Filtr: role (User, Technician, Admin)
- GET /api/users/technicians - pobranie listy techników (do przypisywania zgłoszeń)
- GET /api/users/{id} - pobranie szczegółów użytkownika
- POST /api/users - utworzenie nowego użytkownika (tylko Admin)
- PUT /api/users/{id} - aktualizacja użytkownika (tylko Admin)
- PUT /api/users/{id}/role - zmiana roli użytkownika (tylko Admin)
- DELETE /api/users/{id} - soft delete użytkownika (IsActive = false) (tylko Admin)

4.2. Dokumentacja Swagger

The screenshot shows the IT Help Desk API v1 documentation generated by Swagger. It includes sections for Tickets and Users, each with a list of HTTP methods and their corresponding URLs and descriptions.

Tickets

- GET /api/Tickets Get all tickets with SFWP
- POST /api/Tickets Create new ticket
- GET /api/Tickets/{id} Get ticket by ID
- PUT /api/Tickets/{id} Update ticket
- DELETE /api/Tickets/{id} Delete ticket
- POST /api/Tickets/{id}/comments Add comment
- GET /api/Tickets/statistics Get statistics

Users

- GET /api/Users Get all users
- GET /api/Users/technicians Get technicians
- GET /api/Users/{id} Get user by ID

4.3. Zgodność z poziomem 2 Richardsoна

System spełnia wymagania Level 2 - HTTP Verbs modelu dojrzałości Richardsoна:

Zasoby (Resources) - każdy endpoint reprezentuje zasób:

- /api/tickets - zasób zgłoszeń
- /api/users - zasób użytkowników

Metody HTTP - poprawne użycie czasowników:

- GET - pobieranie danych (idempotentne)
- POST - tworzenie nowych zasobów
- PUT - aktualizacja zasobów (idempotentne)
- DELETE - usuwanie zasobów (idempotentne)

Kody statusów HTTP - poprawne użycie:

- 200 OK - pomyślne pobranie/aktualizacja
- 201 Created - pomyślne utworzenie zasobu
- 204 No Content - pomyślne usunięcie
- 400 Bad Request - błąd walidacji
- 404 Not Found - zasób nie istnieje

URI zasobów - logiczna struktura:

- /api/tickets/{id} - konkretny zasób
- /api/tickets - kolekcja zasobów

5. REALIZACJA SORTOWANIA

5.1. Kod backendu

Sortowanie jest implementowane w kontrolerze TicketsController.cs w metodzie ApplySorting():

```
private IQueryables<Ticket> ApplySorting(IQueryables<Ticket> query, string sortBy, string sortOrder)
{
    var isDescending = sortOrder.ToLower() == "desc";

    Expression<Func<Ticket, object>> sortExpression = sortBy.ToLower() switch
    {
        "id" => t => t.Id,
        "title" => t => t.Title,
        "status" => t => t.Status,
        "priority" => t => t.Priority,
        "category" => t => t.Category,
        "createdat" => t => t.CreatedAt,
        "updatedat" => t => t.UpdatedAt,
        "viewcount" => t => t.ViewCount,
        _ => t => t.CreatedAt
    };

    return isDescending
        ? query.OrderByDescending(sortExpression)
        : query.OrderBy(sortExpression);
}
```

5.2. Opis działania

Sortowanie w systemie działa w następujący sposób:

Domyślnie, jeśli parametr sortBy jest pusty, zgłoszenia są sortowane malejąco po dacie utworzenia (CreatedAt DESC). Kierunek sortowania określa parametr sortOrder, który może przyjąć wartość asc (rosnąco - ascending) lub desc (malejąco - descending).

System umożliwia sortowanie po następujących polach: id (ID zgłoszenia), title (tytuł zgłoszenia), status (status - Open, InProgress, Resolved, Closed), priority (priorytet - Low, Medium, High, Critical), category (kategoria), createdAt (data utworzenia).

W przypadku podania nieprawidłowej wartości sortBy, system automatycznie zastosuje domyślne sortowanie malejące po dacie utworzenia.

5.3. Zrzuty ekranu

The screenshot shows a web application titled "All Tickets". At the top, there is a search bar with placeholder text "Search tickets by title, description, or user...". Below it is a "FILTERS & SORTING" section with dropdown menus for Status (Open), Priority (Low), Category (Hardware), Assigned To (All Technicians), and a checkbox for "Overdue Tickets". The status bar indicates "Showing 9 of 9 tickets". The main table lists nine tickets, each with an ID, title, status, priority, category, creation date, and a "VIEW" button. The titles of the tickets include "Mysz podwójnie klika", "Popękane słuchawki", "Głośniki nie działają", "Problemy z połączeniem Bluetooth", "Port USB nie działa", and "Czcionki wyglądają rozmażane".

The screenshot shows the Swagger UI for an API endpoint at `http://localhost:5000/swagger/index.html`. It includes sections for "Responses", "Request URL", and "Server response". The "Responses" section contains a "Curl" command to execute a GET request to `http://localhost:5000/api/Tickets?Page=1&Status=Open&Priority=High&Category=Hardware`. The "Request URL" section shows the same URL. The "Server response" section displays a successful 200 status code response with a JSON payload. The JSON response body is as follows:

```

{
  "items": [
    {
      "id": 51,
      "title": "Niebieskie ekran przy startie",
      "description": "B500 z kodem błędu 0x0000007B pojawia się co kilka uruchomień. Czasem trzeba 3-4 restartów.",
      "status": "Open",
      "priority": "High",
      "category": "Hardware",
      "createdBy": 1,
      "full_name": "Paweł Wojciechowski",
      "email": "pawel.wojciechowski@firma.pl",
      "role": "User",
      "department": "Sprzedaz"
    }
  ],
  "assignedTo": [
    {
      "id": 13,
      "full_name": "Grzegorz Woźnicki",
      "email": "grzegorz.woznak@firma.pl",
      "role": "Technician",
      "department": "Zespół Sieciowy"
    }
  ],
  "createdAt": "2025-09-23T08:25:13.856319",
  "updatedAt": "2025-09-23T08:25:13.856319"
}
  
```

Przykładowy Request URL:

`http://localhost:5000/api/tickets?sortBy=priority&sortOrder=desc`

6. REALIZACJA FILTROWANIA

6.1. Kod backendu

Filtrowanie jest implementowane w kontrolerze TicketsController.cs w metodzie GetTickets():

```
if (parameters.Status.HasValue)
{
    query = query.Where(t => t.Status == parameters.Status.Value);
}

if (parameters.Priority.HasValue)
{
    query = query.Where(t => t.Priority == parameters.Priority.Value);
}

if (parameters.Category.HasValue)
{
    query = query.Where(t => t.Category == parameters.Category.Value);
}

if (parameters.AssignedTold.HasValue)
{
    var assigneeExists = await _context.Users.AnyAsync(u => u.Id == parameters.AssignedTold.Value);
    if (!assigneeExists)
    {
        return BadRequest(new
        {
            message = "Assigned user not found",
            parameter = "AssignedTold",
            value = parameters.AssignedTold.Value
        });
    }
    query = query.Where(t => t.AssignedTold == parameters.AssignedTold.Value);
}

if (parameters.CreatedById.HasValue)
{
    var creatorExists = await _context.Users.AnyAsync(u => u.Id == parameters.CreatedById.Value);
    if (!creatorExists)
    {
        return BadRequest(new
        {
            message = "Creator user not found",
            parameter = "CreatedById",
            value = parameters.CreatedById.Value
        });
    }
    query = query.Where(t => t.CreatedById == parameters.CreatedById.Value);
}

if (parameters.IsOverdue.HasValue && parameters.IsOverdue.Value)
{
    var now = DateTime.UtcNow;
    query = query.Where(t =>
        t.Status != TicketStatus.Resolved &&
        t.Status != TicketStatus.Closed &&
        t.CreateDate < now
    );
}
```

```

        (
            (t.Priority == TicketPriority.Critical && t.CreatedAt.AddHours(4) < now) ||
            (t.Priority == TicketPriority.High && t.CreatedAt.AddHours(24) < now) ||
            (t.Priority == TicketPriority.Medium && t.CreatedAt.AddHours(72) < now) ||
            (t.Priority == TicketPriority.Low && t.CreatedAt.AddHours(168) < now)
        )
    );
}

if (!string.IsNullOrWhiteSpace(parameters.Search))
{
    var searchLower = parameters.Search.ToLower();
    query = query.Where(t =>
        t.Title.ToLower().Contains(searchLower) ||
        t.Description.ToLower().Contains(searchLower) ||
        t.CreatedBy.FirstName.ToLower().Contains(searchLower) ||
        t.CreatedBy.LastName.ToLower().Contains(searchLower) ||
        t.CreatedBy.Email.ToLower().Contains(searchLower) ||
        (t.AssignedTo != null &&
            t.AssignedTo.FirstName.ToLower().Contains(searchLower) ||
            t.AssignedTo.LastName.ToLower().Contains(searchLower)
        ))
};
}

```

6.2. Opis działania

System obsługuje 6 typów filtrów:

Status (status) - filtrowanie po statusie zgłoszenia:

- Open - otwarte
- InProgress - w trakcie realizacji
- Resolved - rozwiążane
- Closed - zamknięte

Priority (priority) - filtrowanie po priorytecie:

- Low - niski (SLA: 72h)
- Medium - średni (SLA: 48h)
- High - wysoki (SLA: 24h)
- Critical - krytyczny (SLA: 4h)

Category (category) - filtrowanie po kategorii:

- Hardware - sprzęt
- Software - oprogramowanie
- Network - sieć
- Access - dostęp
- Other - inne

AssignedTold (assignedTold) - filtrowanie po przypisanyem techniku (ID użytkownika z rolą Technician/Admin)

CreatedById (createdById) - filtrowanie po twórcy zgłoszenia (ID użytkownika który utworzył zgłoszenie)

IsOverdue (isOverdue) - filtrowanie przeterminowanych zgłoszeń:

- true - tylko przeterminowane (przekroczyły SLA)
- false - wszystkie oprócz przeterminowanych

6.3. Walidacja filtrów

System waliduje czy użytkownicy istnieją:

```
if (parameters.AssignedTold.HasValue)
{
    var assigneeExists = await _context.Users.AnyAsync(u => u.Id == parameters.AssignedTold.Value);
    if (!assigneeExists)
    {
        return BadRequest(new
        {
            message = "Assigned user not found",
            parameter = "AssignedTold",
            value = parameters.AssignedTold.Value
        });
    }
    query = query.Where(t => t.AssignedTold == parameters.AssignedTold.Value);
}
```

6.4. Zrzuty ekranu

All Tickets

Browse, search, filter, and sort all IT support tickets

Search tickets by title, description, or user...

FILTERS & SORTING

Status	Priority	Category	Assigned To	Show Only
New	Low	Hardware	All Technicians	<input type="checkbox"/> Overdue Tickets

CLEAR ALL FILTERS

Showing 4 of 4 tickets

ID ↑	TITLE ↓	STATUS ↓	PRIORITY ↓	CATEGORY	CREATED ↓	ACTIONS
#31	Prośba o nową klawiaturę by Paweł Wojciechowski	NEW	LOW	Hardware	22 wrz 2025, 22:25	<button>VIEW</button>
#39	Prośba o nową mysz bezprzewodową by Adam Wrobel	NEW	LOW	Hardware	25 wrz 2025, 18:25	<button>VIEW</button>
#55	Prośba o instalację dodatkowego monitora by Marek Kowalczyk	NEW	LOW	Hardware	26 wrz 2025, 05:25	<button>VIEW</button>
#95	Prośba o większy dysk w laptopie by Małgorzata Zielińska	NEW	LOW	Hardware	13 wrz 2025, 08:25	<button>VIEW</button>

All Tickets

Browse, search, filter, and sort all IT support tickets

Search tickets by title, description, or user...

FILTERS & SORTING

Status	Priority	Category	Assigned To	Show Only
New	Low	Hardware	All Technicians	<input type="checkbox"/> Overdue Tickets

CLEAR ALL FILTERS

Showing 4 of 4 tickets

ID ↑	TITLE ↓	STATUS ↓	PRIORITY ↓	CATEGORY	CREATED ↓	ACTIONS
#31	Prośba o nową klawiaturę by Paweł Wojciechowski	NEW	LOW	Hardware	22 wrz 2025, 22:25	<button>VIEW</button>
#39	Prośba o nową mysz bezprzewodową by Adam Wrobel	NEW	LOW	Hardware	25 wrz 2025, 18:25	<button>VIEW</button>
#55	Prośba o instalację dodatkowego monitora by Marek Kowalczyk	NEW	LOW	Hardware	26 wrz 2025, 05:25	<button>VIEW</button>
#95	Prośba o większy dysk w laptopie by Małgorzata Zielińska	NEW	LOW	Hardware	13 wrz 2025, 08:25	<button>VIEW</button>

Przykładowy Request URL:

`http://localhost:5000/api/tickets?status=Open&priority=High&category=Hardware`

7. REALIZACJA WYSZUKIWANIA

7.1. Kod backendu

Wyszukiwanie jest implementowane w kontrolerze TicketsController.cs w metodzie GetTickets():

```
if (!string.IsNullOrEmpty(parameters.Search))
{
    var searchLower = parameters.Search.ToLower();
    query = query.Where(t =>
        t.Title.ToLower().Contains(searchLower) ||
        t.Description.ToLower().Contains(searchLower) ||
        t.CreatedBy.FirstName.ToLower().Contains(searchLower) ||
        t.CreatedBy.LastName.ToLower().Contains(searchLower) ||
        t.CreatedBy.Email.ToLower().Contains(searchLower) ||
        (t.AssignedTo != null && (
            t.AssignedTo.FirstName.ToLower().Contains(searchLower) ||
            t.AssignedTo.LastName.ToLower().Contains(searchLower)
        ))
    );
}
```

7.2. Opis działania

Mechanizm wyszukiwania w systemie przeszukuje jednocześnie siedem pól: Ticket.Title (tytuł zgłoszenia), Ticket.Description (opis problemu), CreatedBy.FirstName (imię twórcy zgłoszenia), CreatedBy.LastName (nazwisko twórcy), CreatedBy.Email (email twórcy), AssignedTo.FirstName (imię przypisanego technika) oraz AssignedTo.LastName (nazwisko przypisanego technika).

Wyszukiwanie ma charakter case-insensitive, co oznacza że nie rozróżnia wielkich i małych liter. Przeszukiwanie jest częściowe, czyli system wyszukuje wszystkie rekordy zawierające podaną frazę. Zastosowano operator OR, dzięki czemu zwracane są wyniki jeśli fraza występuje w którymkolwiek z przeszukiwanych pól. Wyszukiwanie odbywa się poprzez parametr search w query string.

Przykładowo, zapytanie z parametrem search=drukarka znajdzie wszystkie zgłoszenia zawierające słowo "drukarka" w tytule lub opisie. Zapytanie search=kowalski znajdzie zgłoszenia utworzone przez lub przypisane do osoby o nazwisku Kowalski. Z kolei zapytanie search=network znajdzie wszystkie zgłoszenia związane z tematyką sieciową.

7.3. Zrzuty ekranu

All Tickets

Browse, search, filter, and sort all IT support tickets

The screenshot shows a dark-themed web application for managing IT support tickets. At the top, there's a search bar containing the text "prosba". Below it is a "FILTERS & SORTING" section with dropdown menus for Status (New), Priority (Low), Category (Hardware), Assigned To (All Technicians), and a checked checkbox for "Show Only Overdue Tickets". A "CLEAR ALL FILTERS" button is also present. The main area displays a table with the following data:

ID	TITLE	STATUS	PRIORITY	CATEGORY	CREATED	ACTIONS
#31	Prośba o nową klawiaturę	NEW	LOW	Hardware	22 wrz 2025, 22:25	<button>VIEW</button>
#39	Prośba o nową mysz bezprzewodową	NEW	LOW	Hardware	25 wrz 2025, 18:25	<button>VIEW</button>
#55	Prośba o instalację dodatkowego monitora	NEW	LOW	Hardware	26 wrz 2025, 05:25	<button>VIEW</button>
#95	Prośba o większy dysk w laptopie	NEW	LOW	Hardware	13 wrz 2025, 08:25	<button>VIEW</button>

At the bottom right of the table, it says "Showing 4 of 4 tickets".

The screenshot shows the Swagger UI interface for a REST API endpoint. At the top, it shows the URL "http://localhost:5000/swagger/index.html". The main area is titled "Responses" and contains the following sections:

- Curl:** A code block showing a cURL command to fetch tickets: `curl -X 'GET' \ 'http://localhost:5000/api/Tickets?page=1&search=pro%C5%9Bba' \ -H 'accept: application/json'`
- Request URL:** A text input field containing the URL: `http://localhost:5000/api/Tickets?page=1&search=pro%C5%9Bba`
- Server response:**
 - Code:** 200
 - Details:** Response body (JSON object)

```
{
  "items": [
    {
      "id": 77,
      "title": "Prośba o instalację oprogramowania CAD",
      "description": "Inżynier potrzebuje AutoCAD do pracy nad projektami.",
      "status": "New",
      "priority": "Medium",
      "category": "Software",
      "createdBy": {
        "id": 25,
        "fullName": "Dorota Piotrowska",
        "email": "dorota.piotrowska@firma.pl",
        "role": "User",
        "department": "HR"
      },
      "assignedTo": null,
      "createdAt": "2025-10-05T10:25:13.856324",
      "updatedAt": "2025-10-05T12:25:13.856324",
      "resolvedAt": null,
      "closedAt": null,
      "resolutionNotes": null,
      "viewCount": 25,
      "commentCount": 4,
      "attachmentCount": 0
    }
  ]
}
```

Buttons for "Download" and "Copy" are visible next to the response body.

- Response headers:** A code block showing the response headers: `content-type: application/json; charset=utf-8 date: Wed, 15 Oct 2025 16:28:15 GMT server: Kestrel transfer-encoding: chunked`

Przykładowy Request URL:

`http://localhost:5000/api/tickets?search=network`

8. REALIZACJA PAGINACJI

8.1. Kod backendu

Paginacja jest implementowana w kontrolerze TicketsController.cs w głównej metodzie GetTickets():

```
var pageSize = parameters.PageSize;
var pageNumber = parameters.Page;

var totalPages = (int)Math.Ceiling(totalCount / (double)pageSize);
if (pageNumber > totalPages && totalCount > 0)
{
    return BadRequest(new
    {
        message = $"Page number {pageNumber} exceeds total pages ({totalPages})",
        parameter = "Page",
        value = pageNumber,
        totalPages = totalPages,
        totalCount = totalCount
    });
}

var items = await query
    .Skip((pageNumber - 1) * pageSize)
    .Take(pageSize)
    .Select(t => MapToTicketDto(t))
    .ToListAsync();
```

8.2. Opis działania

System implementuje pełną paginację po stronie backendu z następującymi cechami:

Parametry paginacji:

Page (numer strony):

- Domyślnie: 1
- Minimalna wartość: 1
- Walidacja: Musi być większy lub równy 1

PageSize (rozmiar strony):

- Domyślnie: 10
- Zakres: 1-100
- Walidacja: Musi być między 1 a 100

Walidacje:

System wykonuje 5 walidacji przed zwróceniem danych:

1. Walidacja Page < 1 - zwraca błąd 400 Bad Request
2. Walidacja PageSize poza zakresem - zwraca błąd 400 Bad Request
3. Walidacja Page > totalPages - zwraca błąd 400 Bad Request
4. Walidacja nieistniejącego assignedTold - zwraca błąd 400 Bad Request
5. Walidacja nieistniejącego createdByld - zwraca błąd 400 Bad Request

Odpowiedź API:

```
{  
  "items": [ /* tablica zgłoszeń */ ],  
  "totalCount": 125,  
  "pageNumber": 1,  
  "pageSize": 10,  
  "totalPages": 13,  
  "hasPreviousPage": false,  
  "hasNextPage": true  
}
```

Przetwarzanie:

- Backend odbiera parametry `page` i `pageSize`
- Waliduje parametry (zwraca 400 Bad Request jeśli nieprawidłowe)
- Stosuje filtry, wyszukiwanie, sortowanie
- Liczy całkowitą liczbę wyników (`totalCount`)
- Oblicza liczbę stron (`totalPages`)
- Pobiera tylko żądaną stronę (SKIP/TAKE)
- Zwraca dane z metadanymi paginacji

8.3. Zrzuty ekranu

#	Tytuł zgłoszenia	Status	Priorytet	Kategoria	Data utworzenia	Akcja
#57	Problemy z połączeniem Bluetooth by Wojciech Witkowski	OPEN	LOW	Hardware	2 paź 2025, 14:25	<button>VIEW</button>
#99	Backup zajmuje zbyt wiele czasu by Agnieszka Kamińska	IN PROGRESS	MEDIUM	Software	2 paź 2025, 12:25	<button>VIEW</button>
#100	Prośba o zmianę hasła administratora by Olwiańska	RESOLVED	CRITICAL	Account	1 paź 2025, 20:25	<button>VIEW</button>
#89	Wiadomości trafiają do spamu by Krzysztof Szymański	IN PROGRESS	HIGH	Email	1 paź 2025, 10:25	<button>VIEW</button>
#24	WiFi rozłącza się na laptopie by Wojciech Witkowski	RESOLVED	LOW	Email	30 wrz 2025, 23:25	<button>VIEW</button>
#9	Wolne działanie sieci w biurze by Jacek Kaczmarek	IN PROGRESS	HIGH	Network	30 wrz 2025, 21:25	<button>VIEW</button>
#70	Komunikat o pełnym dysku C by Natalia Kucharska	OPEN	MEDIUM	Software	30 wrz 2025, 09:25	<button>VIEW</button>
#40	Popękane słuchawki by Dariusz Michalski	OPEN	LOW	Hardware	30 wrz 2025, 08:25	<button>VIEW</button>

PREVIOUS 1 2 3 4 5 6 7 ... 12 NEXT

Curl

```
curl -X "GET" \
  "http://localhost:5000/api/Tickets?Page=1&PageSize=3" \
  -H "accept: application/json"
```

Request URL

```
http://localhost:5000/api/Tickets?Page=1&PageSize=3
```

Server response

Code Details

200 Response body

```
{ "items": [ { "id": 34, "title": "Problem z rozpoznaniem drukarki", "description": "System nie widzi drukarki sieciowej. Próbowano ponownej instalacji driverów.", "status": "Closed", "priority": "Critical", "category": "account", "createdBy": { "id": 22, "fullname": "Paweł Wojciechowski", "email": "pawel.wojciechowski@firma.pl", "role": "User", "department": "Sprzedaż" }, "assignedTo": { "id": 10, "fullname": "Stanisław Piątrowski", "email": "stanislaw.piatrowski@firma.pl", "role": "Admin", "department": "Infrastruktura IT" }, "createdAt": "2025-10-08T07:25:13.856Z", "updatedAt": "2025-10-15T14:23:54.745Z" } ] }
```

Download

Responses

Code Description

Links

Przykładowy Request URL:

<http://localhost:5000/api/tickets?page=2&pageSize=5>

8.4. Kod filtrowania, sortowania, paginacji po stronie frontendu

```
<script setup lang="ts">
import { onMounted, ref, computed, watch } from "vue";
import { useRouter, useRoute } from "vue-router";
import { useTicketStore } from "../../stores/ticketStore";
import { useUserStore } from "../../stores/userStore";
import NavigationGlobal from "../../components/navigation-global.vue";
import FooterGlobal from "../../components/footer-global.vue";
import StatusBadge from "../../components/StatusBadge.vue";
import SearchInput from "../../components/SearchInput.vue";
import Pagination from "../../components/Pagination.vue";
import {
  TicketStatus,
  TicketPriority,
  TicketCategory,
} from "../../types/ticket.types";

const router = useRouter();
const route = useRoute();
const ticketStore = useTicketStore();
const userStore = useUserStore();

const searchQuery = ref("");
const selectedStatus = ref<TicketStatus | "">("");
const selectedPriority = ref<TicketPriority | "">("");
const selectedCategory = ref<TicketCategory | "">("");
const selectedAssignee = ref<number | "">("");
const showOverdueOnly = ref(false);
const sortBy = ref("createdAt");
const sortOrder = ref<"asc" | "desc">("desc");
const currentPage = ref(1);
const pageSize = ref(10);

const tickets = computed(() => ticketStore.tickets);
const pagedResult = computed(() => ticketStore.pagedResult);
const loading = computed(() => ticketStore.loading);
const technicians = computed(() => userStore.technicians);

onMounted(async () => {
  userStore.loadCurrentUser();
  await userStore.fetchTechnicians();

  if (route.query.status)
    selectedStatus.value = route.query.status as TicketStatus;
  if (route.query.priority)
    selectedPriority.value = route.query.priority as TicketPriority;
  if (route.query.category)
    selectedCategory.value = route.query.category as TicketCategory;
  if (route.query.search) searchQuery.value = route.query.search as string;

  await fetchTickets();
});

const fetchTickets = async () => {
```

```
const params: any = {
  page: currentPage.value,
  pageSize: pageSize.value,
  sortBy: sortBy.value,
  sortOrder: sortOrder.value,
};

if (searchQuery.value) params.search = searchQuery.value;
if (selectedStatus.value) params.status = selectedStatus.value;
if (selectedPriority.value) params.priority = selectedPriority.value;
if (selectedCategory.value) params.category = selectedCategory.value;
if (selectedAssignee.value) params.assignedToId = selectedAssignee.value;
if (showOverdueOnly.value) params.isOverdue = true;

await ticketStore.fetchTickets(params);
};

const handleSearch = (value: string) => {
  searchQuery.value = value;
  currentPage.value = 1;
  fetchTickets();
};

const handleFilterChange = () => {
  currentPage.value = 1;
  fetchTickets();
};

const handleSort = (field: string) => {
  if (sortBy.value === field) {
    sortOrder.value = sortOrder.value === "asc" ? "desc" : "asc";
  } else {
    sortBy.value = field;
    sortOrder.value = "desc";
  }
  fetchTickets();
};

const handlePageChange = (page: number) => {
  currentPage.value = page;
  fetchTickets();
  window.scrollTo({ top: 0, behavior: "smooth" });
};

const clearFilters = () => {
  searchQuery.value = "";
  selectedStatus.value = "";
  selectedPriority.value = "";
  selectedCategory.value = "";
  selectedAssignee.value = "";
  showOverdueOnly.value = false;
  currentPage.value = 1;
  fetchTickets();
};

const goToTicket = (id: number) => {
```

```

router.push(` /tickets/${id}`);
};

const formatDate = (date: string): string => {
  return new Date(date).toLocaleDateString("pl-PL", {
    year: "numeric",
    month: "short",
    day: "numeric",
    hour: "2-digit",
    minute: "2-digit",
  });
};

const getSortIcon = (field: string): string => {
  if (sortBy.value !== field) return "↑";
  return sortOrder.value === "asc" ? "↑" : "↓";
};
</script>

<template>
<div class="min-h-screen w-screen bg-black text-white">
<NavigationGlobal color="black" />

<section class="flex w-full items-center justify-center py-16">
<div class="w-4/5 max-w-7xl md:w-11/12 lg:w-4/5">
  <!-- Header -->
  <div class="mb-8">
    <h1 class="mb-4 text-4xl font-extrabold tracking-tight md:text-5xl">
      All <span class="text-k-main">Tickets</span>
    </h1>
    <p class="text-lg text-zinc-400">
      Browse, search, filter, and sort all IT support tickets
    </p>
  </div>

  <!-- Search Bar -->
  <div class="mb-6">
    <SearchInput
      v-model="searchQuery"
      @search="handleSearch"
      placeholder="Search tickets by title, description, or user..." />
  </div>

  <!-- Filters -->
  <div class="mb-8 rounded-lg border border-zinc-800 bg-zinc-900 p-6">
    <h2 class="mb-4 text-lg font-bold uppercase tracking-wide">
      Filters & Sorting
    </h2>
    <div class="grid gap-4 md:grid-cols-2 lg:grid-cols-5">
      <!-- Status Filter -->
      <div>
        <label class="mb-2 block text-sm font-semibold text-zinc-400">
          Status</label>
        >
        <select
          v-model="selectedStatus">

```

```
@change="handleFilterChange"
class="w-full rounded-lg border border-zinc-700 bg-black px-4 py-2 text-white transition duration-300
focus:border-k-main focus:outline-none">
<option value="">All Statuses</option>
<option
v-for="status in Object.values(TicketStatus)"
:key="status"
:value="status">
{{ status }}
</option>
</select>
</div>

<!-- Priority Filter -->
<div>
<label class="mb-2 block text-sm font-semibold text-zinc-400">
>Priority</label>
>
<select
v-model="selectedPriority"
@change="handleFilterChange"
class="w-full rounded-lg border border-zinc-700 bg-black px-4 py-2 text-white transition duration-300
focus:border-k-main focus:outline-none">
<option value="">All Priorities</option>
<option
v-for="priority in Object.values(TicketPriority)"
:key="priority"
:value="priority">
{{ priority }}
</option>
</select>
</div>

<!-- Category Filter -->
<div>
<label class="mb-2 block text-sm font-semibold text-zinc-400">
>Category</label>
>
<select
v-model="selectedCategory"
@change="handleFilterChange"
class="w-full rounded-lg border border-zinc-700 bg-black px-4 py-2 text-white transition duration-300
focus:border-k-main focus:outline-none">
<option value="">All Categories</option>
<option
v-for="category in Object.values(TicketCategory)"
:key="category"
:value="category">
{{ category }}
</option>
</select>
</div>

<!-- Assignee Filter -->
<div>
<label class="mb-2 block text-sm font-semibold text-zinc-400">
```

```

>Assigned To</label>
>
<select
  v-model="selectedAssignee"
  @change="handleFilterChange"
  class="w-full rounded-lg border border-zinc-700 bg-black px-4 py-2 text-white transition duration-300
focus:border-k-main focus:outline-none">
  <option value="">All Technicians</option>
  <option
    v-for="tech in technicians"
    :key="tech.id"
    :value="tech.id">
    {{ tech.fullName }}
  </option>
</select>
</div>

<!-- Overdue Filter --&gt;
&lt;div&gt;
  &lt;label class="mb-2 block text-sm font-semibold text-zinc-400"&gt;
    &gt;Show Only&lt;/label&gt;
  &gt;
  &lt;label class="flex items-center gap-2"&gt;
    &lt;input
      type="checkbox"
      v-model="showOverdueOnly"
      @change="handleFilterChange"
      class="h-5 w-5 rounded border-zinc-700 bg-black text-k-main focus:ring-k-main" /&gt;
    &lt;span class="text-white"&gt;Overdue Tickets&lt;/span&gt;
  &lt;/label&gt;
&lt;/div&gt;
&lt;/div&gt;

&lt;div class="mt-4 flex items-center justify-between"&gt;
  &lt;button
    @click="clearFilters"
    class="text-sm font-semibold uppercase tracking-wide text-zinc-400 transition duration-300 hover:text-k-main"&gt;
    Clear All Filters
  &lt;/button&gt;
  &lt;p class="text-sm text-zinc-400"&gt;
    Showing
    &lt;span class="font-bold text-white"&gt;{{ pagedResult?.items.length || 0 }}&lt;/span&gt;
    of
    &lt;span class="font-bold text-white"&gt;{{ pagedResult?.totalCount || 0 }}&lt;/span&gt;
    tickets
  &lt;/p&gt;
&lt;/div&gt;
&lt;/div&gt;

<!-- Loading State --&gt;
&lt;div v-if="loading" class="py-20 text-center"&gt;
  &lt;div&gt;</pre>

```

```

    class="inline-block h-12 w-12 animate-spin rounded-full border-4 border-solid border-k-main border-r-transparent">></div>
    <p class="mt-4 text-lg text-zinc-400">Loading tickets...</p>
</div>

<!-- Tickets List -->
<div v-else-if="tickets.length === 0" class="py-20 text-center">
    <p class="text-xl text-zinc-400">
        No tickets found with current filters.
    </p>
    <button
        @click="clearFilters"
        class="mt-4 font-semibold uppercase tracking-wide text-k-main transition duration-300 hover:underline">
        Clear Filters
    </button>
</div>

<!-- Desktop Table View -->
<div
    v-else
    class="hidden overflow-hidden rounded-lg border border-zinc-800 lg:block">
    <table class="w-full">
        <thead class="bg-zinc-900">
            <tr>
                <th
                    @click="handleSort('id')"
                    class="cursor-pointer px-6 py-4 text-left text-sm font-bold uppercase tracking-wide text-zinc-400 transition duration-300 hover:text-k-main">
                        ID {{ getSortIcon("id") }}
                </th>
                <th
                    @click="handleSort('title')"
                    class="cursor-pointer px-6 py-4 text-left text-sm font-bold uppercase tracking-wide text-zinc-400 transition duration-300 hover:text-k-main">
                        Title {{ getSortIcon("title") }}
                </th>
                <th
                    @click="handleSort('status')"
                    class="cursor-pointer px-6 py-4 text-left text-sm font-bold uppercase tracking-wide text-zinc-400 transition duration-300 hover:text-k-main">
                        Status {{ getSortIcon("status") }}
                </th>
                <th
                    @click="handleSort('priority')"
                    class="cursor-pointer px-6 py-4 text-left text-sm font-bold uppercase tracking-wide text-zinc-400 transition duration-300 hover:text-k-main">
                        Priority {{ getSortIcon("priority") }}
                </th>
                <th
                    class="px-6 py-4 text-left text-sm font-bold uppercase tracking-wide text-zinc-400">
                        Category
                </th>
                <th
                    @click="handleSort('createdAt')"
                    class="cursor-pointer px-6 py-4 text-left text-sm font-bold uppercase tracking-wide text-zinc-400 transition duration-300 hover:text-k-main">

```

```

Created {{ getSortIcon("createdAt") }}
```

	Actions
<p>ticket.id</p> <p>ticket.title</p> <p>by {{ ticket.createdBy.fullName }}</p> <p>Status: {{ ticket.status }}</p> <p>Priority: {{ ticket.priority }}</p> <p>Category: {{ ticket.category }}</p> <p>Created: {{ formatDate(ticket.createdAt) }}</p>	<a @click="goToTicket(ticket.id)" class="rounded-lg bg-k-main px-4 py-2 text-sm font-bold uppercase tracking-wide text-white transition duration-300 hover:scale-105 active:translate-y-0.5" href="#">View

```

<!-- Mobile Card View -->
<div v-if="!loading && tickets.length > 0" class="space-y-4 lg:hidden">
<div
  v-for="ticket in tickets"
  :key="ticket.id"
  @click="goToTicket(ticket.id)">
```

```
    class="cursor-pointer rounded-lg border border-zinc-800 bg-zinc-900 p-6 transition duration-300 hover:border-k-  
main">  
    <div class="mb-4 flex items-start justify-between">  
        <h3 class="text-xl font-bold text-white">  
            #{{ ticket.id }} - {{ ticket.title }}  
        </h3>  
        <div class="flex gap-2">  
            <StatusBadge type="status" :value="ticket.status" small />  
            <StatusBadge type="priority" :value="ticket.priority" small />  
        </div>  
    </div>  
    <div class="space-y-2 text-sm text-zinc-400">  
        <p>📁 {{ ticket.category }}</p>  
        <p>👤 {{ ticket.createdBy.fullName }}</p>  
        <p>🕒 {{ formatDate(ticket.createdAt) }}</p>  
        <p v-if="ticket.assignedTo">  
            🧑 Assigned: {{ ticket.assignedTo.fullName }}  
        </p>  
    </div>  
    </div>  
    </div>  
  
    <!-- Pagination -->  
    <Pagination  
        v-if="pagedResult"  
        :current-page="pagedResult.pageNumber"  
        :total-pages="pagedResult.totalPages"  
        :has-next="pagedResult.hasNextPage"  
        :has-previous="pagedResult.hasPreviousPage"  
        @page-changed="handlePageChange" />  
    </div>  
    </section>  
  
    <FooterGlobal />  
    </div>  
</template>
```

9. PODSUMOWANIE

System IT Help Desk został w pełni zaimplementowany zgodnie z założeniami projektu. Aplikacja spełnia wszystkie wymagania funkcjonalne i techniczne:

Zrealizowane funkcjonalności:

- REST API - 10 endpoint'ów zgodnych z poziomem 2 Richardsona
- SPA - Vue.js 3 z TypeScript
- Sortowanie - 5 pól, 2 kierunki
- Filtrowanie - 4 typów filtrów z walidacją
- Wyszukiwanie - 7 pól, case-insensitive, częściowe dopasowanie
- Paginacja - z pełną walidacją po stronie backendu
- Swagger - kompletna dokumentacja wszystkich endpoint'ów
- Walidacja - wszystkie parametry walidowane, zwracanie błędów 400

Wykorzystane technologie:

- Backend: ASP.NET Core 9.0, C# 12, Entity Framework Core 9.0, SQLite
- Frontend: Vue.js 3.5, TypeScript 5, Vite 6.0, Pinia, Vue Router
- API: REST, Swagger/OpenAPI, Swashbuckle
- Baza danych: SQLite, Code First, 5 tabel z relacjami

Statystyki projektu:

- Endpoint'ów: 10
- Pół sortowania: 8
- Typów filtrów: 6
- Pół wyszukiwania: 7
- Tabel w bazie: 5