**Politechnika Śląska**
Wydział Automatyki, Elektroniki i Informatyki

# System rozgrywek turniejowych

Uniwersalna platforma do tworzenia i śledzenia różnego
rodzaju turniejów sportowych i planszowych

Etap 4: **Modelowanie zachowania (4-MZ-P2) - Diagramy sekwencji + Prototyp #2**

Katowice, 09 stycznia 2026 r.

Zespół projektowy:
Piotr Maj – Menadżer projektu/Lider techniczny
Bartosz Jędryka – Programista
Aleksandra Kuś – Tester
Wojciech Pędziwiatr – Programista
Adrian Suchenia – Programista
Kontakt (Menadżer projektu): pm311399@student.polsl.pl

**Opis Przypadku Użycia: Logowanie użytkownika**

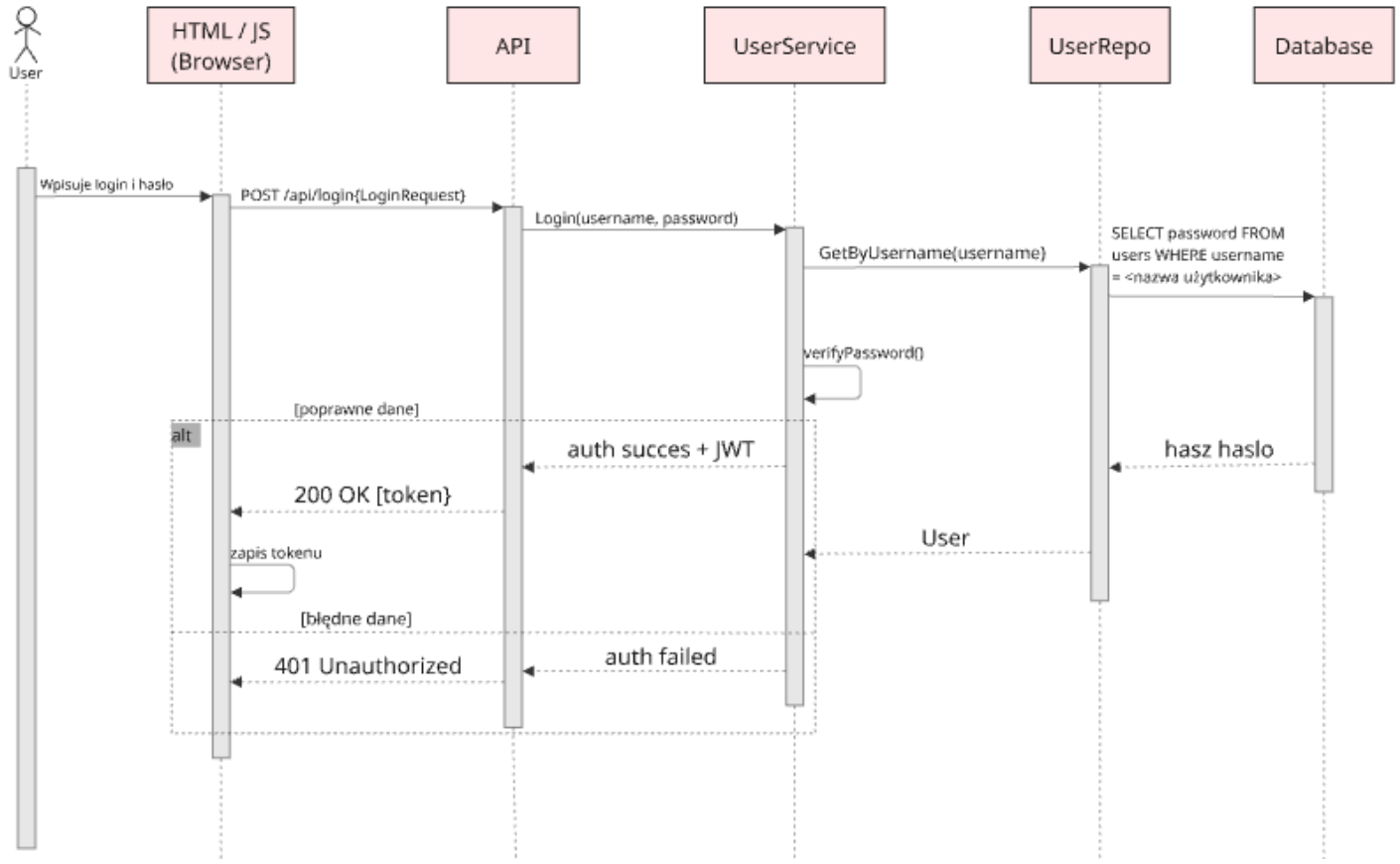
**Cel:** Autoryzacja użytkownika w systemie i nadanie uprawnień do korzystania z platformy poprzez wygenerowanie tokena JWT.

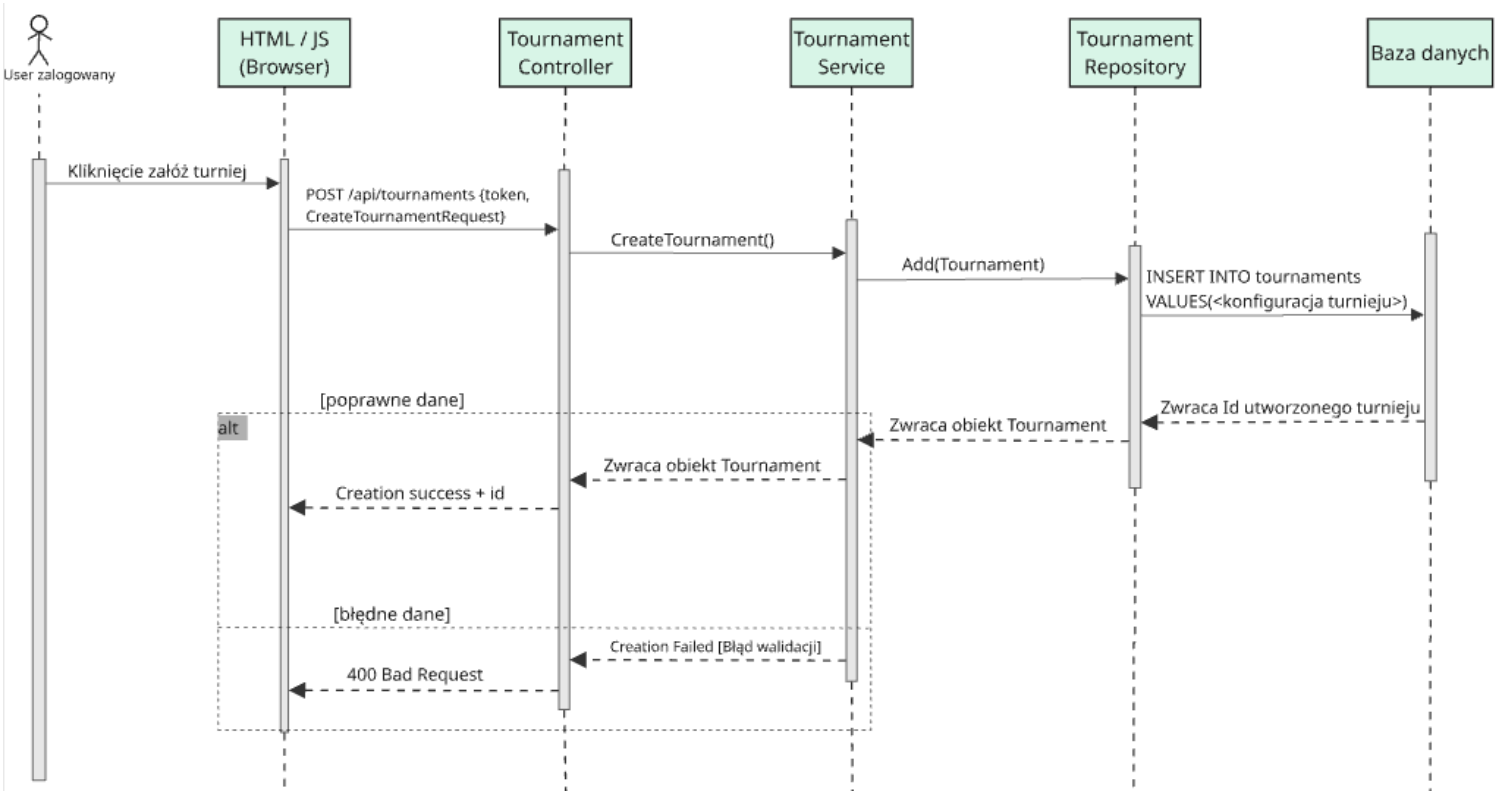
**Przebieg procesu:**

- **Inicjacja:** Użytkownik wprowadza login oraz hasło w formularzu przeglądarki (HTML/JS).
- **Żądanie API:** Przeglądarka przesyła dane za pomocą metody POST /api/login do bramki API, która deleguje zadanie do UserService.
- **Dostęp do danych:** UserService komunikuje się z UserRepo, aby pobrać dane użytkownika z bazy danych za pomocą zapytania SQL SELECT password FROM users....
- **Weryfikacja:** Baza danych zwraca zahaszowane hasło, a UserService wykonuje wewnętrzną metodę verifyPassword(), aby porównać dane wprowadzone przez użytkownika z tymi zapisanymi w systemie.

**Scenariusze wynikowe (Blok ALT):**

1. **Logowanie pomyślne (Poprawne dane):**
    - System generuje token autoryzacyjny JWT.
    - API zwraca status 200 OK [token].
    - Przeglądarka wykonuje operację zapisu tokenu (np. w Local Storage), co umożliwia dalszą pracę w systemie.
2. **Błąd autoryzacji (Błędne dane):**
    - UserService zwraca informację o niepowodzeniu (auth failed).
    - API przesyła do użytkownika komunikat o błędzie ze statusem 401 Unauthorized.


User
HTML / JS (Browser)
API
UserService
UserRepo
Database
Wpisuje login i hasło
POST /api/login{LoginRequest}
Login(username, password)
GetByUsername(username)
SELECT password FROM users WHERE username = <nazwa użytkownika>
verifyPassword()
[poprawne dane]
alt
auth succes + JWT
hasz haslo
200 OK [token}
User
zapis tokenu
[błędne dane]
401 Unauthorized
auth failed


User zalogowany
HTML / JS (Browser)
Tournament Controller
Tournament Service
Tournament Repository
Baza danych
Kliknięcie załóż turniej
POST /api/tournaments {token, CreateTournamentRequest}
CreateTournament()
Add(Tournament)
INSERT INTO tournaments VALUES(<konfiguracja turnieju>)
[poprawne dane]
alt
Zwraca Id utworzonego turnieju
Zwraca obiekt Tournament
Zwraca obiekt Tournament
Creation success + id
[błędne dane]
Creation Failed [Błąd walidacji]
400 Bad Request

**Opis Przypadku Użycia: Tworzenie nowego turnieju**

**Cel:** Umożliwienie zalogowanemu organizatorowi utworzenia nowej instancji turnieju w systemie wraz z zapisem jego konfiguracji w bazie danych.

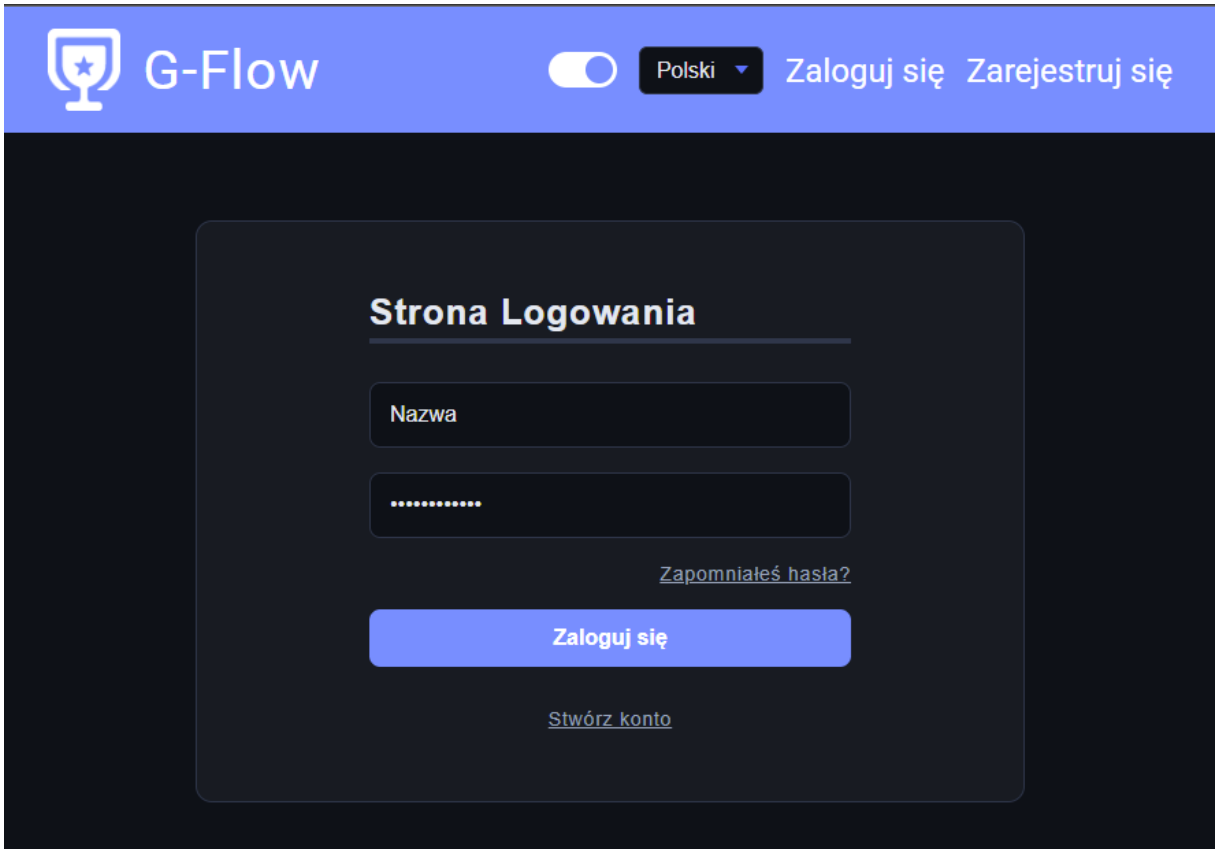
**Opis przebiegu (scenariusz główny):**

1. **Inicjacja:** Zalogowany użytkownik wypełnia formularz i klika przycisk "Załóż turniej" w interfejsie przeglądarki (HTML/JS).
2. **Przesłanie danych:** Warstwa front-endowa wysyła żądanie POST /api/tournaments, przekazując w nagłówku token autoryzacyjny oraz obiekt CreateTournamentRequest w formacie JSON do Tournament Controller.
3. **Logika biznesowa:** Kontroler deleguje zadanie do warstwy logiki (Tournament Service) poprzez metodę CreateTournament(). Serwis odpowiada za walidację danych wejściowych (np. sprawdzenie poprawności dat i nazw).
4. **Persystencja:** Po pomyślnej walidacji, serwis wywołuje metodę Add(Tournament) w warstwie dostępu do danych (Tournament Repository).
5. **Zapis w bazie:** Repozytorium wykonuje zapytanie SQL INSERT INTO tournaments do Bazy danych. Baza zwraca unikalny identyfikator (ID) nowo utworzonego rekordu.
6. **Potwierdzenie:** Informacja o sukcesie i nowym obiekcie jest przekazywana z powrotem przez wszystkie warstwy. Kontroler zwraca do interfejsu odpowiedź 201 Created (na diagramie jako Creation success) wraz z nadanym ID.

**Scenariusz alternatywny (Błąd walidacji):**

- W przypadku, gdy dane w Tournament Service nie przejdą walidacji, proces zapisu w bazie zostaje przerwany.
- System zwraca informację o błędzie (Creation Failed) do kontrolera, który wysyła do użytkownika odpowiedź 400 Bad Request.

Frontend (HTML/JS)




```
function handleLogin(e) {
  e.preventDefault();
  if (typeof loginUser === 'function') {
    loginUser(username, password);
  } else {
    console.warn('loginUser is not a function', loginUser);
  }
}
```

```
async function loginUser(username, password) {
  await loginAPI(username, password)
    .then((res) => {
      if (res) {
        localStorage.setItem('token', res?.data.accessToken);
        const userObj = {
          username: res?.data.username,
          email: res?.data.email,
          id: res?.data.id,
        };
        localStorage.setItem('user', JSON.stringify(userObj));
        setToken(res?.data.accessToken);
        setUser(userObj);
        const theme = localStorage.getItem('theme');
        toast.success(`${t('loginSuccess')}!`, {
          theme: theme,
        });
        navigate('/');
      }
    })
    .catch((e) => {
      const theme = localStorage.getItem('theme');
      handleError(e);
    });
}
```

```
export async function loginAPI(username, password) {
  try {
    const data = api.post('/login', {
      username: username,
      password: password,
    });
    return data;
  } catch (error) {
    handleError(error);
  }
}
```

```
[ApiController]
[Route("api")]
1 odwołanie
public class UserController : ControllerBase
{

    Odwołania: 3
    private readonly IUserService _userService;

    Odwołania: 0
    public UserController(IUserService userService)
    {
        _userService = userService;
    }

    [HttpPost("login")]
    Odwołania: 0
    public IActionResult Login([FromBody] LoginRequest request)
    {
        User? user =_userService.Login(request);

        if(user is null)
        {
            return Unauthorized();
        }

        return Ok(new
        {
            id=user.Id,
            username=user.Username,
            email=user.Email,
            token=user.Token
        });
    }
```

```
Odwołania: 2
public class UserService : IUserService
{

    Odwołania: 2
    public User? Login(LoginRequest request)
    {
        var user = _userRepo.GetByUsername(request.username);
        if (user == null) return null;

        bool isPasswordValid = _passwordHasher.VerifyPassword(request.password, user.PasswordHash);
        if (!isPasswordValid) return null;

        user.Token = _tokenProvider.GenerateToken(user.Id, user.Username);
        return user;
    }
```

```
Odwołania: 2
public class UserRepositoryPostgres : IUserRepository
{

    Odwołania: 2
    public User? Add(User user)
    {
        _context.Users.Add(user);
        _context.SaveChanges();
        return user;
    }
```

Tworzenie Turnieju

```
[ApiController]
[Route("api/[controller]")]
1 odwołanie
public class TournamentController : ControllerBase
{

    [HttpPost]
    Odwołania: 0
    public IActionResult Create(CreateTournamentRequest request)
    {
        Tournament? tournament = tournamentService.CreateTournament(request);

        if(tournament is null)
        {
            return BadRequest();
        }

        return CreatedAtAction(nameof(Get), new { id = tournament.Id.ToString() }, null);
    }
```

```
1 odwołanie
public class TournamentService : ITournamentService
{
    Odwołania: 2
    public Tournament? CreateTournament(CreateTournamentRequest request)
    {
        if (string.IsNullOrWhiteSpace(request.Name) || request.Name.Length < 5)
        {
            return null;
        }

        if (request.MaxParticipants <= 0)
        {
            return null;
        }

        var tournament = new Tournament
        {
            Id = Guid.NewGuid().ToString(),
            Name = request.Name,
            OrganizerId = request.OrganizerId,
            SystemType = request.SystemType,
            PlayerLimit = request.MaxParticipants,
            Status = Domain.ValueObjects.TournamentStatus.CREATED,
            StartDate = request.StartDate,
            EndDate = request.EndDate
        };

        return _tournamentRepo.Add(tournament);
    }
```

```
Odwołania: 2
public class TournamentRepositoryPostgres : ITournamentRepository
{

    Odwołania: 2
    public Tournament? Add(Tournament tournament)
    {
        _context.Tournaments.Add(tournament);
        _context.SaveChanges();
        return tournament;
    }
```