Bazy danych 2025

Mini projekt - boardly.

Dzień i godzina zajęć: wtorek 13:15

Jakub Jurczyk, Aleksander Grzybek

AGH

Spis treści

1. Funkcjonalność 5
   1.1. Zarządzanie kontem użytkownika 5
   1.2. Zarządzanie tablicami 5
   1.3. Struktura tablicy 7
   1.4. Zarządzanie zadaniami (Karty) 8
   1.5. Współpraca w czasie rzeczywistym 9
2. Uprawnienia 10
3. Wykorzystana technologia 10
4. Struktura bazy danych 11
   4.1. Kolekcja tablic [boards] 11
   4.2. Kolekcja kart [cards] 12
   4.3. Kolekcja użytkowników [users] 12
   4.4. Kolekcja tokenów odświeżających [refreshTokens] 12
5. Zapytania bazodanowe 13
   5.1. Tablice [boards] 13
      5.1.1. Pobranie tablic po identyfikatorze użytkownika 13
      5.1.2. Pobranie tablic wraz z danymi członków po identyfikatorze użytkownika 13
      5.1.3. Pobranie członków danej tablicy 14
      5.1.4. Pobranie roli członka tablicy 14
      5.1.5. Utworzenie tablicy 14
      5.1.6. Zmiana aktywności członka tablicy 14
      5.1.7. Aktualizacja tablicy 15
      5.1.8. Aktualizacja tablicy wraz z jej zwrotem po aktualizacji z dołączonymi danymi członków 16
      5.1.9. Usunięcie tablicy 16
   5.2. Paski pływające [swimlanes] 17
      5.2.1. Pobranie pasków wszystkich pasków pływających danej tablicy 17
      5.2.2. Utworzenie paska pływającego 17
      5.2.3. Utworzenie paska pływającego wraz z jego zwrotem po utworzeniu 18
      5.2.4. Aktualizacja paska pływającego 18
      5.2.5. Aktualizacja paska pływającego wraz z jego zwrotem po aktualizacji 19
      5.2.6. Usunięcie paska pływającego 20
   5.3. Listy [lists] 21
      5.3.1. Pobranie wszystkich list paska pływającego 21
      5.3.2. Utworzenie nowej listy 21
      5.3.3. Utworzenie nowej listy wraz z jej zwrotem po utworzeniu 22
      5.3.4. Aktualizacja listy 22
      5.3.5. Aktualizacja listy wraz z jej zwrotem po aktualizacji 23
      5.3.6. Usunięcie listy 24
   5.4. Tagi [tags] 25
      5.4.1. Pobranie wszystkich tagów z paska pływającego 25
      5.4.2. Utworzenie nowego taga 25
      5.4.3. Utworzenie nowego taga wraz z jego zwrotem po utworzeniu 26
      5.4.4. Aktualizacja taga 27
      5.4.5. Aktualizacja taga wraz z jego zwrotem po zaaktualizowaniu 28
      5.4.6. Usunięcie taga 28
   5.5. Karty [cards] 29
      5.5.1. Pobranie kart zadanej tablicy 29
      5.5.2. Pobranie wybranej karty 31

5.5.3. Pobranie uproszczonego użytkownika ... 33
5.5.4. Pobranie użytkownika, który zablokował kartę ... 33
5.5.5. Utworzenie nowej karty ... 33
5.5.6. Utworzenie nowej karty wraz z jej zwrotem po utworzeniu ... 34
5.5.7. Aktualizacja karty ... 34
5.5.8. Aktualizacja karty wraz z jej zwrotem po zaaktualizowaniu ... 36
5.5.9. Przeniesienie karty do innej listy ... 36
5.5.10. Usunięcie karty ... 38
5.6. Tokeny odświeżenia [refreshTokens] ... 39
5.6.1. Dodanie tokena odświeżenia ... 39
5.6.2. Usunięcie wszystkich tokenów odświeżenia ... 39
5.6.3. Pobranie użytkownika na podstawie tokena odświeżenia ... 39
5.7. Użytkownicy [users] ... 40
5.7.1. Dodanie nowego użytkownika ... 40
5.7.2. Aktualizacja hasła użytkownika ... 40
5.7.3. Znalezienie użytkownika po dokładnej nazwie ... 41
5.7.4. Znalezienie wszystkich użytkowników ... 41
5.7.5. Porównanie hashy haseł ... 41
6. Schemat API REST ... 43
6.1. POST `/auth/signin` ... 43
6.2. POST `/auth/signup` ... 43
6.3. POST `/auth/refresh` ... 43
6.4. POST `/auth/revoke` ... 44
6.5. GET `/boards` ... 44
6.6. POST `/boards` ... 45
6.7. GET `/boards/{boardId}` ... 46
6.8. PATCH `/boards/{boardId}` ... 47
6.9. DELETE `/boards/{boardId}` ... 47
6.10. GET `/boards/{boardId}/cards` ... 48
6.11. POST `/boards/{boardId}/cards` ... 48
6.12. GET `/boards/{boardId}/cards/{cardId}` ... 49
6.13. PATCH `/boards/{boardId}/cards/{cardId}` ... 50
6.14. DELETE `/boards/{boardId}/cards/{cardId}` ... 51
6.15. PATCH `/boards/{boardId}/cards/{cardId}/move` ... 51
6.16. GET `/boards/{boardId}/swimlanes/{swimlaneId}/lists` ... 51
6.17. POST `/boards/{boardId}/swimlanes/{swimlaneId}/lists` ... 52
6.18. GET `/boards/{boardId}/swimlanes/{swimlaneId}/lists/{listId}` ... 52
6.19. PATCH `/boards/{boardId}/swimlanes/{swimlaneId}/lists/{listId}` ... 52
6.20. DELETE `/boards/{boardId}/swimlanes/{swimlaneId}/lists/{listId}` ... 53
6.21. GET `/boards/{boardId}/swimlanes` ... 53
6.22. POST `/boards/{boardId}/swimlanes` ... 53
6.23. GET `/boards/{boardId}/swimlanes/{swimlaneId}` ... 54
6.24. PATCH `/boards/{boardId}/swimlanes/{swimlaneId}` ... 55
6.25. DELETE `/boards/{boardId}/swimlanes/{swimlaneId}` ... 55
6.26. GET `/boards/{boardId}/swimlanes/{swimlaneId}/tags` ... 56
6.27. POST `/boards/{boardId}/swimlanes/{swimlaneId}/tags` ... 56
6.28. GET `/boards/{boardId}/swimlanes/{swimlaneId}/tags/{tagId}` ... 56
6.29. PATCH `/boards/{boardId}/swimlanes/{swimlaneId}/tags/{tagId}` ... 57
6.30. DELETE `/boards/{boardId}/swimlanes/{swimlaneId}/tags/{tagId}` ... 57

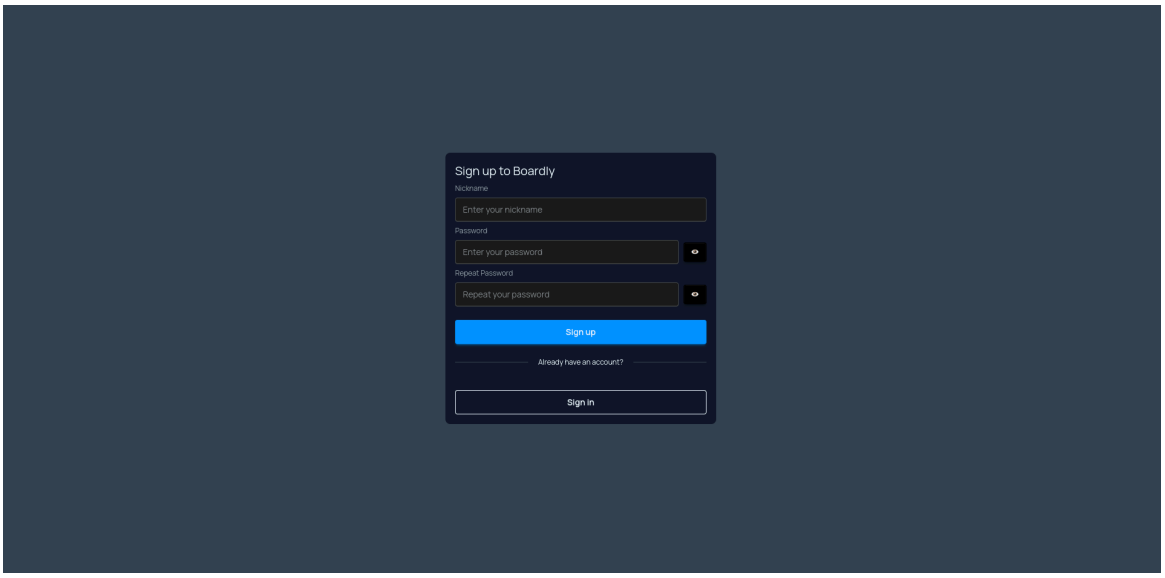
6.31. GET /users/me 57
6.32. PATCH /users/me 58
6.33. GET /users 58
6.34. GET /users/{userId} 58

1. Funkcjonalność

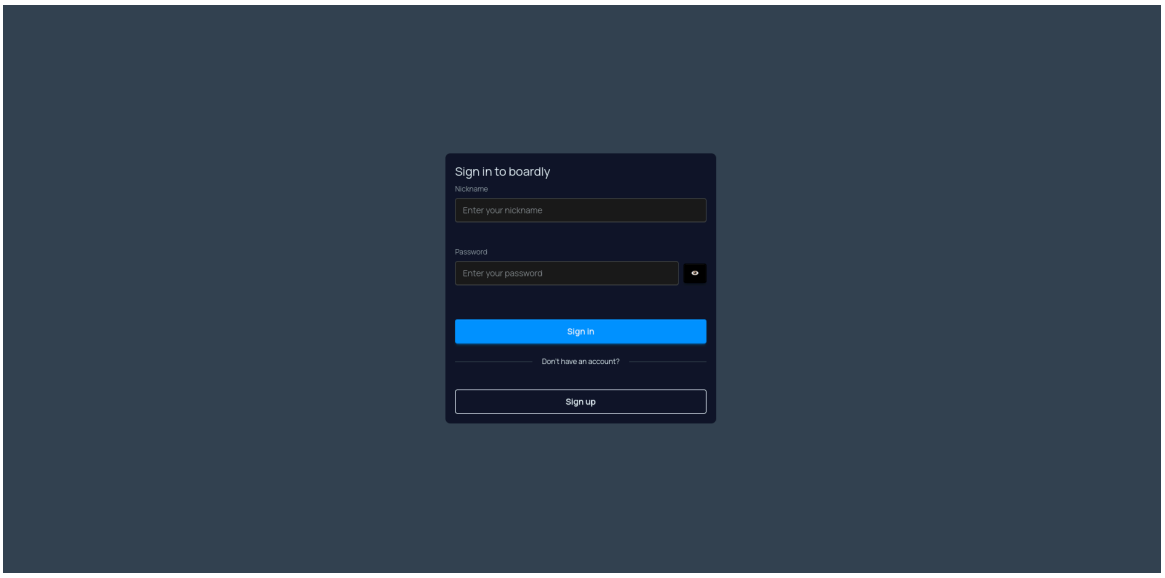
Boardly to aplikacja webowa służąca do kolaboracyjnego zarządzania projektami zgodnie z metodyką Kanban. Umożliwia tworzenie tablic projektowych, zarządzanie zadaniami w postaci kart oraz pracę zespołową w czasie rzeczywistym. Struktura aplikacji jest elastyczna i pozwala dopasować układ tablicy do potrzeb zespołu.

1.1. Zarządzanie kontem użytkownika

- **Rejestracja** – każdy użytkownik może założyć konto, podając unikalny nickname i hasło. System pilnuje, żeby nick się nie powtarzał.

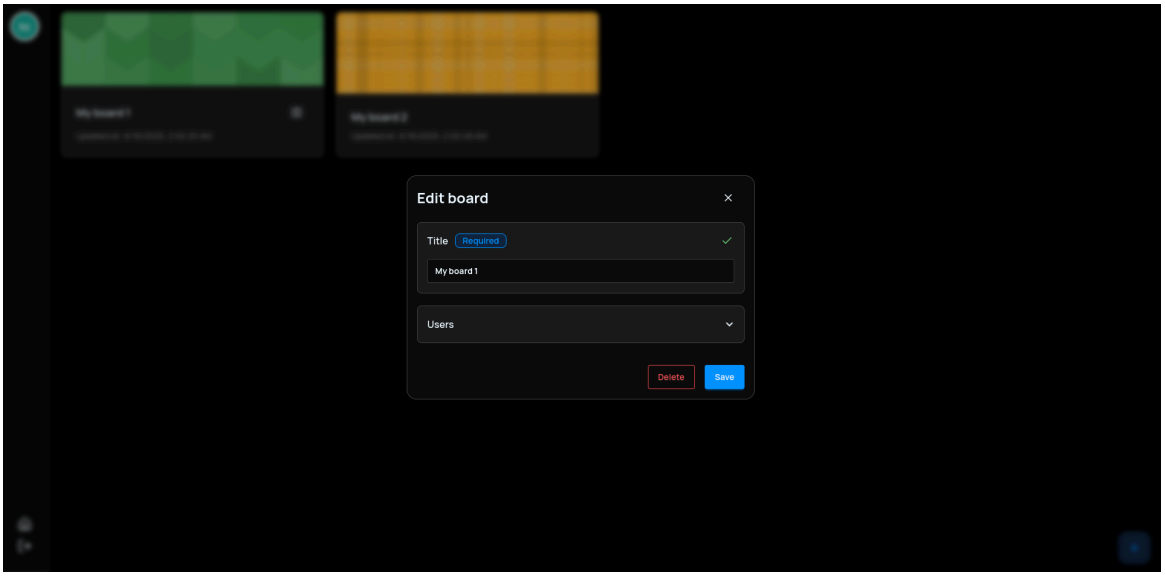


- **Logowanie** – po zalogowaniu użytkownik ma dostęp do swoich danych i tablic. Uwierzytelnianie działa w oparciu o **JWT**.

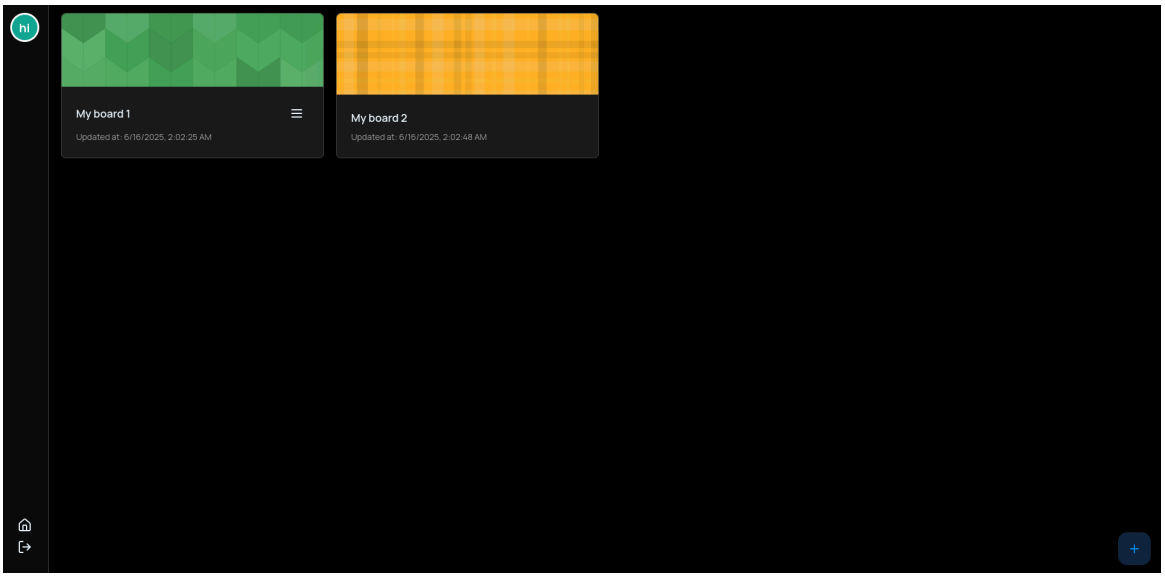


1.2. Zarządzanie tablicami

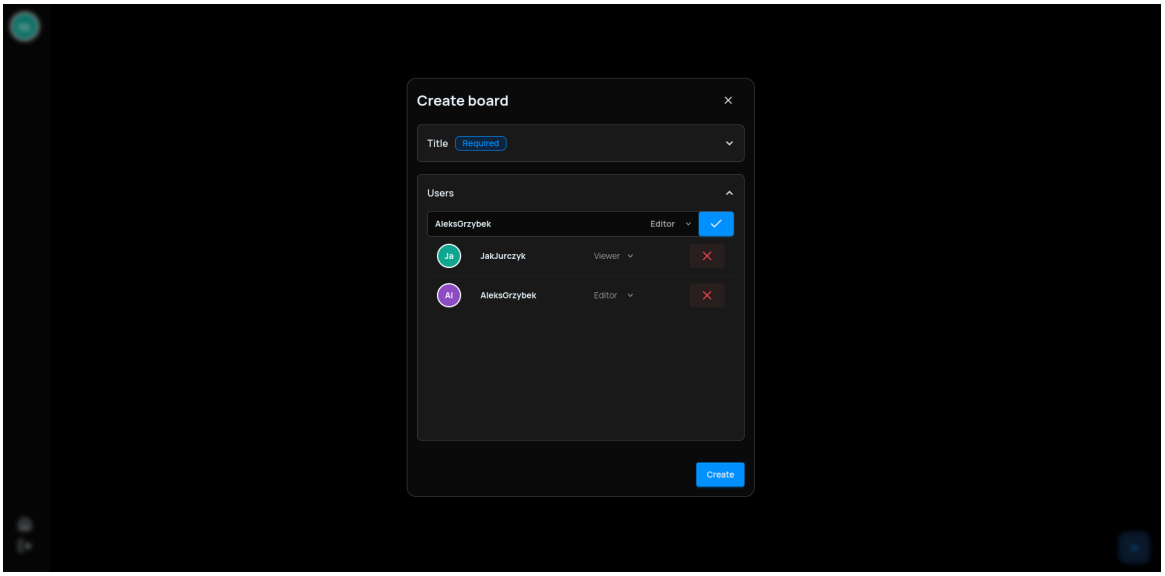
- **Tworzenie, edytowanie i usuwanie** – użytkownik może dodać nową tablicę lub edytować wcześniej przez niego stworzoną. Usunięcie tablicy usuwa też wszystkie powiązane dane (swimlanes, listy, karty).



- **Lista tablic** – po zalogowaniu użytkownik widzi wszystkie tablice, do których został przypisany.

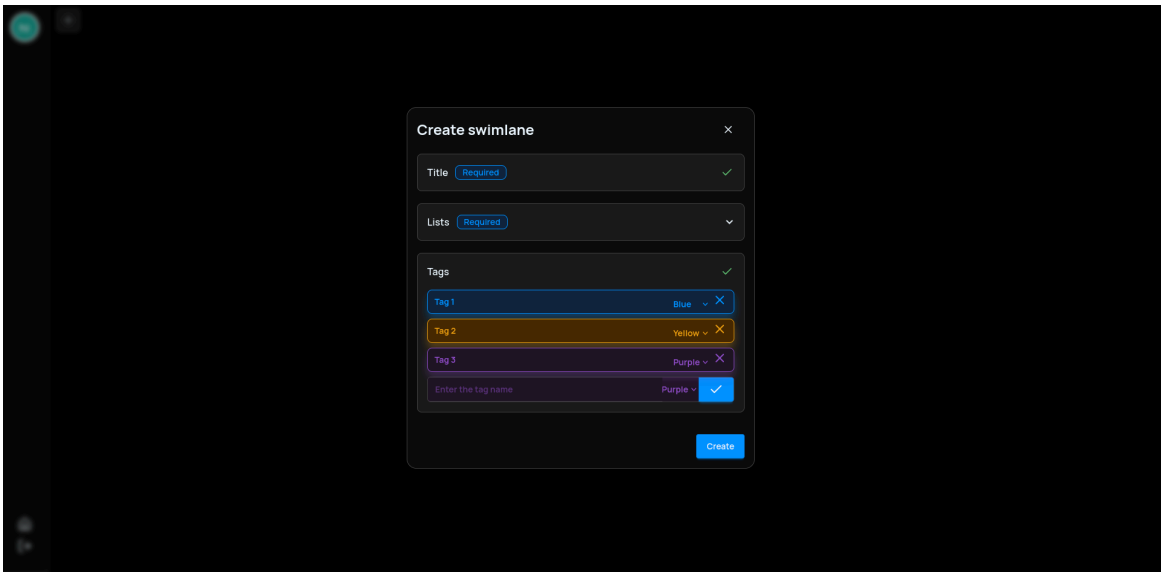


- **Członkowie tablicy** – możliwe jest dodawanie i usuwanie członków oraz zmiana ich ról i podgląd aktywności (kto aktualnie pracuje).

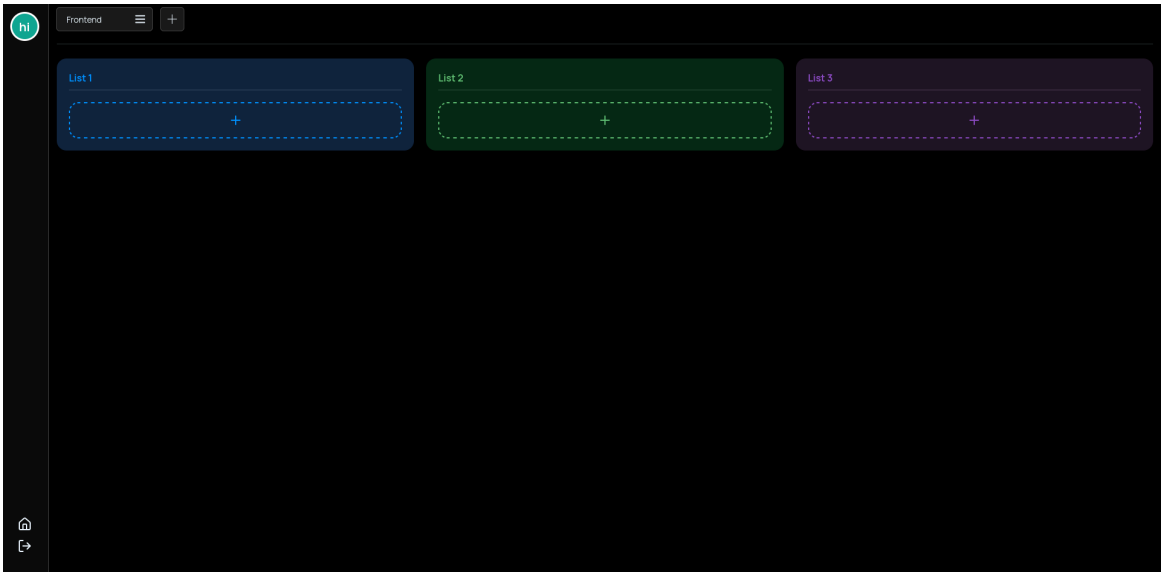


1.3. Struktura tablicy

- **Swimlanes (paski pływające)** – służą do grupowania zadań (np. „Funkcjonalności”, „Błędy”). Można je tworzyć, edytować i usuwać. Dla każdego swimlane można zdefiniować zestaw tagów, które będą dostępne do oznaczania kart w jego obrębie. Pozwala to lepiej kategoryzować i filtrować zadania.

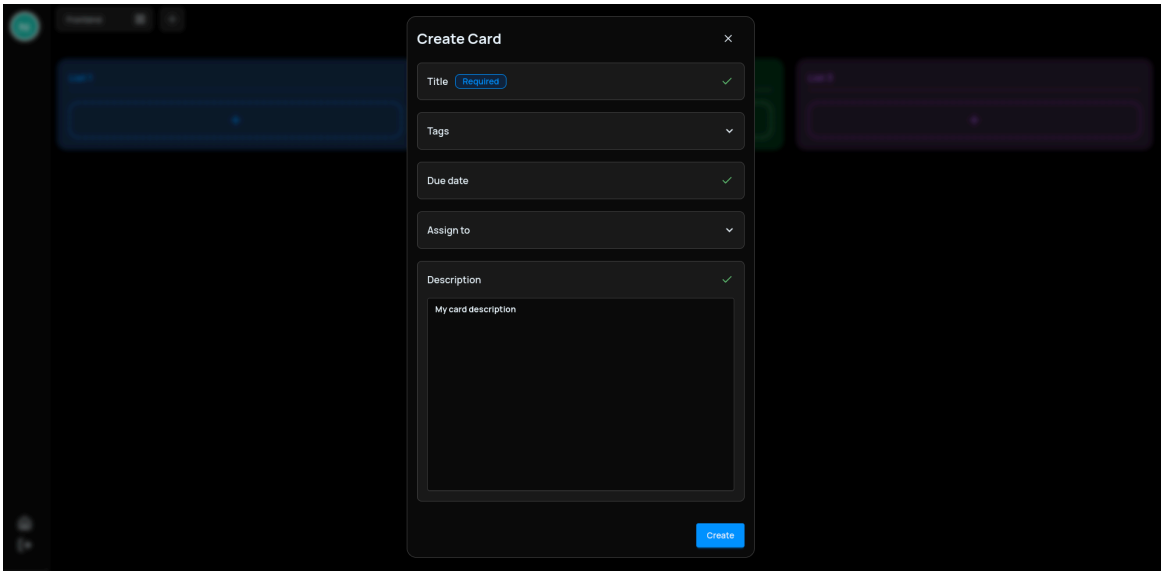


- **Listy** – znajdują się wewnątrz swimlane’ów. Oznaczają etapy pracy, np. „Do zrobienia”, „W trakcie”, „Gotowe”.

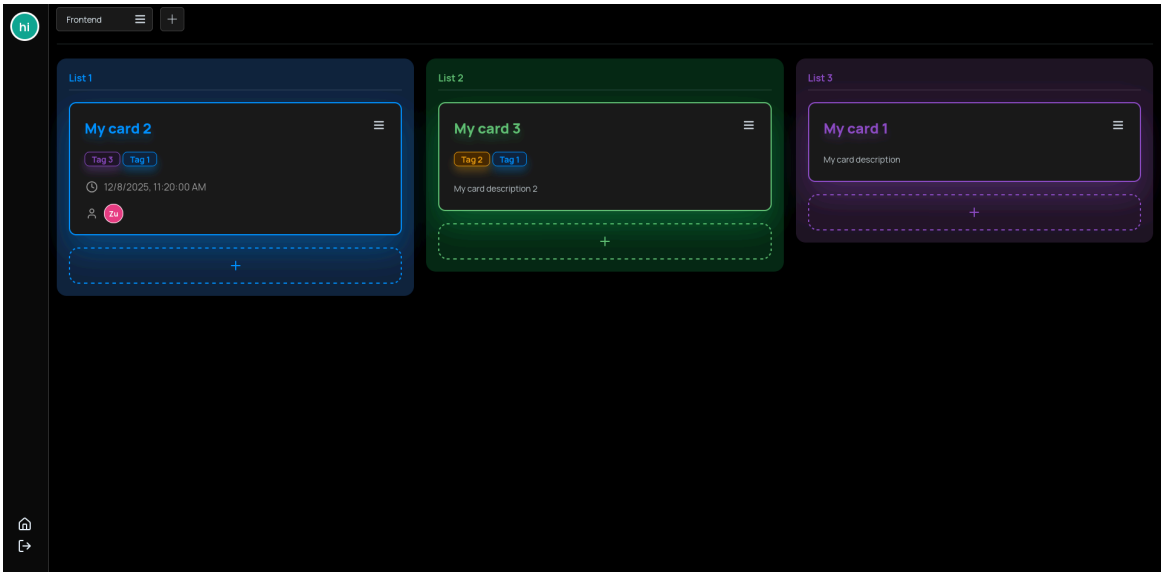


1.4. Zarządzanie zadaniami (Karty)

- **Dodawanie i usuwanie** – użytkownicy mogą tworzyć nowe karty i usuwać niepotrzebne.
- **Edycja** – każdą kartę można edytować: tytuł, opis, datę wykonania, tagi, przypisanych użytkowników.



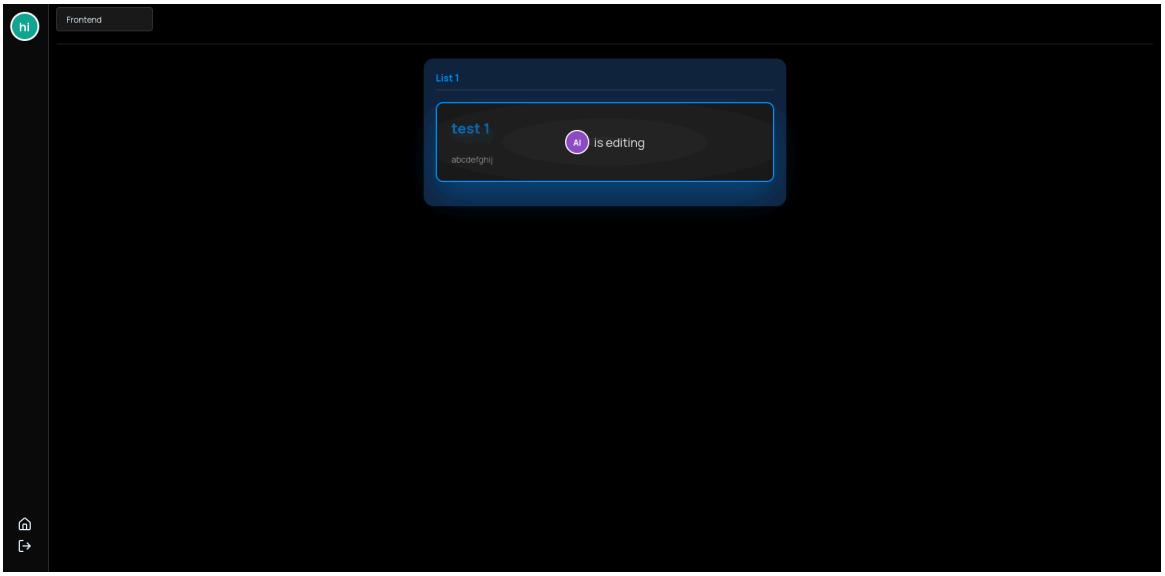
- **Przenoszenie** – karty można przeciągać między listami (drag and drop), co pozwala śledzić postęp pracy.



1.5. Współpraca w czasie rzeczywistym

Aplikacja została zbudowana z myślą o pracy zespołowej.

- **Aktualizacje w czasie rzeczywistym** – dzięki SignalR wszystkie zmiany wewnątrz tablicy są natychmiast widoczne dla pozostałych użytkowników bez odświeżania.
- **Blokowanie kart**: gdy ktoś edytuje kartę, jest ona blokowana dla innych by uniknąć konfliktów. Widać też kto aktualnie ją edytuje.



- **Ochrona przed konfliktami** (Optimistic Concurrency Control) - system śledzi wersje modyfikowanych obiektów. Jeśli użytkownik próbuje zapisać zmiany w obiekcie, który w międzyczasie został zmodyfikowany przez kogoś innego, operacja zostanie zablokowana, a użytkownik otrzyma stosowny komunikat.

2. Uprawnienia

Użytkownik może posiadać jeden z 4 stopni uprawnień. Są one indywidualne dla każdej tablicy i przechowywane w kolekcji boards. Są to kolejno:

- **Viewer** - najniższy stopień uprawnień, pozwala na:
  - Odczyt wszystkich pól z kolekcji boards i cards
  - Zmianę ListId kart, do których jest przypisany
- **Editor** - wszystkie powyższe uprawnienia, a poza tym:
  - Dodawanie, edycja i usuwanie kart, z dostępem do wszystkich ich pól
- **Admin** - wszystkie powyższe uprawnienia, a poza tym:
  - Dodawanie i usuwanie użytkowników, którzy nie mają roli **Owner**
  - Zmiana nazwy board
  - Dodawanie, usuwanie i edycja swimlane
  - Dodawanie, usuwanie i edycja list
  - Dodawanie, usuwanie i edycja tagów
- **Owner** - w jednej tablicy może być tylko jeden użytkownik o tej roli, posiada wszystkie powyższe uprawnienia, a ponadto:
  - Dodawanie i usuwanie tablicy
  - Przypisywanie roli **Owner** innemu użytkownikowi - wtedy obecny **Owner** traci uprawnienia

3. Wykorzystana technologia

Projekt podzielony jest na dwie aplikacje - Web oraz API.

- **Aplikacja Web**:
  - Język programowania: TypeScript
  - Framework: SvelteKit
  - Framework CSS: Tailwind CSS v4.1
  - Komunikacja w czasie rzeczywistym: SignalR (połączenie z API)zeczywistym
- **Aplikacja API**:
  - Język programowania: C#
  - Framework: ASP.NET
  - Wersja SDK: .NET 9.0
  - Komunikacja w czasie rzeczywistym: SignalR (połączenie z klientami)
  - Komunikacja z bazą danych: MongoDB C# Driver 3.4.0
- **Baza danych**:
  - System: MongoDB
  - Konfiguracja: Replica Set
  - Ustawienia połączenia:
    - Read concern: `snapshot`
    - Write concern: `majority`
    - Read preference: `primary`

4. Struktura bazy danych

4.1. Kolekcja tablic [boards]

```
{
  _id: ObjectId,
  title: string,
  members: [
    {
      userId: ObjectId,
      role: "Owner" | "Admin" | "Editor" | "Viewer",
      isActive: boolean
    },
    ...
  ],
  swimlanes: [
    {
      _id: ObjectId,
      title: string,
      color: "Red" | "Green" | "Blue" | "Purple" | "Yellow" | "Pink" | "Teal",
      tags: [
        {
          _id: ObjectId,
          title: string,
          color: "Red" | "Green" | "Blue" | "Purple" | "Yellow" | "Pink" | "Teal"
        },
        ...
      ],
      lists: [
        {
          _id: ObjectId,
          title: string,
          color: "Red" | "Green" | "Blue" | "Purple" | "Yellow" | "Pink" | "Teal",
          maxWIP: number | null
        },
        ...
      ]
    },
    ...
  ],
  createdAt: ISODate,
  updatedAt: ISODate
}
```

Dodatkowo:

- Tekstowy indeks na polu `title`, wspierający pełnotekstowe wyszukiwanie plansz po tytule.

4.2. Kolekcja kart [cards]

```
{
  _id: ObjectId,
  boardId: ObjectId,
  swimlaneId: ObjectId,
  listId: ObjectId,
  title: string,
  description: string | null,
  dueDate: ISODate | null,
  assignedUsers: ObjectId[],
  tags: ObjectId[],
  createdAt: ISODate,
  updatedAt: ISODate
}
```

Dodatkowo:
- Złożony indeks na `boardId`, `listId`, `swimlaneId`, przyspieszający filtrowanie kart przypisanych do konkretnego miejsca na tablicy.

4.3. Kolekcja użytkowników [users]

```
{
  _id: ObjectId,
  nickname: string,
  password: string,
  createdAt: ISODate
  updatedAt: ISODate
}
```

Dodatkowo:
- Tekstowy indeks na polu `nickname`, wspierający pełnotekstowe wyszukiwanie oraz wymuszający unikalność nicków użytkowników.

4.4. Kolekcja tokenów odświeżających [refreshTokens]

```
{
  _id: ObjectId,
  userId: ObjectId,
  token: string,
  expiresAt: ISODate
}
```

Dodatkowo:
- Indeks rosnący na polu `token`, zapewniający unikalność tokenów odświeżania.
- Indeks TTL na polu `expiresAt`, który automatycznie usuwa tokeny po ich wygaśnięciu (expireAfterSeconds: 0)

5. Zapytania bazodanowe

5.1. Tablice [boards]

5.1.1. Pobranie tablic po identyfikatorze użytkownika

```C#
public IQueryable<Board> GetRawBoardsByUserId(ObjectId userId, IClientSessionHandle? session = null)
{
    return (session == null
        ? _boardsCollection.AsQueryable()
        : _boardsCollection.AsQueryable(session))
        .Where(b => b.Members.Any(m => m.UserId == userId));
}
```

5.1.2. Pobranie tablic wraz z danymi członków po identyfikatorze użytkownika

```C#
public IQueryable<BoardWithUser> GetBoardsByUserId(ObjectId userId, IClientSessionHandle? session = null)
{
    return (session == null
        ? _boardsCollection.AsQueryable()
        : _boardsCollection.AsQueryable(session))
        .Where(b => b.Members.Any(m => m.UserId == userId))
        .SelectMany(x => x.Members.Select(member => new { Board = x, Member = member }))
        .Join(
            _usersCollection,
            x => x.Member.UserId,
            user => user.Id,
            (x, user) => new
            {
                x.Board,
                Member = new MemberWithUser
                {
                    UserId = x.Member.UserId,
                    Role = x.Member.Role,
                    IsActive = x.Member.IsActive,
                    Nickname = user.Nickname
                }
            }
        )
        .GroupBy(x => x.Board.Id)
        .Select(x => new BoardWithUser
        {
            Id = x.First().Board.Id,
            Title = x.First().Board.Title,
            CreatedAt = x.First().Board.CreatedAt,
            UpdatedAt = x.First().Board.UpdatedAt,
            Swimlanes = x.First().Board.Swimlanes,
```

```
32              Members = x.Select(m => m.Member),
33          });
34  }
```

5.1.3. Pobranie członków danej tablicy

```C#
public IQueryable<Member> GetBoardMembers(ObjectId boardId, IClientSessionHandle? session = null)
{
    return (session == null
        ? _boardsCollection.AsQueryable()
        : _boardsCollection.AsQueryable(session))
        .Where(b => b.Id == boardId)
        .SelectMany(b => b.Members);
}
```

5.1.4. Pobranie roli członka tablicy

```C#
public async Task<BoardRole?> GetUserBoardRoleAsync(ObjectId boardId, ObjectId userId,
    IClientSessionHandle? session = null, CancellationToken cancellationToken = default) =>
    (await GetBoardMembers(boardId, session).FirstOrDefaultAsync(m => m.UserId == userId,
    cancellationToken))?.Role;
```

5.1.5. Utworzenie tablicy

```C#
public async Task CreateBoardAsync(Board board, CancellationToken cancellationToken = default)
{
    if (board.Members.Count(x => x.Role == BoardRole.Owner) != 1)
        throw new ArgumentException("Board must have at least one member that is an owner.");
    await _boardsCollection.InsertOneAsync(board, cancellationToken: cancellationToken);
}
```

5.1.6. Zmiana aktywności członka tablicy

```C#
public async Task ChangeUserActivity(ObjectId id, ObjectId userId,
    bool isActive, CancellationToken cancellationToken = default)
{
    var filter = Builders<Board>.Filter.And(
        Builders<Board>.Filter.Eq(x => x.Id, id),
        Builders<Board>.Filter.ElemMatch(x=> x.Members, Builders<Member>.Filter.Eq(x =>
        x.UserId, userId)));

    var update = Builders<Board>.Update.Set("members.$.isActive", isActive);

    await _boardsCollection.UpdateOneAsync(filter, update, null, cancellationToken);
}
```

5.1.7. Aktualizacja tablicy

```C#
public async Task UpdateBoardAsync(Board board, ObjectId userId, IClientSessionHandle session, CancellationToken cancellationToken = default)
{
    IEnumerable<Member> owners = board.Members.Where(x => x.Role == BoardRole.Owner);
    if (owners.Count() != 1)
        throw new ArgumentException("Board must have exactly one member that is an owner.");
    ObjectId ownerId = owners.Single().UserId;
    var result = await _boardsCollection
        .AsQueryable(session)
        .Where(x => x.Id == board.Id)
        .Select(x => new
        {
            x.Members,
            x.UpdatedAt
        })
        .FirstOrDefaultAsync(cancellationToken)
        ?? throw new RecordDoesNotExist("Board has not been found.");
    if (board.UpdatedAt != default && board.UpdatedAt != result.UpdatedAt)
        throw new PreconditionFailedException("Board has been modified by another user since it was last read.");
    HashSet<Member> members = result.Members;
    BoardRole? role = members.FirstOrDefault(x => x.UserId == userId)?.Role;
    if (role == null || (role != BoardRole.Owner && role != BoardRole.Admin))
        throw new ForbiddenException("User is not authorized to update this board.");
    IEnumerable<Member> existingOwners = members.Where(x => x.Role == BoardRole.Owner);
    if (existingOwners.Count() != 1)
        throw new InvalidOperationException("An unexpected error occurred while updating board. Board has more or less than one owner!");
    ObjectId existingOwnerId = existingOwners.Single().UserId;
    if (existingOwnerId != ownerId && existingOwnerId != userId)
        throw new ForbiddenException("User is not authorized to change the owner of this board.");
    var newTime = DateTime.UtcNow;
    var filter = Builders<Board>.Filter.Eq(b => b.Id, board.Id);
    if (board.UpdatedAt != default)
        filter &= Builders<Board>.Filter.Eq(b => b.UpdatedAt, board.UpdatedAt);
    var update = Builders<Board>.Update
        .Set(b => b.Title, board.Title)
        .Set(b => b.Members, board.Members)
        .Set(b => b.UpdatedAt, newTime);
    UpdateResult updateResult = await _boardsCollection.UpdateOneAsync(session, filter, update, cancellationToken: cancellationToken);
    if (updateResult.MatchedCount == 0 && board.UpdatedAt != default)
        throw new PreconditionFailedException("Board has been modified by another user since it was last read.");
    else if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
```

```
41            throw new InvalidOperationException("An unexpected error occurred while updating
              board. Board has not been modified.");
42        board.UpdatedAt = newTime;
43    }
```

5.1.8. Aktualizacja tablicy wraz z jej zwrotem po aktualizacji z dołączonymi danymi członków

```C#
1   public async Task<BoardWithUser> UpdateAndFindBoardAsync(Board board, ObjectId userId,
    CancellationToken cancellationToken = default)
2   {
3       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
4       return await session.WithTransactionAsync(async (s, ct) =>
5       {
6           await UpdateBoardAsync(board, userId, s, ct);
7           return await GetBoardsByUserId(userId, s).SingleAsync(b => b.Id == board.Id, ct);
8       }, cancellationToken: cancellationToken);
9   }
```

5.1.9. Usunięcie tablicy

```C#
1   public async Task DeleteBoardAsync(ObjectId id, ObjectId userId, DateTime updatedAt =
    default, CancellationToken cancellationToken = default)
2   {
3       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
4       await session.WithTransactionAsync(async (s, ctx) =>
5       {
6           var result = await _boardsCollection
7               .AsQueryable(s)
8               .Where(x => x.Id == id)
9               .Select(x => new
10              {
11                  Member = x.Members.FirstOrDefault(m => m.UserId == userId),
12                  x.UpdatedAt
13              })
14              .FirstOrDefaultAsync(ctx)
15              ?? throw new RecordDoesNotExist("Board has not been found.");
16          if (updatedAt != default && updatedAt != result.UpdatedAt)
17              throw new PreconditionFailedException("Board has been modified by another user
                since it was last read.");
18          if (result.Member?.Role != BoardRole.Owner)
19              throw new ForbiddenException("User is not authorized to delete this board.");
20          var filter = Builders<Board>.Filter.Eq(b => b.Id, id);
21          if (updatedAt != default)
22              filter &= Builders<Board>.Filter.Eq(b => b.UpdatedAt, updatedAt);
23          DeleteResult deleteResult = await _boardsCollection.DeleteOneAsync(s, filter,
            cancellationToken: ctx);
24          if (deleteResult.DeletedCount == 0 && updatedAt != default)
```

```
25                 throw new PreconditionFailedException("Board has been modified by another user
                   since it was last read.");
26             await _cardsCollection.DeleteManyAsync(s, x => x.BoardId == id, cancellationToken:
               ctx);
27             return Task.CompletedTask;
28         }, cancellationToken: cancellationToken);
29     }
```

5.2. Paski pływające [swimlanes]

5.2.1. Pobranie pasków wszystkich pasków pływających danej tablicy

```
1  public IQueryable<Swimlane> GetSwimlanesByBoardId(ObjectId boardId, IClientSessionHandle?
   session = null)                                                                          C#
2  {
3      return (session == null
4          ? _boardsCollection.AsQueryable()
5          : _boardsCollection.AsQueryable(session))
6          .Where(b => b.Id == boardId)
7          .SelectMany(b => b.Swimlanes);
8  }
```

5.2.2. Utworzenie paska pływającego

```
1  public async Task<DateTime> CreateSwimlaneAsync(ObjectId boardId, ObjectId userId,
   Swimlane swimlane,                                                                       C#
2      IClientSessionHandle session, DateTime updatedAt = default, CancellationToken
       cancellationToken = default)
3  {
4      var result = await _boardsCollection
5          .AsQueryable(session)
6          .Where(x => x.Id == boardId)
7          .Select(x => new
8          {
9              Member = x.Members.FirstOrDefault(m => m.UserId == userId),
10             x.UpdatedAt
11         })
12         .FirstOrDefaultAsync(cancellationToken)
13         ?? throw new RecordDoesNotExist("Board has not been found.");
14     if (updatedAt != default && updatedAt != result.UpdatedAt)
15         throw new PreconditionFailedException("Board has been modified by another user since
           it was last read.");
16     if (result.Member?.Role != BoardRole.Owner && result.Member?.Role != BoardRole.Admin)
17         throw new ForbiddenException("User is not authorized to create a swimlane on this
           board.");
18     var filter = Builders<Board>.Filter.Eq(b => b.Id, boardId);
19     if (updatedAt != default)
20         filter &= Builders<Board>.Filter.Eq(b => b.UpdatedAt, updatedAt);
21     var now = DateTime.UtcNow;
22     var update = Builders<Board>.Update
```

```
23          .Push(b => b.Swimlanes, swimlane)
24          .Set(b => b.UpdatedAt, now);
25      UpdateResult updateResult = await _boardsCollection.UpdateOneAsync(session, filter,
        update, cancellationToken: cancellationToken);
26      if (updatedAt != default && updateResult.MatchedCount == 0)
27          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
28      if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
29          throw new InvalidOperationException("An unexpected error occurred while creating
            swimlane. Board has not been modified.");
30      return now;
31  }
```

5.2.3. Utworzenie paska pływającego wraz z jego zwrotem po utworzeniu

```C#
1   public async Task<(Swimlane, DateTime)> CreateAndFindSwimlaneAsync(ObjectId boardId,
    ObjectId userId,
2       Swimlane swimlane, DateTime updatedAt = default, CancellationToken cancellationToken =
        default)
3   {
4       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
5       return await session.WithTransactionAsync(async (s, ctx) =>
6       {
7           var newUpdatedAt = await CreateSwimlaneAsync(boardId, userId, swimlane, s, updatedAt,
            ctx);
8           return (await GetSwimlanesByBoardId(boardId, s).SingleAsync(s => s.Id == swimlane.Id,
            ctx), newUpdatedAt);
9       }, cancellationToken: cancellationToken);
10  }
```

5.2.4. Aktualizacja paska pływającego

```C#
1   public async Task<DateTime> UpdateSwimlaneAsync(ObjectId boardId, ObjectId userId,
    Swimlane swimlane,
2       IClientSessionHandle session, DateTime updatedAt = default, CancellationToken
        cancellationToken = default)
3   {
4       var result = await _boardsCollection
5           .AsQueryable(session)
6           .Where(b => b.Id == boardId)
7           .Select(b => new
8           {
9               Member = b.Members.FirstOrDefault(m => m.UserId == userId),
10              Swimlane = b.Swimlanes.FirstOrDefault(s => s.Id == swimlane.Id),
11              b.UpdatedAt
12          })
13          .FirstOrDefaultAsync(cancellationToken)
14          ?? throw new RecordDoesNotExist("Board has not been found.");
15      if (updatedAt != default && updatedAt != result.UpdatedAt)
```

```
16          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
17      if (result.Member == null)
18          throw new ForbiddenException("User is not a member of this board.");
19      if (result.Member.Role != BoardRole.Owner && result.Member.Role != BoardRole.Admin)
20          throw new ForbiddenException("User is not authorized to update a swimlane on this
            board.");
21      if (result.Swimlane == null)
22          throw new RecordDoesNotExist("Swimlane has not been found.");
23      var filter = Builders<Board>.Filter.Eq(b => b.Id, boardId);
24      if (updatedAt != default)
25          filter &= Builders<Board>.Filter.Eq(b => b.UpdatedAt, updatedAt);
26      var now = DateTime.UtcNow;
27      var update = Builders<Board>.Update
28          .Set("swimlanes.$[sw].title", swimlane.Title)
29          .Set("swimlanes.$[sw].color", swimlane.Color)
30          .Set(b => b.UpdatedAt, now);
31      var arrayFilters = new List<ArrayFilterDefinition>
32      {
33          new BsonDocumentArrayFilterDefinition<BsonDocument>(new BsonDocument("sw._id",
            swimlane.Id)),
34      };
35      var updateOptions = new UpdateOptions { ArrayFilters = arrayFilters };
36      UpdateResult updateResult = await _boardsCollection.UpdateOneAsync(session, filter,
        update, updateOptions, cancellationToken);
37      if (updatedAt != default && updateResult.MatchedCount == 0)
38          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
39      if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
40          throw new InvalidOperationException("An unexpected error occurred while updating
            swimlane. Board has not been modified.");
41      return now;
42  }
```

5.2.5. Aktualizacja paska pływającego wraz z jego zwrotem po aktualizacji

```
1   public async Task<(Swimlane, DateTime)> UpdateAndFindSwimlaneAsync(ObjectId boardId,
    ObjectId userId,                                                                    C#
2       Swimlane swimlane, DateTime updatedAt = default, CancellationToken cancellationToken =
        default)
3   {
4       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
5       return await session.WithTransactionAsync(async (s, ctx) =>
6       {
7           var newUpdatedAt = await UpdateSwimlaneAsync(boardId, userId, swimlane, s, updatedAt,
            ctx);
8           return (await GetSwimlanesByBoardId(boardId, s).SingleAsync(s => s.Id == swimlane.Id,
            ctx), newUpdatedAt);
```

```
9       }, cancellationToken: cancellationToken);
10  }
```

5.2.6. Usunięcie paska pływającego

```C#
1   public async Task<DateTime> DeleteSwimlaneAsync(ObjectId boardId, ObjectId swimlaneId,
2       ObjectId userId, DateTime updatedAt = default, CancellationToken cancellationToken =
        default)
3   {
4       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
5       return await session.WithTransactionAsync(async (s, ctx) =>
6       {
7           var result = await _boardsCollection
8               .AsQueryable(s)
9               .Where(b => b.Id == boardId)
10              .Select(b => new
11              {
12                  Swimlane = b.Swimlanes.FirstOrDefault(s => s.Id == swimlaneId),
13                  Member = b.Members.FirstOrDefault(m => m.UserId == userId),
14                  b.UpdatedAt
15              })
16              .FirstOrDefaultAsync(ctx)
17              ?? throw new RecordDoesNotExist("Board has not been found.");
18          if (updatedAt != default && updatedAt != result.UpdatedAt)
19              throw new PreconditionFailedException("Board has been modified by another user
                since it was last read.");
20          if (result.Member?.Role != BoardRole.Owner && result.Member?.Role != BoardRole.Admin)
21              throw new ForbiddenException("User is not authorized to delete this swimlane.");
22          if (result.Swimlane == null)
23              throw new RecordDoesNotExist("Swimlane has not been found.");
24          var filter = Builders<Board>.Filter.Eq(b => b.Id, boardId);
25          if (updatedAt != default)
26              filter &= Builders<Board>.Filter.Eq(b => b.UpdatedAt, updatedAt);
27          var now = DateTime.UtcNow;
28          var update = Builders<Board>.Update
29              .Set(b => b.UpdatedAt, now)
30              .PullFilter(b => b.Swimlanes, Builders<Swimlane>.Filter.Eq(s => s.Id, swimlaneId)
31          );
32          UpdateResult updateResult = await _boardsCollection.UpdateOneAsync(s, filter, update,
            cancellationToken: ctx);
33          if (updatedAt != default && updateResult.MatchedCount == 0)
34              throw new PreconditionFailedException("Board has been modified by another user
                since it was last read.");
35          if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
36              throw new InvalidOperationException("An unexpected error occurred while deleting
                swimlane. Board has not been modified.");
```

```
37          await _cardsCollection.DeleteManyAsync(s, x => x.SwimlaneId == swimlaneId,
            cancellationToken: ctx);
38          return now;
39      }, cancellationToken: cancellationToken);
40  }
```

5.3. Listy [lists]

5.3.1. Pobranie wszystkich list paska pływającego

```C#
1  public IQueryable<List> GetListsBySwimlaneId(ObjectId boardId, ObjectId swimlaneId,
   IClientSessionHandle? session = null)
2  {
3      return (session == null
4          ? _boardsCollection.AsQueryable()
5          : _boardsCollection.AsQueryable(session))
6          .Where(b => b.Id == boardId)
7          .SelectMany(b => b.Swimlanes.Where(s => s.Id == swimlaneId).SelectMany(s => s.Lists));
8  }
```

5.3.2. Utworzenie nowej listy

```C#
1   public async Task<DateTime> CreateListAsync(ObjectId boardId, ObjectId swimlaneId,
    ObjectId userId, List list,
2       IClientSessionHandle session, DateTime updatedAt = default, CancellationToken
        cancellationToken = default)
3   {
4       var result = await _boardsCollection
5           .AsQueryable(session)
6           .Where(b => b.Id == boardId)
7           .Select(b => new
8           {
9               Member = b.Members.FirstOrDefault(m => m.UserId == userId),
10              b.UpdatedAt
11          })
12          .FirstOrDefaultAsync(cancellationToken)
13          ?? throw new RecordDoesNotExist("Board has not been found.");
14      if (updatedAt != default && updatedAt != result.UpdatedAt)
15          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
16      if (result.Member?.Role != BoardRole.Owner && result.Member?.Role != BoardRole.Admin)
17          throw new ForbiddenException("User is not authorized to create a list on this
            board.");
18
19      var filter = Builders<Board>.Filter.Eq(b => b.Id, boardId);
20      if (updatedAt != default)
21          filter &= Builders<Board>.Filter.Eq(b => b.UpdatedAt, updatedAt);
22      var now = DateTime.UtcNow;
23      var update = Builders<Board>.Update
24          .Push("swimlanes.$[sw].lists", list)
```

```
25          .Set(b => b.UpdatedAt, now);
26
27      var arrayFilters = new List<ArrayFilterDefinition>
28      {
29          new BsonDocumentArrayFilterDefinition<BsonDocument>(new BsonDocument("sw._id",
            swimlaneId))
30      };
31      var updateOptions = new UpdateOptions { ArrayFilters = arrayFilters };
32
33      var updateResult = await _boardsCollection.UpdateOneAsync(session, filter, update,
        updateOptions, cancellationToken);
34
35      if (updatedAt != default && updateResult.MatchedCount == 0)
36          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
37      if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
38          throw new InvalidOperationException("An unexpected error occurred while creating list.
            Board has not been modified.");
39      return now;
40  }
```

5.3.3. Utworzenie nowej listy wraz z jej zwrotem po utworzeniu

```C#
1   public async Task<(List, DateTime)> CreateAndFindListAsync(ObjectId boardId, ObjectId
    swimlaneId, ObjectId userId,
2       List list, DateTime updatedAt = default, CancellationToken cancellationToken = default)
3   {
4       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
5       return await session.WithTransactionAsync(async (s, ctx) =>
6       {
7           var newUpdatedAt = await CreateListAsync(boardId, swimlaneId, userId, list, s,
            updatedAt, ctx);
8           return (await GetListBySwimlaneId(boardId, swimlaneId, s).SingleAsync(s => s.Id ==
            list.Id, ctx), newUpdatedAt);
9       }, cancellationToken: cancellationToken);
10  }
```

5.3.4. Aktualizacja listy

```C#
1   public async Task<DateTime> UpdateListAsync(ObjectId boardId, ObjectId swimlaneId,
    ObjectId userId, List list,
2       IClientSessionHandle session, DateTime updatedAt = default, CancellationToken
        cancellationToken = default)
3   {
4       var result = await _boardsCollection
5           .AsQueryable(session)
6           .Where(b => b.Id == boardId)
7           .Select(b => new
8           {
```

```
9               Member = b.Members.FirstOrDefault(m => m.UserId == userId),
10              List = b.Swimlanes.Where(s => s.Id == swimlaneId).Select(s =>
                s.Lists.FirstOrDefault(l => l.Id == list.Id)).FirstOrDefault(),
11              b.UpdatedAt
12          }).FirstOrDefaultAsync(cancellationToken)
13          ?? throw new RecordDoesNotExist("Board has not been found.");
14      if (updatedAt != default && updatedAt != result.UpdatedAt)
15          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
16      if (result.Member?.Role != BoardRole.Owner && result.Member?.Role != BoardRole.Admin)
17          throw new ForbiddenException("User is not authorized to update a list on this
            board.");
18      if (result.List == null)
19          throw new RecordDoesNotExist("List has not been found.");
20      var filter = Builders<Board>.Filter.Eq(b => b.Id, boardId);
21      if (updatedAt != default)
22          filter &= Builders<Board>.Filter.Eq(b => b.UpdatedAt, updatedAt);
23      var now = DateTime.UtcNow;
24      var update = Builders<Board>.Update
25          .Set("swimlanes.$[sw].lists.$[lst].title", list.Title)
26          .Set("swimlanes.$[sw].lists.$[lst].color", list.Color)
27          .Set("swimlanes.$[sw].lists.$[lst].maxWIP", list.MaxWIP)
28          .Set(b => b.UpdatedAt, now);
29      var arrayFilters = new List<ArrayFilterDefinition>
30      {
31          new BsonDocumentArrayFilterDefinition<BsonDocument>(new BsonDocument("sw._id",
            swimlaneId)),
32          new BsonDocumentArrayFilterDefinition<BsonDocument>(new BsonDocument("lst._id",
            list.Id))
33      };
34      var updateOptions = new UpdateOptions { ArrayFilters = arrayFilters };
35      var updateResult = await _boardsCollection.UpdateOneAsync(session, filter, update,
        updateOptions, cancellationToken);
36      if (updatedAt != default && updateResult.MatchedCount == 0)
37          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
38      if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
39          throw new InvalidOperationException("An unexpected error occurred while updating list.
            Board has not been modified.");
40      return now;
41  }
```

5.3.5. Aktualizacja listy wraz z jej zwrotem po aktualizacji

```C#
1   public async Task<(List, DateTime)> UpdateAndFindListAsync(ObjectId boardId, ObjectId
    swimlaneId, ObjectId userId,
2       List list, DateTime updatedAt = default, CancellationToken cancellationToken = default)
3   {
```

```
    using var session = await _mongoClient.StartSessionAsync(cancellationToken:
    cancellationToken);
    return await session.WithTransactionAsync(async (s, ctx) =>
    {
        var newUpdatedAt = await UpdateListAsync(boardId, swimlaneId, userId, list, s,
        updatedAt, ctx);
        return (await GetListBySwimlaneId(boardId, swimlaneId, s).SingleAsync(s => s.Id ==
        list.Id, ctx), newUpdatedAt);
    }, cancellationToken: cancellationToken);
}
```

5.3.6. Usunięcie listy

```C#
public async Task<DateTime> DeleteListAsync(ObjectId boardId, ObjectId swimlaneId,
ObjectId listId, ObjectId userId,
    DateTime updatedAt = default, CancellationToken cancellationToken = default)
{
    using var session = await _mongoClient.StartSessionAsync(cancellationToken:
    cancellationToken);
    return await session.WithTransactionAsync(async (s, ctx) =>
    {
        var result = await _boardsCollection
            .AsQueryable(s)
            .Where(b => b.Id == boardId)
            .Select(b => new
            {
                List = b.Swimlanes.Where(s => s.Id == swimlaneId).Select(s =>
                s.Lists.FirstOrDefault(l => l.Id == listId)).FirstOrDefault(),
                Member = b.Members.FirstOrDefault(m => m.UserId == userId),
                b.UpdatedAt
            })
            .FirstOrDefaultAsync(ctx)
            ?? throw new RecordDoesNotExist("Board has not been found.");
        if (updatedAt != default && updatedAt != result.UpdatedAt)
            throw new PreconditionFailedException("Board has been modified by another user
            since it was last read.");
        if (result.Member?.Role != BoardRole.Owner && result.Member?.Role != BoardRole.Admin)
            throw new ForbiddenException("User is not authorized to delete this list.");
        if (result.List == null)
            throw new RecordDoesNotExist("List has not been found.");
        var filter = Builders<Board>.Filter.Eq(b => b.Id, boardId);
        if (updatedAt != default)
            filter &= Builders<Board>.Filter.Eq(b => b.UpdatedAt, updatedAt);
        var now = DateTime.UtcNow;
        var update = Builders<Board>.Update.PullFilter(
            "swimlanes.$[sw].lists",
            Builders<BsonDocument>.Filter.Eq("_id", listId))
            .Set(b => b.UpdatedAt, now);
```

```
32          var arrayFilters = new List<ArrayFilterDefinition>
33          {
34              new BsonDocumentArrayFilterDefinition<BsonDocument>(new BsonDocument("sw._id",
                swimlaneId))
35          };
36          var updateOptions = new UpdateOptions { ArrayFilters = arrayFilters };
37          UpdateResult updateResult = await _boardsCollection.UpdateOneAsync(s, filter, update,
                updateOptions, ctx);
38          if (updatedAt != default && updateResult.MatchedCount == 0)
39              throw new PreconditionFailedException("Board has been modified by another user
                since it was last read.");
40          if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
41              throw new InvalidOperationException("An unexpected error occurred while deleting
                list. Board has not been modified.");
42          await _cardsCollection.DeleteManyAsync(s, x => x.ListId == listId, cancellationToken:
                ctx);
43          return now;
44      }, cancellationToken: cancellationToken);
45  }
```

5.4. Tagi [tags]

5.4.1. Pobranie wszystkich tagów z paska pływającego

```C#
1  public IQueryable<Entities.Board.Tag> GetTagsBySwimlaneIdAsync(ObjectId boardId, ObjectId
   swimlaneId, IClientSessionHandle? session = null)
2  {
3      return (session == null
4          ? _boardsCollection.AsQueryable()
5          : _boardsCollection.AsQueryable(session))
6          .Where(b => b.Id == boardId)
7          .SelectMany(b => b.Swimlanes
8              .Where(s => s.Id == swimlaneId)
9              .SelectMany(s => s.Tags));
10 }
```

5.4.2. Utworzenie nowego taga

```C#
1  public async Task<DateTime> CreateTagAsync(ObjectId boardId, ObjectId swimlaneId, ObjectId
   userId, Entities.Board.Tag tag,
2      IClientSessionHandle session, DateTime updatedAt = default, CancellationToken
       cancellationToken = default)
3  {
4      var result = await _boardsCollection
5          .AsQueryable(session)
6          .Where(b => b.Id == boardId)
7          .Select(x => new
8          {
9              Member = x.Members.FirstOrDefault(m => m.UserId == userId),
10             x.UpdatedAt
```

```
11          })
12          .FirstOrDefaultAsync(cancellationToken)
13          ?? throw new RecordDoesNotExist("Board has not been found.");
14      if (updatedAt != default && updatedAt != result.UpdatedAt)
15          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
16      if (result.Member?.Role != BoardRole.Owner && result.Member?.Role != BoardRole.Admin)
17          throw new ForbiddenException("User is not authorized to create a tag on this board.");
18      var filter = Builders<Board>.Filter.Eq(b => b.Id, boardId);
19      if (updatedAt != default)
20          filter &= Builders<Board>.Filter.Eq(b => b.UpdatedAt, updatedAt);
21      var now = DateTime.UtcNow;
22      var update = Builders<Board>.Update
23          .Push("swimlanes.$[sw].tags", tag)
24          .Set(b => b.UpdatedAt, now);
25
26      var arrayFilters = new List<ArrayFilterDefinition>
27      {
28          new BsonDocumentArrayFilterDefinition<BsonDocument>(new BsonDocument("sw._id",
            swimlaneId))
29      };
30      var updateOptions = new UpdateOptions { ArrayFilters = arrayFilters };
31
32      var updateResult = await _boardsCollection.UpdateOneAsync(session, filter, update,
        updateOptions, cancellationToken);
33
34      if (updatedAt != default && updateResult.MatchedCount == 0)
35          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
36      if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
37          throw new InvalidOperationException("An unexpected error occurred while creating tag.
            Board has not been modified.");
38      return now;
39  }
```

5.4.3. Utworzenie nowego taga wraz z jego zwrotem po utworzeniu

```C#
1   public async Task<(Entities.Board.Tag, DateTime)> CreateAndFindTagAsync(ObjectId boardId,
    ObjectId swimlaneId,
2       ObjectId userId, Entities.Board.Tag tag, DateTime updatedAt = default, CancellationToken
        cancellationToken = default)
3   {
4       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
5       return await session.WithTransactionAsync(async (s, ctx) =>
6       {
7           var newUpdatedAt = await CreateTagAsync(boardId, swimlaneId, userId, tag, s,
            updatedAt, ctx);
```

```
8           return (await GetTagsBySwimlaneIdAsync(boardId, swimlaneId, s).SingleAsync(t => t.Id
            == tag.Id, ctx), newUpdatedAt);
9       }, cancellationToken: cancellationToken);
10  }
```

5.4.4. Aktualizacja taga

```C#
1   public async Task<DateTime> UpdateTagAsync(ObjectId boardId, ObjectId swimlaneId, ObjectId
    userId, Entities.Board.Tag tag,
2       IClientSessionHandle session, DateTime updatedAt = default, CancellationToken
        cancellationToken = default)
3   {
4       var result = await _boardsCollection
5           .AsQueryable(session)
6           .Where(b => b.Id == boardId)
7           .Select(x => new
8           {
9               Member = x.Members.FirstOrDefault(x => x.UserId == userId),
10              Swimlane = x.Swimlanes.FirstOrDefault(x => x.Id == swimlaneId),
11              x.UpdatedAt
12          }).FirstOrDefaultAsync(cancellationToken)
13          ?? throw new RecordDoesNotExist("Board has not been found.");
14      if (updatedAt != default && updatedAt != result.UpdatedAt)
15          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
16      if (result.Member?.Role != BoardRole.Owner && result.Member?.Role != BoardRole.Admin)
17          throw new ForbiddenException("User is not authorized to update a tag on this board.");
18      if (result.Swimlane == null)
19          throw new RecordDoesNotExist("Swimlane has not been found.");
20      var filter = Builders<Board>.Filter.Eq(b => b.Id, boardId);
21      if (updatedAt != default)
22          filter &= Builders<Board>.Filter.Eq(b => b.UpdatedAt, updatedAt);
23      var now = DateTime.UtcNow;
24      var update = Builders<Board>.Update
25          .Set("swimlanes.$[sw].tags.$[tg].title", tag.Title)
26          .Set("swimlanes.$[sw].tags.$[tg].color", tag.Color)
27          .Set(b => b.UpdatedAt, now);
28      var arrayFilters = new List<ArrayFilterDefinition>
29      {
30          new BsonDocumentArrayFilterDefinition<BsonDocument>(new BsonDocument("sw._id",
            swimlaneId)),
31          new BsonDocumentArrayFilterDefinition<BsonDocument>(new BsonDocument("tg._id",
            tag.Id))
32      };
33      var updateOptions = new UpdateOptions { ArrayFilters = arrayFilters };
34      var updateResult = await _boardsCollection.UpdateOneAsync(session, filter, update,
        updateOptions, cancellationToken);
35      if (updatedAt != default && updateResult.MatchedCount == 0)
```

```
36          throw new PreconditionFailedException("Board has been modified by another user since
            it was last read.");
37      if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
38          throw new InvalidOperationException("An unexpected error occurred while updating tag.
            Board has not been modified.");
39      return now;
40  }
```

5.4.5. Aktualizacja taga wraz z jego zwrotem po zaaktualizowaniu

```
1   public async Task<(Entities.Board.Tag, DateTime)> UpdateAndFindTagAsync(ObjectId boardId,
    ObjectId swimlaneId,                                                                    C#
2       ObjectId userId, Entities.Board.Tag tag, DateTime updatedAt = default, CancellationToken
        cancellationToken = default)
3   {
4       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
5       return await session.WithTransactionAsync(async (s, ctx) =>
6       {
7           var newUpdatedAt = await UpdateTagAsync(boardId, swimlaneId, userId, tag, s,
            updatedAt, ctx);
8           return (await GetTagsBySwimlaneIdAsync(boardId, swimlaneId, s).SingleAsync(t => t.Id
            == tag.Id, ctx), newUpdatedAt);
9       }, cancellationToken: cancellationToken);
10  }
```

5.4.6. Usunięcie taga

```
1   public async Task<DateTime> DeleteTagAsync(ObjectId boardId, ObjectId swimlaneId, ObjectId
    tagId, ObjectId userId,                                                                 C#
2       DateTime updatedAt = default, CancellationToken cancellationToken = default)
3   {
4       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
5       return await session.WithTransactionAsync(async (s, ctx) =>
6       {
7           var result = await _boardsCollection
8               .AsQueryable(s)
9               .Where(x => x.Id == boardId)
10              .Select(x => new
11              {
12                  Tag = x.Swimlanes.Where(s => s.Id == swimlaneId).Select(s =>
                    s.Tags.FirstOrDefault(t => t.Id == tagId)).FirstOrDefault(),
13                  Member = x.Members.FirstOrDefault(m => m.UserId == userId),
14                  x.UpdatedAt
15              })
16              .FirstOrDefaultAsync(ctx)
17              ?? throw new RecordDoesNotExist("Board has not been found.");
18          if (updatedAt != default && updatedAt != result.UpdatedAt)
```

```
19              throw new PreconditionFailedException("Board has been modified by another user
                since it was last read.");
20          if (result.Member?.Role != BoardRole.Owner && result.Member?.Role != BoardRole.Admin)
21              throw new ForbiddenException("User is not authorized to delete this tag.");
22          if (result.Tag == null)
23              throw new RecordDoesNotExist("Tag has not been found.");
24          var filter = Builders<Board>.Filter.Eq(b => b.Id, boardId);
25          var now = DateTime.UtcNow;
26          var update = Builders<Board>.Update.PullFilter(
27              "swimlanes.$[sw].tags",
28              Builders<BsonDocument>.Filter.Eq("_id", tagId))
29              .Set(b => b.UpdatedAt, now);
30          var arrayFilters = new List<ArrayFilterDefinition>
31          {
32              new BsonDocumentArrayFilterDefinition<BsonDocument>(new BsonDocument("sw._id",
                swimlaneId))
33          };
34          var updateOptions = new UpdateOptions { ArrayFilters = arrayFilters };
35          var updateResult = await _boardsCollection.UpdateOneAsync(s, filter, update,
            updateOptions, ctx);
36          if (updatedAt != default && updateResult.MatchedCount == 0)
37              throw new PreconditionFailedException("Board has been modified by another user
                since it was last read.");
38          if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
39              throw new InvalidOperationException("An unexpected error occurred while deleting
                tag. Board has not been modified.");
40
41          var cardFilter = Builders<Card>.Filter.AnyEq(c => c.Tags, tagId);
42          var cardUpdate = Builders<Card>.Update.Pull(c => c.Tags, tagId);
43          await _cardsCollection.UpdateManyAsync(s, cardFilter, cardUpdate, cancellationToken:
            ctx);
44          return now;
45      }, cancellationToken: cancellationToken);
46  }
```

5.5. Karty [cards]

5.5.1. Pobranie kart zadanej tablicy

```
1   public async Task<List<CardWithAssignedUserAndTags>> GetCardsByBoardId(ObjectId boardId,
    IClientSessionHandle? session = null,                                                    C#
2       CancellationToken cancellationToken = default)
3   {
4       List<CardWithAssignedUserAndTags> data = await (session == null
5           ? _cardsCollection.AsQueryable()
6           : _cardsCollection.AsQueryable(session))
7           .Where(c => c.BoardId == boardId)
8           .SelectMany(c => c.AssignedUsers.DefaultIfEmpty(), (card, assignedUserId) => new
            { card, assignedUserId })
```

```
9           .GroupJoin(_usersCollection,
10              c => c.assignedUserId,
11              u => u.Id,
12              (x, u) => new { x.card, x.assignedUserId, user = u.FirstOrDefault() })
13          .SelectMany(x => x.card.Tags.DefaultIfEmpty(), (x, tagId) => new { x.card,
            x.assignedUserId, x.user, tagId })
14          .Join(_boardsCollection,
15              x => x.card.BoardId,
16              board => board.Id,
17              (x, board) => new { x.card, x.assignedUserId, x.user, x.tagId, board })
18          .Select(x => new
19          {
20              x.card.Id,
21              x.card.BoardId,
22              x.card.SwimlaneId,
23              x.card.ListId,
24              x.card.Title,
25              x.card.Description,
26              x.card.DueDate,
27              x.card.CreatedAt,
28              x.card.UpdatedAt,
29              Tag = x.board.Swimlanes
30                  .Where(s => s.Id == x.card.SwimlaneId)
31                  .Select(s => s.Tags.FirstOrDefault(t => t.Id == x.tagId))
32                  .FirstOrDefault(),
33              AssignedUser = x.user == null ? null : new SimplifiedUser
34              {
35                  Id = x.user.Id,
36                  Nickname = x.user.Nickname,
37              },
38          })
39          .GroupBy(x => x.Id)
40          .Select(g => new CardWithAssignedUserAndTags
41          {
42              Id = g.Key,
43              BoardId = g.First().BoardId,
44              SwimlaneId = g.First().SwimlaneId,
45              ListId = g.First().ListId,
46              Title = g.First().Title,
47              Description = g.First().Description,
48              DueDate = g.First().DueDate,
49              CreatedAt = g.First().CreatedAt,
50              UpdatedAt = g.First().UpdatedAt,
51              Tags = g.Where(x => x.Tag != null).Select(x => x.Tag!),
52              AssignedUsers = g.Where(x => x.AssignedUser != null).Select(x => x.AssignedUser!),
53          }).ToListAsync(cancellationToken: cancellationToken);
```

```
    var lockedUserIds = LockedCards.Values.Distinct().ToList();
    var lockedUsers = await GetSimplifiedUser(session).Where(u =>
    lockedUserIds.Contains(u.Id)).ToListAsync(cancellationToken);
    var lockedUsersDict = lockedUsers.ToDictionary(u => u.Id, u => u);
    foreach (var card in data)
    {
        if (LockedCards.TryGetValue(card.Id, out var lockedUserId) &&
        lockedUsersDict.TryGetValue(lockedUserId, out var lockedUser))
        {
            card.LockedByUser = new SimplifiedUser
            {
                Id = lockedUser.Id,
                Nickname = lockedUser.Nickname
            };
        }
    }
    return data;
}
```

5.5.2. Pobranie wybranej karty

```C#
public async Task<CardWithAssignedUserAndTags?> GetCardById(ObjectId cardId, ObjectId
boardId,
    IClientSessionHandle? session = null, CancellationToken cancellationToken = default)
{
    CardWithAssignedUserAndTags? data = await (session == null
        ? _cardsCollection.AsQueryable()
        : _cardsCollection.AsQueryable(session))
        .Where(c => c.BoardId == boardId && c.Id == cardId)
        .SelectMany(c => c.AssignedUsers.DefaultIfEmpty(), (card, assignedUserId) => new
        { card, assignedUserId })
        .GroupJoin(_usersCollection,
            c => c.assignedUserId,
            u => u.Id,
            (x, u) => new { x.card, x.assignedUserId, user = u.FirstOrDefault() })
        .SelectMany(x => x.card.Tags.DefaultIfEmpty(), (x, tagId) => new { x.card,
        x.assignedUserId, x.user, tagId })
        .Join(_boardsCollection,
            x => x.card.BoardId,
            board => board.Id,
            (x, board) => new { x.card, x.assignedUserId, x.user, x.tagId, board })
        .Select(x => new
        {
            x.card.Id,
            x.card.BoardId,
            x.card.SwimlaneId,
            x.card.ListId,
            x.card.Title,
```

```
25              x.card.Description,
26              x.card.DueDate,
27              x.card.CreatedAt,
28              x.card.UpdatedAt,
29              Tag = x.board.Swimlanes
30                  .Where(s => s.Id == x.card.SwimlaneId)
31                  .Select(s => s.Tags.FirstOrDefault(t => t.Id == x.tagId))
32                  .FirstOrDefault(),
33              AssignedUser = x.user == null ? null : new SimplifiedUser
34              {
35                  Id = x.user.Id,
36                  Nickname = x.user.Nickname,
37              },
38          })
39          .GroupBy(x => x.Id)
40          .Select(g => new CardWithAssignedUserAndTags
41          {
42              Id = g.Key,
43              BoardId = g.First().BoardId,
44              SwimlaneId = g.First().SwimlaneId,
45              ListId = g.First().ListId,
46              Title = g.First().Title,
47              Description = g.First().Description,
48              DueDate = g.First().DueDate,
49              CreatedAt = g.First().CreatedAt,
50              UpdatedAt = g.First().UpdatedAt,
51              Tags = g.Where(x => x.Tag != null).Select(x => x.Tag!),
52              AssignedUsers = g.Where(x => x.AssignedUser != null).Select(x => x.AssignedUser!),
53          }).FirstOrDefaultAsync(cancellationToken: cancellationToken);
54      if (data == null)
55          return null;
56      if (LockedCards.TryGetValue(data.Id, out var lockedUserId))
57      {
58          var lockedUser = await GetLockingUserById(lockedUserId, session, cancellationToken);
59          if (lockedUser != null)
60          {
61              data.LockedByUser = new SimplifiedUser
62              {
63                  Id = lockedUser.Id,
64                  Nickname = lockedUser.Nickname
65              };
66          }
67      }
68      return data;
69  }
```

5.5.3. Pobranie uproszczonego użytkownika

```C#
private IQueryable<SimplifiedUser> GetSimplifiedUser(IClientSessionHandle? session = null)
{
    return (session == null
        ? _usersCollection.AsQueryable()
        : _usersCollection.AsQueryable(session))
        .Select(x => new SimplifiedUser
        {
            Id = x.Id,
            Nickname = x.Nickname,
        });
}
```

5.5.4. Pobranie użytkownika, który zablokował kartę

```C#
public async Task<SimplifiedUser?> GetLockingUserById(ObjectId id, IClientSessionHandle?
session = null, CancellationToken cancellationToken = default)
{
    return await GetSimplifiedUser(session).FirstOrDefaultAsync(u => u.Id == id,
    cancellationToken);
}
```

5.5.5. Utworzenie nowej karty

```C#
public async Task CreateCardAsync(Card card, ObjectId userId, IClientSessionHandle
session, CancellationToken cancellationToken = default)
{
    var result = await _boardsCollection
        .AsQueryable(session)
        .Where(b => b.Id == card.BoardId)
        .Select(b => new
        {
            b.Members,
            List = b.Swimlanes.Where(s => s.Id == card.SwimlaneId).Select(s =>
            s.Lists.FirstOrDefault(l => l.Id == card.ListId)).FirstOrDefault(),
        })
        .FirstOrDefaultAsync(cancellationToken)
        ?? throw new RecordDoesNotExist("Board has not been found.");
    Member? member = result.Members.FirstOrDefault(x => x.UserId == userId);
    if (member?.Role == BoardRole.Viewer)
        throw new ForbiddenException("User does not have permission to create card.");
    if (card.AssignedUsers.Any(x => !result.Members.Any(m => m.UserId == x)))
        throw new ForbiddenException("One or more assigned users are not members of this
        board.");
    if (result.List == null)
        throw new RecordDoesNotExist("List has not been found.");
    if (result.List.MaxWIP.HasValue && result.List.MaxWIP.Value > 0)
    {
        var filter = Builders<Card>.Filter.Eq(c => c.BoardId, card.BoardId)
```

```
23              & Builders<Card>.Filter.Eq(c => c.SwimlaneId, card.SwimlaneId)
24              & Builders<Card>.Filter.Eq(c => c.ListId, card.ListId);
25          long currentWIP = await _cardsCollection.CountDocumentsAsync(session, filter,
            cancellationToken: cancellationToken);
26          if (currentWIP >= result.List.MaxWIP.Value)
27              throw new ForbiddenException("Maximum cards WIP limit reached for this list.");
28      }
29      card.CreatedAt = DateTime.UtcNow;
30      card.UpdatedAt = DateTime.UtcNow;
31      await _cardsCollection.InsertOneAsync(session, card, cancellationToken:
        cancellationToken);
32  }
```

5.5.6. Utworzenie nowej karty wraz z jej zwrotem po utworzeniu

```C#
1  public async Task<CardWithAssignedUserAndTags> CreateAndFindCardAsync(Card card, ObjectId
   userId, CancellationToken cancellationToken = default)
2  {
3      using var session = await _mongoClient.StartSessionAsync(cancellationToken:
       cancellationToken);
4      return await session.WithTransactionAsync(async (s, ct) =>
5      {
6          await CreateCardAsync(card, userId, s, ct);
7          return await GetCardById(card.Id, card.BoardId, s, ct) ?? throw new
           InvalidOperationException("Card cannot just disappear!!!");
8      }, cancellationToken: cancellationToken);
9  }
```

5.5.7. Aktualizacja karty

```C#
1  public async Task UpdateCardAsync(Card card, ObjectId userId, IClientSessionHandle
   session , CancellationToken cancellationToken = default)
2  {
3      if (LockedCards.TryGetValue(card.Id, out var lockedUserId) && lockedUserId != userId)
4          throw new ForbiddenException("Card is locked by another user. Please try again
           later.");
5      var result = await _boardsCollection
6          .AsQueryable(session)
7          .Where(b => b.Id == card.BoardId)
8          .Select(b => new
9          {
10             b.Members,
11             MaxWIP = b.Swimlanes
12                 .Where(s => s.Id == card.SwimlaneId)
13                 .Select(s => s.Lists
14                     .Where(l => l.Id == card.ListId)
15                     .Select(l => l.MaxWIP)
16                     .FirstOrDefault())
17                 .FirstOrDefault()
```

```
18          })
19          .FirstOrDefaultAsync(cancellationToken)
20          ?? throw new RecordDoesNotExist("Board has not been found.");
21      BoardRole role = result.Members.FirstOrDefault(x => x.UserId == userId)?.Role
22          ?? throw new ForbiddenException("User is not a member of this board.");
23      if (role == BoardRole.Viewer)
24          throw new ForbiddenException("User does not have permission to create card.");
25      if (card.AssignedUsers.Any(x => !result.Members.Any(m => m.UserId == x)))
26          throw new ForbiddenException("One or more assigned users are not members of this
            board.");
27      var oldCard = await _cardsCollection
28          .AsQueryable(session)
29          .Where(x => x.Id == card.Id)
30          .Select(x => new { x.ListId, x.UpdatedAt })
31          .FirstOrDefaultAsync(cancellationToken);
32      if (result.MaxWIP.HasValue && result.MaxWIP.Value > 0)
33      {
34          var countFilter = Builders<Card>.Filter.Eq(c => c.BoardId, card.BoardId)
35              & Builders<Card>.Filter.Eq(c => c.SwimlaneId, card.SwimlaneId)
36              & Builders<Card>.Filter.Eq(c => c.ListId, card.ListId);
37          long currentWIP = await _cardsCollection.CountDocumentsAsync(session, countFilter,
            cancellationToken: cancellationToken);
38          if (oldCard.ListId != card.ListId && currentWIP >= result.MaxWIP.Value)
39              throw new ForbiddenException("Maximum WIP limit reached for this list.");
40      }
41
42      if (card.UpdatedAt != default && oldCard.UpdatedAt != card.UpdatedAt)
43          throw new PreconditionFailedException("Card has been modified or deleted by another
            user since it was last read.");
44      var now = DateTime.UtcNow;
45      var update = Builders<Card>.Update
46          .Set(c => c.Title, card.Title)
47          .Set(c => c.Description, card.Description)
48          .Set(c => c.DueDate, card.DueDate)
49          .Set(c => c.Tags, card.Tags)
50          .Set(c => c.AssignedUsers, card.AssignedUsers)
51          .Set(c => c.UpdatedAt, now);
52      var filter = Builders<Card>.Filter.Eq(c => c.Id, card.Id);
53      if (card.UpdatedAt != default)
54          filter &= Builders<Card>.Filter.Eq(c => c.UpdatedAt, card.UpdatedAt);
55      UpdateResult updateResult = await _cardsCollection.UpdateOneAsync(session, filter, update,
        cancellationToken: cancellationToken);
56      if (updateResult.MatchedCount == 0 && card.UpdatedAt != default)
57          throw new PreconditionFailedException("Card has been modified or deleted by another
            user by another user since it was last read.");
58      else if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
```

```
59              throw new InvalidOperationException("An unexpected error occurred while updating card.
                Card has not been modified.");
60          card.UpdatedAt = now;
61          UnlockCard(card.Id, userId);
62      }
```

5.5.8. Aktualizacja karty wraz z jej zwrotem po zaaktualizowaniu

```
1   public async Task<CardWithAssignedUserAndTags> UpdateAndFindCardAsync(Card card, ObjectId
    userId, CancellationToken cancellationToken = default)                                    C#
2   {
3       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
4       return await session.WithTransactionAsync(async (s, ct) =>
5       {
6           await UpdateCardAsync(card, userId, s, ct);
7           return await GetCardById(card.Id, card.BoardId, s, ct) ?? throw new
            InvalidOperationException("Card cannot just disappear!!!");
8       }, cancellationToken: cancellationToken);
9   }
```

5.5.9. Przeniesienie karty do innej listy

```
1   public async Task<DateTime> MoveCardAsync(ObjectId cardId, ObjectId userId, ObjectId
    listId,                                                                                   C#
2       DateTime updatedAt = default, CancellationToken cancellationToken = default)
3   {
4       using var session = await _mongoClient.StartSessionAsync(cancellationToken:
        cancellationToken);
5       return await session.WithTransactionAsync(async (s, ctx) =>
6       {
7           var result = await _cardsCollection
8               .AsQueryable()
9               .Where(card => card.Id == cardId)
10              .Join(
11                  _boardsCollection,
12                  c => c.BoardId,
13                  b => b.Id,
14                  (c, b) => new
15                  {
16                      c.UpdatedAt,
17                      CardAssignedUsers = c.AssignedUsers,
18                      BoardMembers = b.Members,
19                      BoardId = b.Id,
20                      MaxWIP = b.Swimlanes
21                          .Where(s => s.Id == c.SwimlaneId)
22                          .Select(s => s.Lists
23                              .Where(l => l.Id == listId)
24                              .Select(l => l.MaxWIP)
```

```
25                              .FirstOrDefault())
26                          .FirstOrDefault()
27                  }
28              )
29              .FirstOrDefaultAsync(cancellationToken)
30              ?? throw new RecordDoesNotExist("Card has not been found");
31          if (updatedAt != default && updatedAt != result.UpdatedAt)
32              throw new PreconditionFailedException("Board has been modified by another user
                since it was last read.");
33          var member = result.BoardMembers.FirstOrDefault(m => m.UserId == userId)
34              ?? throw new ForbiddenException("User is not a member of this board.");
35          if (member.Role == BoardRole.Viewer && !result.CardAssignedUsers.Contains(userId))
36              throw new ForbiddenException("User does not have permission to move this card");
37          if (result.MaxWIP.HasValue && result.MaxWIP.Value > 0)
38          {
39              long currentWIP = await _cardsCollection.CountDocumentsAsync(x => x.BoardId ==
                result.BoardId
40                  && x.ListId == listId, cancellationToken: cancellationToken);
41              var oldListId = await _cardsCollection
42                  .AsQueryable()
43                  .Where(x => x.Id == cardId)
44                  .Select(x => x.ListId)
45                  .FirstOrDefaultAsync(cancellationToken);
46              if (oldListId != listId && currentWIP >= result.MaxWIP.Value)
47                  throw new ForbiddenException("Maximum WIP limit reached for this list.");
48          }
49          var filter = Builders<Card>.Filter.Eq(c => c.Id, cardId);
50          if (updatedAt != default)
51              filter &= Builders<Card>.Filter.Eq(c => c.UpdatedAt, updatedAt);
52          var now = DateTime.UtcNow;
53          var update = Builders<Card>.Update
54              .Set(c => c.ListId, listId)
55              .Set(c => c.UpdatedAt, now);
56          UpdateResult updateResult = await _cardsCollection.UpdateOneAsync(filter, update,
            cancellationToken: cancellationToken);
57          if (updateResult.MatchedCount == 0 && updatedAt != default)
58              throw new PreconditionFailedException("Card has been modified by another user
                since it was last read.");
59          else if (updateResult.MatchedCount != 0 && updateResult.ModifiedCount == 0)
60              throw new InvalidOperationException("An unexpected error occurred while moving
                card. Card has not been modified.");
61          return now;
62      }, cancellationToken: cancellationToken);
63  }
```

5.5.10. Usunięcie karty

```C#
public async Task DeleteCardAsync(ObjectId cardId, ObjectId userId, DateTime updatedAt = default, CancellationToken cancellationToken = default)
{
    if (LockedCards.TryGetValue(cardId, out var lockedUserId) && lockedUserId != userId)
        throw new ForbiddenException("Card is locked by another user. Please try again later.");
    using var session = await _mongoClient.StartSessionAsync(cancellationToken: cancellationToken);
    await session.WithTransactionAsync(async (s, ctx) =>
    {
        var result = await _cardsCollection
        .AsQueryable(s)
        .Where(card => card.Id == cardId)
        .Join(
            _boardsCollection,
            c => c.BoardId,
            b => b.Id,
            (c, b) => new
            {
                c.UpdatedAt,
                CardId = c.Id,
                BoardRole = b.Members
                    .Where(x => x.UserId == userId)
                    .Select(x => x.Role)
                    .FirstOrDefault(),
            }
        )
        .FirstOrDefaultAsync(ctx);
        if (result == null || result.CardId == default)
            throw new RecordDoesNotExist("Card has not been found.");
        if (updatedAt != default && updatedAt != result.UpdatedAt)
            throw new PreconditionFailedException("Card has been modified or deleted by another user since it was last read.");
        if (result.BoardRole == BoardRole.Viewer)
            throw new ForbiddenException("User does not have permission to delete card.");
        var filter = Builders<Card>.Filter.Eq(c => c.Id, cardId);
        if (updatedAt != default)
            filter &= Builders<Card>.Filter.Eq(c => c.UpdatedAt, updatedAt);
        DeleteResult deleteResult = await _cardsCollection.DeleteOneAsync(s, filter, null, cancellationToken: ctx);
        if (deleteResult.DeletedCount == 0 && updatedAt != default)
            throw new PreconditionFailedException("Card has been modified or deleted by another user since it was last read.");
        return Task.CompletedTask;
    }, cancellationToken: cancellationToken);
    UnlockCard(cardId, userId);
```

```
41 }
```

5.6. Tokeny odświeżenia [refreshTokens]

5.6.1. Dodanie tokena odświeżenia

```C#
public async Task AddRefreshToken(RefreshToken token, CancellationToken cancellationToken = default)
{
    try
    {
        await _refreshTokensCollection.InsertOneAsync(token, null, cancellationToken);
    }
    catch (MongoWriteException ex) when (ex.WriteError.Category == ServerErrorCategory.DuplicateKey)
    {
        throw new RecordAlreadyExists("Duplicate of refresh tokens occured!");
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An unexpected error occurred while adding refresh token.");
        throw new InvalidOperationException("An unexpected error occurred while adding refresh token.", ex);
    }
}
```

5.6.2. Usunięcie wszystkich tokenów odświeżenia

```C#
public async Task DeleteAllRefreshTokens(ObjectId userId, CancellationToken cancellationToken = default)
{
    try
    {
        var deleteFilter = Builders<RefreshToken>.Filter.And(
            Builders<RefreshToken>.Filter.Eq(rt => rt.UserId, userId)
        );

        await _refreshTokensCollection.DeleteManyAsync(deleteFilter, cancellationToken);
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An unexpected error occurred while deleting refresh tokens.");
        throw new InvalidOperationException("An unexpected error occurred while deleting refresh tokens.", ex);
    }
}
```

5.6.3. Pobranie użytkownika na podstawie tokena odświeżenia

```C#
public async Task<User?> GetUserByRefreshTokenAsync(string refreshToken, CancellationToken cancellationToken = default)
```

```
2   {
3       try
4       {
5           var tokenFilter = Builders<RefreshToken>.Filter.And(
6               Builders<RefreshToken>.Filter.Eq(rt => rt.Token, refreshToken),
7               Builders<RefreshToken>.Filter.Gt(rt => rt.ExpiresAt, DateTime.UtcNow)
8           );
9
10          RefreshToken? token = await
            _refreshTokensCollection.FindOneAndDeleteAsync(tokenFilter, cancellationToken:
            cancellationToken);
11
12          if (token == null)
13              return null;
14
15          return await _userService.GetUserByIdAsync(token.UserId, cancellationToken);
16      }
17      catch (Exception ex)
18      {
19          _logger.LogError(ex, "An unexpected error occurred while getting user by refresh
            token.");
20          throw new InvalidOperationException("An unexpected error occurred while getting user
            by refresh tokenn.", ex);
21      }
22  }
```

5.7. Użytkownicy [users]

5.7.1. Dodanie nowego użytkownika

```C#
1   public async Task AddUserAsync(User user, CancellationToken cancellationToken = default)
2   {
3       try
4       {
5           user.Password = _passwordHasher.HashPassword(user, user.Password);
6           await _usersCollection.InsertOneAsync(user, cancellationToken: cancellationToken);
7       }
8       catch (MongoWriteException ex) when (ex.WriteError.Category ==
        ServerErrorCategory.DuplicateKey)
9       {
10          throw new RecordAlreadyExists("User with such nickname exists!");
11      }
12  }
```

5.7.2. Aktualizacja hasła użytkownika

```C#
1   public async Task UpdatePasswordAsync(User user, CancellationToken cancellationToken =
    default)
2   {
3       user.Password = _passwordHasher.HashPassword(user, user.Password);
```

```
4       user.UpdatedAt = DateTime.UtcNow;
5       var filter = Builders<User>.Filter.Eq(u => u.Id, user.Id);
6       var update = Builders<User>.Update
7           .Set(u => u.Password, user.Password)
8           .Set(u => u.UpdatedAt, user.UpdatedAt);
9       UpdateResult result = await _usersCollection.UpdateOneAsync(filter, update,
        cancellationToken: cancellationToken);
10      if (result.ModifiedCount == 0)
11          throw new RecordDoesNotExist("User has not been found.");
12  }
```

5.7.3. Znalezienie użytkownika po dokładnej nazwie

```
1   public async Task<User?> GetUserByNicknameAsync(string nickname, CancellationToken
    cancellationToken = default)                                                    C#
2   {
3       return await _usersCollection.Find(u => u.Nickname == nickname,
        null).FirstOrDefaultAsync(cancellationToken);
4   }
```

5.7.4. Znalezienie wszystkich użytkowników

```
1   public async Task<List<User>> FindUsersAsync(string? nickname, List<ObjectId>
    blacklistIds, CancellationToken cancellationToken = default)                    C#
2   {
3       var nicknameFilter = string.IsNullOrWhiteSpace(nickname)
4           ? Builders<User>.Filter.Empty
5           : Builders<User>.Filter.Regex(u => u.Nickname, new
            BsonRegularExpression($".*{Regex.Escape(nickname)}.*", "i"));
6
7       var blacklistFilter = (blacklistIds != null && blacklistIds.Count > 0)
8           ? Builders<User>.Filter.Nin(u => u.Id, blacklistIds)
9           : Builders<User>.Filter.Empty;
10
11      var combinedFilter = Builders<User>.Filter.And(nicknameFilter, blacklistFilter);
12
13      return await
        _usersCollection.Find(combinedFilter).Limit(10).ToListAsync(cancellationToken);
14  }
```

5.7.5. Porównanie hashy haseł

```
1   public async Task<bool> VerifyHashedPassword(User user, string providedPassword,
    CancellationToken cancellationToken = default)                                  C#
2   {
3       var result = _passwordHasher.VerifyHashedPassword(user, user.Password, providedPassword);
4       if (result == PasswordVerificationResult.SuccessRehashNeeded)
5       {
6           await _usersCollection.UpdateOneAsync(
7               u => u.Id == user.Id,
```

```
8               Builders<User>.Update
9                   .Set(u => u.Password, _passwordHasher.HashPassword(user, providedPassword))
10                  .Set(u => u.UpdatedAt, DateTime.UtcNow),
11              null,
12              cancellationToken
13          );
14      }
15      return result != PasswordVerificationResult.Failed;
16  }
```

6. Schemat API REST

6.1. POST `/auth/signin`

Treść zapytania:

- Content-Type: `application/json`

```
{
  nickname: string
  password: string
}
```

Odpowiedzi:

- 200 OK

```
{
  accessToken: string
  accessTokenExpiresIn: integer (uint32)
  refreshToken: string
  refreshTokenExpiresIn: integer (uint32)
}
```

- 400 Bad Request
- 401 Unauthorized
- 500 Internal Server Error

6.2. POST `/auth/signup`

Treść zapytania:

- Content-Type: `application/json`

```
{
  nickname: string
  password: string
}
```

Odpowiedzi:

- 200 OK

```
{
  accessToken: string
  accessTokenExpiresIn: integer (uint32)
  refreshToken: string
  refreshTokenExpiresIn: integer (uint32)
}
```

- 400 Bad Request
- 401 Unauthorized
- 500 Internal Server Error

6.3. POST `/auth/refresh`

Treść zapytania:

- Content-Type: `application/json`

```
{
  refreshToken: string
}
```

Odpowiedzi:
- 200 OK

```
{
  accessToken: string
  accessTokenExpiresIn: integer (uint32)
  refreshToken: string
  refreshTokenExpiresIn: integer (uint32)
}
```

- 400 Bad Request
- 401 Unauthorized
- 500 Internal Server Error

6.4. POST `/auth/revoke`

Odpowiedzi:
- 200 OK

```
{
  message: string
}
```

- 401 Unauthorized
- 500 Internal Server Error

6.5. GET `/boards`

Odpowiedzi:
- 200 OK

```
[
  {
    id: string (objectid)
    title: string
    members: [
      {
        userId: string (objectid)
        nickname: string
        role: string
        isActive: boolean
      }
      ...
    ]
    createdAt: string (date-time)
    updatedAt: string (date-time)
  }
  ...
```

```
18 ]
```

- 401 Unauthorized
- 500 Internal Server Error

6.6. POST `/boards`

Treść zapytania:

- Content-Type: `application/json`

```
1  {
2    title: string
3    members: [
4      {
5        userId: string (objectid)
6        role: string
7      }
8      ...
9    ]
10   swimlanes: [
11     {
12       title: string
13       tags: [
14         {
15           title: string
16           color: string
17         }
18         ...
19       ]
20       lists: [
21         {
22           title: string
23           color: string
24           maxWIP: integer (int32)
25         }
26         ...
27       ]
28       color: string
29     }
30     ...
31   ]
32 }
```

Odpowiedzi:

- 200 OK

```
1 {
2   id: string (objectid)
3 }
```

- 400 Bad Request
- 401 Unauthorized
- 500 Internal Server Error

6.7. GET `/boards/{boardId}`

Odpowiedzi:

- 200 OK

```
{
  id: string (objectid)
  title: string
  swimlanes: [
    {
      id: string (objectid)
      title: string
      lists: [
        {
          id: string (objectid)
          title: string
          color: string
          maxWIP: integer (int32)
        }
        ...
      ]
      tags: [
        {
          id: string
          title: string
          color: integer
        }
        ...
      ]
      color: string
    }
    ...
  ]
  members: [
    {
      userId: string (objectid)
      nickname: string
      role: string
      isActive: boolean
    }
    ...
  ]
  createdAt: string (date-time)
  updatedAt: string (date-time)
```

```
40 }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.8. PATCH `/boards/{boardId}`

Treść zapytania:

- Content-Type: `application/json`

```
1  {
2    title: string
3    members: [
4      {
5          userId: string (objectid)
6          role: string
7      }
8      ...
9    ]
10 }
```

Odpowiedzi:

- 200 OK

```
1 {
2   message: string
3 }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 409 Conflict
- 500 Internal Server Error

6.9. DELETE `/boards/{boardId}`

Odpowiedzi:

- 200 OK

```
1 {
2   message: string
3 }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 409 Conflict
- 500 Internal Server Error

6.10. GET `/boards/{boardId}/cards`

Odpowiedzi:

- 200 OK

```
[
  {
    id: string (objectid)
    boardId: string (objectid)
    swimlaneId: string (objectid)
    listId: string (objectid)
    title: string
    description: string
    dueDate: string (date-time)
    assignedUsers: [
      {
        id: string (objectid)
        nickname: string
      }
      ...
    ]
    lockedByUser: {
        id: string (objectid)
        nickname: string
    }
    tags: [
      {
        id: string
        title: string
        color: integer
      }
      ...
    ]
    createdAt: string (date-time)
    updatedAt: string (date-time)
  }
  ...
]
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.11. POST `/boards/{boardId}/cards`

Treść zapytania:

- Content-Type: `application/json`

```
{
  title: string
  swimlaneId: string (objectid)
  listId: string (objectid)
  tags: [
    string (objectid)
    ...
  ]
  dueDate: string (date-time)
  assignedUsers: [
    string (objectid)
    ...
  ]
  description: string
}
```

Odpowiedzi:
- 200 OK

```
{
  id: string (objectid)
}
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.12. GET `/boards/{boardId}/cards/{cardId}`

Odpowiedzi:
- 200 OK

```
{
  id: string (objectid)
  boardId: string (objectid)
  swimlaneId: string (objectid)
  listId: string (objectid)
  title: string
  description: string
  dueDate: string (date-time)
  assignedUsers: [
    {
      id: string (objectid)
      nickname: string
    }
    ...
  ]
```

```
  lockedByUser: {
    id: string (objectid)
    nickname: string
  }
  tags: [
    {
      id: string
      title: string
      color: integer
    }
    ...
  ]
  createdAt: string (date-time)
  updatedAt: string (date-time)
}
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.13. PATCH `/boards/{boardId}/cards/{cardId}`

Treść zapytania:

- Content-Type: `application/json`

```
{
  title: string
  tags: [
    string (objectid)
    ...
  ]
  dueDate: string (date-time)
  assignedUsers: [
    string (objectid)
    ...
  ]
  description: string
}
```

Odpowiedzi:

- 200 OK

```
{
  message: string
}
```

- 400 Bad Request
- 401 Unauthorized

- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.14. DELETE `/boards/{boardId}/cards/{cardId}`

Odpowiedzi:

- 200 OK

```
1 {
2   message: string
3 }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.15. PATCH `/boards/{boardId}/cards/{cardId}/move`

Treść zapytania:

- Content-Type: `application/json`

```
1 {
2   listId: string (objectid)
3 }
```

Odpowiedzi:

- 200 OK

```
1 {
2   message: string
3 }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.16. GET `/boards/{boardId}/swimlanes/{swimlaneId}/lists`

Odpowiedzi:

- 200 OK

```
1 [
2   {
3     id: string (objectid)
4     title: string
5     color: string
6     maxWIP: integer (int32)
7   }
8   ...
```

```
9 ]
```

- 400 Bad Request
- 401 Unauthorized
- 404 Not Found
- 500 Internal Server Error

6.17. POST `/boards/{boardId}/swimlanes/{swimlaneId}/lists`

Treść zapytania:

- Content-Type: `application/json`

```
1 {
2   title: string
3   color: string
4   maxWIP: integer (int32)
5 }
```

Odpowiedzi:

- 200 OK

```
1 {
2   id: string (objectid)
3 }
```

- 400 Bad Request
- 401 Unauthorized
- 404 Not Found
- 500 Internal Server Error

6.18. GET `/boards/{boardId}/swimlanes/{swimlaneId}/lists/{listId}`

Odpowiedzi:

- 200 OK

```
1 {
2   id: string (objectid)
3   title: string
4   color: string
5   maxWIP: integer (int32)
6 }
```

- 400 Bad Request
- 401 Unauthorized
- 404 Not Found
- 500 Internal Server Error

6.19. PATCH `/boards/{boardId}/swimlanes/{swimlaneId}/lists/{listId}`

Treść zapytania:

- Content-Type: `application/json`

```
1 {
2   title: string
```

```
3    color: string
4    maxWIP: integer (int32)
5  }
```

Odpowiedzi:
- 200 OK

```
1  {
2    message: string
3  }
```

- 400 Bad Request
- 401 Unauthorized
- 404 Not Found
- 500 Internal Server Error

6.20. DELETE `/boards/{boardId}/swimlanes/{swimlaneId}/lists/{listId}`

Odpowiedzi:
- 200 OK

```
1  {
2    message: string
3  }
```

- 400 Bad Request
- 401 Unauthorized
- 404 Not Found
- 500 Internal Server Error

6.21. GET `/boards/{boardId}/swimlanes`

Odpowiedzi:
- 200 OK

```
1  [
2    {
3      id: string (objectid)
4      title: string
5      color: string
6    }
7    ...
8  ]
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.22. POST `/boards/{boardId}/swimlanes`

Treść zapytania:
- Content-Type: `application/json`

```
{
  title: string
  tags: [
    {
      title: string
      color: string
    }
    ...
  ]
  lists: [
    {
      title: string
      color: string
      maxWIP: integer (int32)
    }
    ...
  ]
  color: string
}
```

Odpowiedzi:
- 200 OK

```
{
  id: string (objectid)
}
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.23. GET `/boards/{boardId}/swimlanes/{swimlaneId}`

Odpowiedzi:
- 200 OK

```
{
  id: string (objectid)
  title: string
  lists: [
    {
      id: string (objectid)
      title: string
      color: string
      maxWIP: integer (int32)
    }
    ...
```

```
12    ]
13    tags: [
14      {
15        id: string
16        title: string
17        color: integer
18      }
19      ...
20    ]
21    color: string
22  }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.24. PATCH `/boards/{boardId}/swimlanes/{swimlaneId}`

Treść zapytania:

- Content-Type: `application/json`

```
1  {
2    title: string
3    color: string
4  }
```

Odpowiedzi:

- 200 OK

```
1  {
2    message: string
3  }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.25. DELETE `/boards/{boardId}/swimlanes/{swimlaneId}`

Odpowiedzi:

- 200 OK

```
1  {
2    message: string
3  }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden

- 404 Not Found
- 500 Internal Server Error

6.26. GET `/boards/{boardId}/swimlanes/{swimlaneId}/tags`

Odpowiedzi:

- 200 OK

```
[
  {
    id: string
    title: string
    color: integer
  }
  ...
]
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.27. POST `/boards/{boardId}/swimlanes/{swimlaneId}/tags`

Treść zapytania:

- Content-Type: `application/json`

```
{
  title: string
  color: string
}
```

Odpowiedzi:

- 200 OK

```
{
  id: string (objectid)
}
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.28. GET `/boards/{boardId}/swimlanes/{swimlaneId}/tags/{tagId}`

Odpowiedzi:

- 200 OK

```
{
  id: string
  title: string
```

```
4    color: integer
5  }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.29. PATCH `/boards/{boardId}/swimlanes/{swimlaneId}/tags/{tagId}`

Treść zapytania:

- Content-Type: `application/json`

```
1  {
2    title: string
3    color: string
4  }
```

Odpowiedzi:

- 200 OK

```
1  {
2    message: string
3  }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.30. DELETE `/boards/{boardId}/swimlanes/{swimlaneId}/tags/{tagId}`

Odpowiedzi:

- 200 OK

```
1  {
2    message: string
3  }
```

- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

6.31. GET `/users/me`

Odpowiedzi:

- 200 OK

```
1  {
2    id: string (objectid)
```

```
3    nickname: string
4    createdAt: string (date-time)
5    updatedAt: string (date-time)
6  }
```

- 401 Unauthorized
- 500 Internal Server Error

6.32. PATCH `/users/me`

Treść zapytania:

- Content-Type: `application/json`

```
1  {
2    password: string
3  }
```

Odpowiedzi:

- 200 OK

```
1  {
2    message: string
3  }
```

- 400 Bad Request
- 401 Unauthorized
- 404 Not Found
- 500 Internal Server Error

6.33. GET `/users`

Odpowiedzi:

- 200 OK

```
1  [
2    {
3      id: string (objectid)
4      nickname: string
5      createdAt: string (date-time)
6      updatedAt: string (date-time)
7    }
8    ...
9  ]
```

- 401 Unauthorized
- 500 Internal Server Error

6.34. GET `/users/{userId}`

Odpowiedzi:

- 200 OK

```
1  {
2    id: string (objectid)
```

```
3    nickname: string
4    createdAt: string (date-time)
5    updatedAt: string (date-time)
6  }
```

- 400 Bad Request
- 401 Unauthorized
- 404 Not Found
- 500 Internal Server Error