# ЧАСТНА ПРОФЕСИОНАЛНА ГИМНАЗИЯ ЗА ДИГИТАЛНИ НАУКИ

# „СОФТУНИ БУДИТЕЛ“, гр. София

# 

# ДИПЛОМЕН ПРОЕКТ

# На Алеко Ивайлов Нешев

# Ученик от XII А клас

# професия- код: 481030, “Приложен програмист”

# специалност- код: 4810301, “Приложно програмиране”

# 

# Тема: Изработка на крос-платформа - приложение тип социална мрежа, насочена към различни научни общности, която да съдържа място за споделяне на научни разработки, групи, диференцирани по категории и чат за дискусии по научни проблеми

# 

# Ръководител-Консултант: Светлана Виктомирова

# Сесия: май-юни 2024г.

# Дата:.........................

# Съдържание:

1. Въведение
2. Архитектура
   1. База
      1. Таблици
3. API
   1. Дизайн
   2. Services
   3. Контролери
   4. SignalR Hubs
   5. Мапиране
4. Zest Frontend
   1. Страници
   2. Навигация
   3. SignalR Hubs
5. Authentication and Authorization
6. Заключение
   1. Бъдещо развитие

# 

# 

# Въведение

“Zest” е социална мрежа, предназначена за научната общност. Целта на мрежата е да популяризира и събере на едно място дискусии от всички сфери на науката и да помага на начинаещите по техния път на опознаване и разбиране на нашия свят. Социалните мрежи са неизменна част от нашия живот. Голям процент от населението прекарва значително време в тях. Голям проблем при социалните мрежи е разпространяването на дезинформация и език на омразата. “Zest” спира това чрез ясни правила и строга политика за елиминиране на тези проблеми.

Някои социалните мрежи се използват не по предназначение. Част от тях се използват за разпространяване на опасни предизвикателства и пропаганда с политически цели. Изследвания показват, че голяма част от младото поколение има значително влошаване на поведението, липса на емоционална интелигентност и проблем със задържането на внимание, в резултат на дълго и продължително използване на дадените приложения. “Zest” не е сред тях. Социалната мрежа е пълна с образователен материал и общности за провеждане на дискусии по дадена научна тематика. Приложението има потенциал да привлече голяма част от учениците и да ги откъсне от платформите със нездравословно съдържание. За учените това ще е място, в което те свободно могат да дискутират последните теории и открития, за останалите това ще е място, в което ще могат успешно да вникнат в сложния, но невероятен свят на науката. За учениците и студентите са предназначени общости, в които да могат да задават своите въпроси и да получат отговор от разбиращите дадената материя.

Проблем на социалните мрежи е тяхното таргетиране на своите потребители, чрез алгоритми, с цел задържане на внимание в приложението. Алгоритмите, които API на Zest предоставя, не позволява това да се случи. Отвореният код на системата е гарант за потребителите.

# 

# Архитектура

Системата е съставена от три части:

1. База данни
2. API
3. Cross-platform client application

Целта на разделянето на сървърната и предната част на системата, е по-лесна поддръжка и гъвкавост. По този начин, се дава възможност за разработване на много клиентски приложения за различни платформи и съответно по-голям избор за крайните потребители.

## База

Базата данни включва няколко свързани таблици, които представляват структурата на данните за социалната мрежа.

### Таблици

#### Accounts

* + Съхранява информация за потребителските акаунти.
  + Съдържа колони
    - Id
    - Username
    - Email
    - IsAdmin
    - CreatedOn.

#### Communities

* + Представлява общностите в приложението.
  + Съдържа колони
    - Id
    - Name
    - Information
    - CreatedOn
  + Релации:
    - FK\_Communities\_Accounts - сочи към създателя на публикацията

#### Posts

* + Съдържа публикациите, които потребителите на мрежата са създали.
  + Съдържа колони
    - Id
    - Title
    - Text
    - AccountId
    - CommunityId
    - CreatedOn.
  + Релации:
    - FK\_Posts\_Accounts - сочи към създателят на публикацията.
    - FK\_Posts\_Communities - сочи към общостта, към която публикацията принадлежи

#### Comments

* + Таблицата за коментарите, свързани с дадена публикация..
  + Съдържа колони
    - Id
    - Text
    - AccountId
    - PostId
    - CommentId
    - CreatedOn.
  + Релации
    - FK\_Comments\_Accounts - сочи към създателя на коментара.
    - FK\_Comments\_Posts - сочи към публикацията, към която е създаден коментарът.
    - FK\_Comments\_Comments - сочи към коментар, ако сегашния обект е коментар към него (Reply).

#### Likes

* + Съхранява реакциите на потребителите към дадена публикация или коментар.
  + Съдържа колони
    - Id
    - Value
    - AccountId
    - PostId
    - CommentId
    - CreatedOn
  + Релации
    - FK\_Likes\_Accounts - сочи към създателят на реакцията
    - FK\_Likes\_Posts - сочи към публикацията, към която е създадена реакцията (ако принадлежи към публикация)
    - FK\_Likes\_Comments - сочи към коментара, към която е създадена реакцията (ако принадлежи към коментар)

#### Followers

* + Съдържа следването между потребителите.
  + Съдържа колони
    - FollowerId
    - FollowedId
  + Релации:
    - FK\_Followers\_Accounts - сочи към последователя
    - FK\_Followers\_Accounts1 - сочи към следвания

#### Messages

* + Отговаря за съхранението на лични съобщения между потребителите.
  + Съдържа колони
    - Id
    - Text
    - SenderId
    - ReceiverId
    - CreatedOn.
  + Релации:
    - FK\_Followers\_Accounts - сочи към получателя на съобщението
    - FK\_Followers\_Accounts1 - сочи към пратилия съобщението

#### CommunityFollowers

* + Съхранява следването на общности от потребителите.
  + Съдържа колони:
    - CommunityId
    - AccountId.
  + Релации:
    - FK\_CommunityFollowers\_Accounts - сочи към следващия общността потребител
    - FK\_CommunityFollower\_Communities - сочи към следваната общност

#### CommunityModerators

* + Съхранява модераторите на общности.
  + Съдържа колони
    - CommunityId
    - AccountId
    - IsApproved.
  + Релации
    - FK\_CommunityFollowers\_Accounts - сочи към модериращия на общността потребител
    - FK\_CommunityFollower\_Communities - сочи към модерираната общност

#### PostResources

* + Съхранява локацията на ресурсите (снимки или видеа) на публикациите (ако има такава).
  + Съдържа колони:
    - Id
    - PostId
    - Type
    - Name
    - Path
    - CreatedOn.
  + Релации:
    - FK\_PostResources\_Posts - сочи към публикацията, към която ресурсите принадлежат

Ресурсите към публикациите се пазят във файловата система на проекта.

## API

API е разработен на .NET Web API, написан на езика за програмиране C#. Дадената технология е избрана, зареди нейната надеждност, добра и подробна документация и голям избор от публични ресурси.

### Дизайн

API е състван от 5 проекта:

1. Zest - Web Application
2. Zest\_DBModels - Библиотека, в която стои DBContext и DB моделите
3. Zest\_Services - Библиотека, в което стоят услугите на API.
4. Zest\_ViewModels - Библиотека, в която стоят DTO на DB моделите
5. Zest\_UnitTests - Тестове за API

### Services

Услугите в Zest са разделени по предназначение. Всички услуги са ограничени от съответния интерфейс.

**AccountService**

1. **Функционалност**
   1. Предоставя всички нужни услуги за управление на потребителите в социалната мрежа.
2. **Методи**
   1. '**DoesExistAsync(string id)**' - Проверява дали съществува акаунт с даден идентификатор.
   2. '**FindByIdAsync(string id)**' - Връща акаунта с даден идентификатор като обект 'AccountViewModel'.
   3. '**FindByUsernameAsync(string username)**' - Проверява дали съществува акаунт със даденото потребителско име.
   4. '**AddAsync(string accountId, string username, string email**)' - Добавя нов акаунт към базата данни с посочения идентификатор, потребителско име и имейл адрес.
   5. **'GetAllAsync(string accountId, int takeCount, int skipCount)**' - Връща списък с всички акаунти, освен този с дадения идентификатор, сортиран по дата на създаване и ограничен до определен брой елементи. Методът пропуска определен брой потребители, с цел предоставяне на възможността за зареждане на допълнителен брой впоследствие. Всеки акаунт включва булева стойност, показваща дали текущият потребител следва дадения акаунт.
   6. '**FindFollowerAsync(string followerId, string followedId)**' - Връща първия последовател, където идентификаторът на текущия потребител съвпада със зададената стойност, и идентификаторът на следвания акаунт съвпада със зададената стойност.
   7. '**GetBySearchAsync(string search, string accountId, int takeCount, string[]? skipIds)'** - Връща списък с акаунти, сортирани по дата на създаване и ограничен до определен брой елементи. Списъкът изключва акаунти с идентификатори в масива 'skipIds', с цел предоставяне на възможност за зареждане на допълнителен брой акаунти впоследствие, и текущия потребител.

**CommentService**

1. **Функционалност** 
   1. Предоставя всички нужни услуги за управление на коментарите в социалната мрежа.
2. **Методи**:
   1. **'FindLike(IEnumerable<CommentViewModel> commentViewModels, string accountId)'** - рекурсивно проверява за наличие на реакции към даден коментар. Предназначен е за отговори на коментарите.
   2. **'DoesExist(int id)'** - Проверява дали коментар със дадения идентификатор съществува в базата данни.
   3. **'FindAsync(int id, string accountId)'** - Връща коментар с посочения идентификатор заедно с отговори и харесвания, ако има такива. Зареждането на Account, Likes и Replies става чрез Eager Loading. Зарежда до 4 отговора надълбоко, с цел избягване на зареждането на твърде много такива. Методът проверява дали коментарът има реакция от текущия потребител. Прави същото и за неговите отговори, като извиква FindLike().
   4. **'AddAsync(string accountId, int postId, string text, int commentId = 0)'** - Добавя нов коментар към базата данни с предоставените идентификатори на текущия потребител и публикацията, към която той бива създаден, и текст. Ако 'commentId' е нула (не е отговор на друг коментар), методът създава нов коментар; в противен случай той създава отговор на съществуващ коментар с индентификатор 'commentId'.
   5. **'RemoveAsync(int id)'** - Маркира коментар със зададения идентификатор като изтрит в базата данни. Маркираните коментари връщат “Deleted” текст и “Unkown” създател. Прави се с цел избягване на загубата на дискусията под коментара. Оставя се на клиентските приложения да решат дали да ги показват или не.
   6. **'GetCommentsByPostIdAsync(int postId, DateTime lastDate, int takeCount, string accountId)'** - Връща коментари по публикация, създадени след определена дата, сортирани по време на създаване и ограничени до определен брой елементи. Връща коментари създадени след дадена дата, с цел предоставяне на възможност за зареждане на допълнителен брой коментари в последствие. Методът рекурсивно извлича харесванията за коментари, техните отговори и харесвания.
   7. **'Calculate score(Comment comment, double likeWeight, double commentWeight, double decayFactor)'** - Методът прави калкулация на базата на брой реакции на коментара и брой на отговори. Резултатът зависи от това, кога са направени дадените реакции и отговори.
   8. **'GetTrendingCommentsAsync(int[] skipIds, int takeCount, string accountId, int postId)'** - Методът връща популярни коментари, като консумира методът “Calculate Score”. Списъкът изключва коментари с идентификатори в масива 'skipIds', с цел предоставяне на възможност за зареждане на допълнителен брой коментари в последствие.

**CommunityFollowerService**

1. **Функционалност** 
   1. Предоставя всички нужни услуги за управление на последователите на общности в социалната мрежа.
2. **Методи:**
   1. **'DoesExistAsync(string accountId, int communityId)'** - Проверява дали последовател на общност със зададените идентификатори съществува в базата данни.
   2. **'AddAsync(string accountId, int communityId)'** - Добавя нов последовател на общност към базата данни с предоставените идентификатори.
   3. **'DeleteAsync(string accountId, int communityId)'** - Премахва последовател на общност със зададените идентификатори от базата данни.

**CommunityModeratorService**

1. **Функционалност** 
   1. Предоставя всички нужни услуги за управление на модераторите на общности в социалната мрежа.
2. **Методи**:
   1. **'IsModeratorAsync(string accountId, int communityId)'** - Проверява дали акаунтът със зададения идентификатор е модератор на общността.
   2. **'IsModeratorCandidateAsync(string accountId, int communityId)'** - Проверява дали акаунтът със зададения идентификатор е кандидат-модератор на общността.
   3. **'AddModeratorAsync(string accountId, int communityId)'** - Добавя нов кандидат-модератор към базата данни с предоставените идентификатори.
   4. **'GetModeratorsByCommunityAsync(int communityId)'** - Връща модераторите на общността със дадения идентификатор като масив от обекти 'UserViewModel'.
   5. **'GetModeratorCandidatesByCommunityAsync(int communityId)'** - Връща кандидат-модераторите на общността със зададения идентификатор като масив от обекти 'UserViewModel'.
   6. **'ApproveCandidateAsync(string accountId, int communityId)'** - Одобрява кандидат-модератор със зададения идентификатор за дадена общност.
   7. **'RemoveModeratorAsync(string accountId, int communityId)'** - Премахва модератора със зададения идентификатор от общността.

**CommunityService**

1. **Функционалност** 
   1. Предоставя всички нужни услуги за управление на общностите в социалната мрежа.
2. **Методи**:
   1. **'DoesExistAsync(int id)'** - Проверява дали общност със дадения идентификатор съществува в базата данни.
   2. **'GetCommunityByIdAsync(int id, string accountId)'** - Връща общност с дадения идентификатор като обект 'CommunityViewModel' заедно с информация дали текущият потребител следва тази общност.
   3. **'GetAllCommunitiesAsync(string accountId, int skipCount, int takeCount)'** - Връща всички общности от базата данни като масив от обекти ‘CommunityViewModel’ заедно с информация дали текущият потребител следва тази общност.
   4. **'AddCommunityAsync(string creatorId, string name, string description)'** - Добавя нова общност към базата данни с предоставените идентификатори, име и описание. Създателят автоматично се прави модератор на новосъздадената общност.
   5. **'GetCommunitiesByAccount(string accountId, int takeCount, int skipCount)'** - Връща общности, които текущият потребител следва, като масив от обекти 'CommunityViewModel'.
   6. **'GetTrendingCommunitiesAsync(int[] skipIds, int takeCount, string accountId)'** - Връща популярни общности въз основа на определени критерии - брой потребители, публикации, харесвания и коментари, като извикава методите ‘CalculateCommunityStats’ и ‘CalculateComunityScore’
   7. **'CalculateCommunityStats(Community community)'** - Изчислява статистиката на общността въз основа на бров на нейните потребители, публикации, харесвания и коментари.
   8. **'CalculateCommunityScore(int userCount, int postCount, int likeCount, int commentCount)'** - Изчислява резултата на общността въз основа на предоставената статистика.
   9. **'DeleteCommunityAsync(int communityId)'** - Премахва общност със дадения идентификатор от базата данни.

**FollowerService**

1. **Функционалност** 
   1. Предоставя всички нужни услуги за управление на следванията между потребители в социалната мрежа.
2. **Методи**:
   1. **'DoesExistAsync(string id)'** - Проверява дали последовател със зададения идентификатор съществува в базата данни.
   2. **'FindAsync(string followerId, string followedId)'** - Връща последовател с дадените идентификатори като обект 'BaseAccountViewModel'.
   3. **'AddAsync(string followerId, string followedId)'** - Добавя нов последовател към базата данни с предоставените идентификатори.
   4. **'DeleteAsync(string followerId, string followedId)'** - Премахва последовател със зададените идентификатори от базата данни.
   5. **'FindFriendsAsync(string accountId, int takeCount, int skipCount)'** - Връща приятелите на акаунта с дадения идентификатор като масив от обекти 'BaseAccountViewModel'.
   6. **'GetBySearchAsync(string search, string accountId, int takeCount, string[] skipIds)'** - Връща приятелите на потребителя, които отговарят на дадено търсене, като масив от обекти 'BaseAccountViewModel'.

**LikeService**

1. **Функционалност** 
   1. Предоставя всички нужни услуги за управление на реакциите на потребителите в социалната мрежа. Реакциите могат да бъдат сложени на публикация или коментар.
2. **Методи**:
   1. **'AddLikeToPostAsync(string accountId, int postId, bool value)'** - Добавя реакция към публикация със даден идентификатор, от потребителя с дадения идентификатор.
   2. **'AddLikeToCommentAsync(string accountId, int commentId, bool value)'** - Добавя харесване към коментар със даден идентификатор, от потребителя с дадения идентификатор.
   3. **'RemoveLikeAsync(int likeId)'** - Премахва харесване със зададения идентификатор от базата данни.

**MessageService**

1. **Функционалност**
   1. Предоставя всички нужни услуги за управление на съобщенията между потребителите в социалната мрежа.
2. **Метод**:
   1. **'FindAsync(int id)'** - Връща съобщение с дадения идентификатор като обект 'MessageViewModel'.
   2. **'GetMessagesBySenderAndReceiverIdsAsync(string senderId, string receiverId, int takeCount, DateTime date)'** - Връща съобщения между двама потребители със зададените идентификатори. Връща съобщенията преди дадената дата като масив от обекти 'MessageViewModel'. Датата спомага за идентифициране на последното заредено съобщение и предоставя възможност за зареждане на още съобщения.
   3. **'AddAsync(string senderId, string receiverId, string text)'** - Добавя ново съобщение към базата данни с предоставените идентификатори на изпращач и получател, и текст.
   4. **'RemoveAsync(int id)'** - Премахва съобщение със дадения идентификатор от базата данни.

**PostResourcesService**

1. **Функционалност** 
   1. Предоставя всички нужни услуги за управление на ресурсите (снимки или видео) към дадена публикация в социалната мрежа.
2. **Методи**:
   1. **'AddPostResourceAsync(PostResources postResource)'** - Добавя нов ресурс на дадена публикация към файловата система. Прави проверка за допустими типове на ресурса и създава случайно има, изпoлзвайки Guid. Извикава UploadFileAsync.
   2. '**UploadFileAsync(int postId, IFormFileCollection postedFiles)'** - Запазва локацията и типа на ресурсите в базата данни, както и публикацията, към която дадените ресурси принадлежат .
   3. **'GetFileAsync(string fileName)'** - Връща ресурс от файловата система с помощта на персонализиран клас 'CustomFileStreamResult'.
   4. **'GetPostResourcesByPostIdAsync(int postId)'** - Връща данни за ресурсите на дадена публикация като масив от обекти 'PostRescourcesViewModel'.

**PostService**

1. **Функционалност** 
   1. Предоставя всички нужни услуги за управление на публикациите в социалната мрежа.
2. **Методи**:
   1. **'DoesExist(int id)'** - Проверява дали публикация с даден идентификатор съществува в базата данни.
   2. **'FindAsync(int id, string accountId)'** - Връща публикация с даден идентификатор като обект 'PostViewModel'.
   3. **'AddAsync(string title, string text, string accountId, int communityId)'** - Добавя нова публикация към базата данни с предоставените параметри.
   4. **'RemoveAsync(int id)'** - Премахва публикация с даден идентификатор от базата данни.
   5. **'GetByDateAsync(string accountId, DateTime lastDate, int communityId, int takeCount)'** - Връща публикации, създадени преди дадена дата, като се вземат предвид само публикациите, които не са изтрити. Датата се използва като механизъм за зареждане на допълнителен брой публикации в последствие.
   6. **'GetTrendingAsync(int[] skipIds, int takeCount, string accountId, int communityId = 0)'** - Връща публикации, които са популярни според определен алгоритъм, като се вземат предвид само публикациите, които не се съдържат в масива ‘skipIds’. Това се използва като механизъм за зареждане на допълнителен брой публикации в последствие.
   7. **'GetFollowedPostsAsync(int[] skipIds, int takeCount, string accountId)'** - Връща публикации, публикувани от акаунти, които даден потребител следва, както и публикации в общности, които потребителят наблюдава. Вземат се предвид само публикациите, които не се съдържат в масива ‘skipIds’. Това се използва като механизъм за зареждане на допълнителен брой публикации в последствие.
   8. **'GetByCommunityAsync(int communityId)'** - Връща публикации, принадлежащи към определена общност.
   9. **'GetBySearchAsync(string search, string accountId, int takeCount, int communityId, int[]? skipIds)'** - Връща публикации, които съдържат конкретен текст в заглавието или текста си, като се вземат предвид само публикациите, които не се съдържат в масива ‘skipIds’. Това се използва като механизъм за зареждане на допълнителен брой публикации в последствие.
   10. **'IsOwnerAsync(int postId, string accountId)'** - Проверява дали даден потребител е автор на публикация с даден идентификатор.
   11. **'CalculateScore(Post post, double likeWeight, double commentWeight, double decayFactor)'** - Калкулира популярността на публикацията, използвайки бройката на коментарите и харесванията към нея.

**SignalRService**

1. **Функционалност**
   1. Предоставя всички нужни услуги за управление на SignalR хъбовете в социалната мрежа, като добавя или премахва потребител с даден идентификатор на връзката от SignalR група.
2. **Методи:**
   1. **'AddConnectionToGroup(string connectionId, string[]? groupsId)'** - Добавя връзка към една или повече групи въз основа на предоставените идентификатори на групи. Връзката се добавя към МessageHub, ако групата съдържа в себе си ключовата дума “chat”. Връзката се добавя към DeleteHub, ако групата съдържа в себе си ключовата дума “pdd” (post details delete). Връзката се добавя към LikeHub, ако групата съдържа в себе си ключовата дума “pdl” (post details like) или само число (идентификатор на публикация).
   2. **'RemoveConnectionFromAllGroups(string connectionId)'** - Премахва връзка с даден идентификатор от всички групи, към които е била добавена.

### Котролери

Контролерите в API предоставят крайни точки за клиентските приложения. Те използват определен интерфейс, описващ класовете, които предоставят връзка с базата данни и файловата система.

**AccountController**

Контролерът управлява функциите, свързани с потребителите в приложението Zest. Той използва 'IAccountService' за извършване на операциите, свързани с акаунти, включително търсене, добавяне и извличане на информация за акаунти. Той е маркиран с атрибута '[Authorize]', което гарантира, че само автентицирани потребители могат да достъпват неговите методи. Освен това, използва атрибутите '[Route("api/[controller]")]' и '[ApiController]' за определяне на маршрутизацията и поведението на API.

**Зависимости**

- 'IAccountService \_accountService': Интерфейс за услуга, която предоставя методи за работа с акаунти.

**Конструктор**

Конструкторът инициализира контролера с инстанция на 'IAccountService', използвайки Dependency Injection.

**Методи**

1. **'**FindById()**'**
   1. Маршрут: 'get'
   2. Тип на метода: 'HttpGet'
   3. Функционалност: Връща детайли за акаунта, идентифициран чрез 'NameIdentifier' в клеймовете на потребителя.
2. **'**Add(string name, string email)**'**
   1. Маршрут: 'add/{name}/{email}'
   2. Тип на метода: 'HttpPost'
   3. Параметри: 'name' (string), 'email' (string)
   4. Функционалност: Добавя нов акаунт с предоставеното потребителско име и електронна поща. Проверява за съществуващо потребителското име преди добавянето.
3. **'** GetAll(int takeCount, int skipCount = 0)**'**
   1. Маршрут: 'getAll/{takeCount}/{skipCount}'
   2. Tип на метода: 'HttpGet'
   3. Параметри: 'takeCount' (int), 'skipCount' (int)
   4. Функционалност: Връща списък с акаунти, като позволява пагинация чрез параметрите 'takeCount' и 'skipCount'.
4. **'** GetBySearch(string search, int takeCount, [FromBody] string[]? skipIds)**'**
   1. Маршрут: 'getBySearch/{search}/{takeCount}'
   2. Tип на метода: 'HttpPost'
   3. Параметри: 'search' (string), 'takeCount' (int), 'skipIds' ([FromBody] string[]?)
   4. Функционалност: Извлича акаунти, базирани на търсене с предоставените параметри. Прави проверка за празен параметър ‘search’;

**CommentsController**

Контролерът е отговорен за управлението на коментарите в приложението Zest. Този контролер използва услугите 'ICommentsService', 'IPostService' и контекста на SignalR хъба 'IHubContext<DeleteHub>' за изпълнение на операции, свързани с коментарите. Той е маркиран с атрибута '[Authorize]', което гарантира, че само автентицирани потребители могат да достъпват неговите методи. Освен това, използва атрибутите '[Route("api/[controller]")]' и '[ApiController]' за определяне на маршрутизацията и поведението на API.

**Зависимости**

- 'ICommentsService \_commentsService': Услуга за управление на коментари.

- 'IPostService \_postService': Услуга за управление на публикации.

- 'IHubContext<DeleteHub> \_deleteHubContext': Контекст на SignalR хъб за изпращане на съобщения в реално време при изтриване на коментари.

**Конструктор**

Конструкторът инициализира контролера с необходимите услуги и контексти.

**Методи**

1. **'**Find(int id)**'**
   1. Маршрут: '{id}'
   2. Тип на метода: 'HttpGet'
   3. Функционалност: Връща детайли за коментар, идентифициран по неговият идентификатор.
2. **'**Add(int postId, [FromBody] string text, int commentId = 0)**'**
   1. Маршрут: 'add/post/{postId}/comment/{commentId}'
   2. Тип на метода: 'HttpPost'
   3. Параметри: 'postId' (int), 'text' (тяло на заявката), 'commentId' (int, не задължителен)
   4. Функционалност: Позволява на потребителите да добавят коментар към публикация или друг коментар. Проверява за съществуването на публикацията или коментара преди добавянето.
3. **'**Remove(int commentId, int postId)**'**
   1. Маршрут: 'remove/{commentId}/{postId}'
   2. Тип на метода: 'HttpPut'
   3. Функционалност: Позволява премахването на коментар. Изпраща съобщение чрез SignalR хъба при успешно изтриване.
4. **'**GetCommentsByPost(int postId, [FromRoute] DateTime lastDate, int takeCount)**'**
   1. Маршрут: 'getCommentsByPost/{postId}/{lastDate}/{takeCount}'
   2. Тип на метода: 'HttpGet'
   3. Функционалност: Връща коментари за дадена публикация, поддържайки пагинация и филтриране по дата.
5. **'**GetByTrending(int takeCount, int postId, [FromBody] int[]? skipIds)**'**
   1. Маршрут: 'getByTrending/{takeCount}/{postId}'
   2. Тип на метода: 'HttpPost'
   3. Функционалност: Връща популярни коментари за дадена публикация, позволява изключването на определени коментари чрез предаване на техните идентификатори в тялото на заявката.

**LikesController**

Контролерът отговаря за управлението на действията за реакциите върху публикации и коментари в приложението Zest. Този контролер използва различни услуги за изпълнението на своите функции - 'ILikeService', 'IHubContext<LikesHub>', 'IPostService' и 'ICommentsService'. Той е маркиран с атрибута '[Authorize]', което гарантира, че само автентицирани потребители могат да достъпват неговите методи. Освен това, използва атрибутите '[Route("api/[controller]")]' и '[ApiController]' за определяне на маршрутизацията и поведението на API.

**Зависимости**

- 'ILikeService \_likeService': Услуга за обработка на операции, свързани с реакциите.

- 'IHubContext<LikesHub> \_likesHubContext': Контекст на хъба SignalR за комуникация в реално време относно реакциите.

- 'IPostService \_postService': Услуга за операции, свързани с публикации.

- 'ICommentsService \_commentsService': Услуга за операции, свързани с коментари.

**Конструктор**

Конструкторът инициализира контролера с инстанции на услугите, необходими за неговите операции.

**Методи**

1. **'**Add(int postId, int commentId, bool value)**'**
   1. Маршрут: 'add/post/{postId}/comment/{commentId}/value/{value}'
   2. Тип на метода: 'HttpPost'
   3. Параметри: 'postId' (int), 'commentId' (int), 'value' (bool)
   4. Функционалност: Този метод позволява на потребителите да добавят харесване към публикация или коментар. Проверява съществуването на посочената публикация или коментар, преди да продължи. При добавяне на харесване към публикация, извиква 'AddLikeToPostAsync' от '\_likeService'; за коментари използва 'AddLikeToCommentAsync'. Успешните действия за харесване се комуникират към съответните клиенти чрез SignalR, използвайки '\_likesHubContext'.
2. **'**Remove(int likeId, int postId, int commentId)**'**
   1. Маршрут: 'remove/like/{likeId}/{postId}/{commentId}'
   2. Тип на метода: 'HttpDelete'
   3. Параметри: 'likeId' (int), 'postId' (int), 'commentId' (int)
   4. Функционалност: Този метод позволява на потребителите да премахнат харесване от публикация или коментар. Проверява за съществуването на публикацията или коментара преди да продължи.

**CommunityController**

Контролерът е предназначен за управление на общностите в приложението Zest. Той използва услугите 'ICommunityService', 'ICommunityFollowerService' и 'IAccountService' за изпълнение на операции, свързани с общностите. Той е маркиран с атрибута '[Authorize]', което гарантира, че само автентицирани потребители могат да достъпват неговите методи. Освен това, използва атрибутите '[Route("api/[controller]")]' и '[ApiController]' за определяне на маршрутизацията и поведението на API.

**Зависимости**

- 'ICommunityService \_communityService': Услуга за управление на общности.

- 'ICommunityFollowerService \_communityFollowerService': Услуга за управление на последователите на общности.

- 'IAccountService \_accountService': Услуга за управление на акаунтите.

**Конструктор**

Конструкторът инициализира контролера с необходимите услуги за изпълнението на неговите функции.

**Методи**

1. **'**Find(int id)**'**
   1. Маршрут: '{id}'
   2. Тип на метода: 'HttpGet'
   3. Функционалност: Връща детайли за общността, идентифицирана по нейният идентификатор.
2. **'**GetAll(int takeCount, int skipCount = 0)**'**
   1. Маршрут: 'getAll/{takeCount}/{skipCount}'
   2. Тип на метода: 'HttpGet'
   3. Функционалност: Връща списък на всички общности, като поддържа пагинация. Проверява дали текущият акаунт е абониран за всяка от тях.
3. **'**Add(string name, [FromBody] string discription)**'**
   1. Маршрут: 'add/{name}'
   2. Тип на метода: 'HttpPost'
   3. Функционалност: Създава нова общност с предоставеното име и описание.
4. **'**Delete(int communityId)**'**
   1. Маршрут: 'delete/{communityId}'
   2. Тип на метода: 'HttpDelete'
   3. Функционалност: Изтрива общност, ако съществува.
5. **'**GetCommunitiesByAccount(string accountId, int takeCount, int skipCount = 0)**'**
   1. Маршрут: 'getByAccountId/{accountId}/{takeCount}/{skipCount}'
   2. Тип на метода: 'HttpGet'
   3. Функционалност: Връща общности, асоциирани с даден акаунт, като поддържа пагинация.
6. **'**GetCommunitiesByPopularity(int takeCount,[FromBody] int[]? skipIds)**'**
   1. Маршрут: 'getByPopularityId/{takeCount}'
   2. Тип на метода: 'HttpPost'
   3. Функционалност: Връща трендиращи общности, като позволява изключване на някои чрез 'skipIds'.
7. **'**GetBySearch(string search, int takeCount, [FromBody] int[]? skipIds)**'**
   1. Маршрут: 'getBySearch/{search}/{takeCount}'
   2. Тип на метода: 'HttpPost'
   3. Функционалност: Връща общности, които отговарят на търсената фраза, като поддържа пагинация и позволява изключване на някои чрез 'skipIds'.

**CommunityFollowersController**

Контролерът управлява функциите за последователи на общности в приложението Zest. Той използва услугата 'ICommunityFollowerService' за да осъществява операции, свързани с добавянето и изтриването на последователи от общности. Контролерът е маркиран с атрибутите '[Authorize]', '[Route("api/[controller]")]' и '[ApiController]', гарантирайки, че само автентицирани потребители могат да достъпват неговите методи и уточнявайки как API трябва да приема и обработва заявки.

**Зависимости**

- 'ICommunityFollowerService, \_communityFollowerService': Услуга, предоставяща методи за управление на последователите на общности.

**Конструктор**

Конструкторът инициализира контролера с необходимата услуга за управление на последователите на общности. Инжектира интерфейсите чрез Dependency Injection.

**Методи**

1. **'**DoesExist(int communityId)**'**
   1. HTTP Метод: 'HttpGet'
   2. Функционалност: Проверява дали даден акаунт вече следва определена общност, като използва идентификатора на акаунта и идентификатора на общността.
2. **'**Add(int communityId)**'**
   1. Маршрут: 'account/add/community/{communityId}'
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Позволява на акаунта на текущия потребител да стане последовател на общност, указана чрез 'communityId', при условие че вече не следва тази общност.
3. **'**Delete(int communityId**'**
   1. Маршрут: 'account/delete/community/{communityId}'
   2. HTTP Метод: 'HttpDelete'
   3. Функционалност: Премахва акаунта на текущия потребител от списъка на последователите на дадена общност, указана чрез 'communityId', при условие че вече следва тази общност.

**CommunityModeratorsController**

Контролерът управлява функционалността, свързана с модераторите на общности в приложението Zest. Той използва услугите 'ICommunityModeratorService', 'ICommunityService' и 'IAccountService' за извършване на различни операции като проверка за модераторски статус, добавяне и премахване на модератори, както и управление на кандидатите за модератори. Контролерът е маркиран с '[Route("api/[controller]")]' и '[ApiController]', а някои от неговите методи изискват аутентикация чрез '[Authorize]'.

**Зависимости**

- 'ICommunityModeratorService \_communityModeratorService': Услуга за управление на модераторите на общности.

- 'ICommunityService \_communityService': Услуга за управление на общности.

- 'IAccountService \_accountService': Услуга за управление на акаунти.

**Конструктор**

Конструкторът инициализира контролера с необходимите услуги за управление на модераторите и общностите.

**Методи**

1. **'**Find(string accountId, int communityId)**'**
   1. Маршрут: 'isModerator/{accountId}/{communityId}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Проверява дали даден потребител е модератор на конкретна общност.
2. **'**Add(string accountId, int communityId)**'**
   1. Маршрут: 'add/{accountId}/{communityId}'
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Добавя акаунта като кандидат-модератор на указаната общност, ако той не е вече кандидат.
3. **'**GetModeratorsByCommunity(int communityId)**'**
   1. Маршрут: 'getModerators/{communityId}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Връща списък на модераторите на дадена общност.
4. **'**GetModeratorCandidatesByCommunity(int communityId)**'**
   1. Маршрут: 'getCandidates/{communityId}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Връща списък на кандидатите за модератори на дадена общност.
5. **'**ApproveCandidate(string accountId, int communityId)**'**
   1. Маршрут: 'approveCandidate/{accountId}/{communityId}'
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Одобрява кандидатурата на акаунт за модератор на указаната общност.
6. **'**RemoveModerator(string accountId, int communityId)**'**
   1. Маршрут: 'removeModerator/{accountId}/{communityId}'
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Премахва акаунта от списъка на модераторите на указаната общност.

**FollowersController**

Контролерът е предназначен за управление на функционалностите, свързани с последователите в приложението Zest. Използвайки услугите 'IFollowerService' и 'IAccountService', този контролер предоставя възможност за търсене, добавяне и изтриване на последователи, както и за извличане на информация за приятелите и търсене сред тях. Той е маркиран с атрибута '[Authorize]', което гарантира, че само автентицирани потребители могат да достъпват неговите методи. Освен това, използва атрибутите '[Route("api/[controller]")]' и '[ApiController]' за определяне на маршрутизацията и поведението на API.

**Зависимости**

- 'IFollowerService \_followerService': Услуга за управление на последователи.

- 'IAccountService \_accountService': Услуга за управление на акаунти.

**Конструктор**

Конструкторът на 'FollowersController' инициализира контролера с необходимите услуги за управление на последователи и акаунти.

**Методи**

1. **'** Find(string followerId, string followedId)**'**
   1. Маршрут: 'followers/find/{followerId}/{followedId}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Търси и връща информация за връзката между два акаунта, като идентифицира дали едният следва другия.
2. **'** Add(string followedId)**'**
   1. Маршрут: 'add/followed/{followedId}'
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Добавя текущия автентикиран потребител като последовател на акаунт, указан с 'followedId'.
3. **'** Delete(string followedId)**'**
   1. Маршрут: 'delete/followed/{followedId}'
   2. HTTP Метод: 'HttpDelete'
   3. Функционалност: Премахва текущия автентикиран потребител от списъка на последователите на акаунт, указан с 'followedId'.
4. **'**FindFriends(int takeCount, int skipCount = 0)**'**
   1. Маршрут: 'getFriends/{takeCount}/{skipCount}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Връща списък на акаунтите, които са взаимни последователи (приятели) с текущия автентикиран потребител, поддържайки пагинация. Под приятели се разбират и администраторите на мрежата.
5. **'**GetBySearch(string search, int takeCount, [FromBody] string[]? skipIds)**'**
   1. Маршрут: 'getBySearch/{search}/{takeCount}'
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Търси сред приятелите на текущия автентикиран потребител по зададен критерий 'search', поддържайки пагинация и възможност за пропускане на определени акаунти чрез 'skipIds'.

**MessagesController**

Контролерът предоставя API за управление на съобщения в приложението Zest. Използвайки услугите 'IMessageService', 'IHubContext<MessageHub>' и 'IAccountService', контролерът позволява на потребителите да извличат, добавят и взаимодействат със съобщения в реално време. Той е маркиран с атрибута '[Authorize]', което гарантира, че само автентицирани потребители могат да достъпват неговите методи. Освен това, използва атрибутите '[Route("api/[controller]")]' и '[ApiController]' за определяне на маршрутизацията и поведението на API.

**Зависимости**

- 'IMessageService \_messageService': Услуга за управление на съобщенията.

- 'IHubContext<MessageHub> \_hubContext': Контекст на SignalR хъб за реално временно взаимодействие.

- 'IAccountService \_accountService': Услуга за управление на акаунти.

**Конструктор**

Конструкторът инициализира контролера с необходимите услуги за работа със съобщения и акаунти.

**Методи**

1. **'**Find(int id)**'**
   1. Маршрут: 'get/{id}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Връща конкретно съобщение, идентифицирано чрез неговият идентификатор.
2. **'**GetMessagesByReceiverId(string receiverId, int takeCount, [FromRoute]DateTime date)**'**
   1. Маршрут: 'get/receiver/{receiverId}/{takeCount}/{date}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Връща списък на съобщения, изпратени към определен получател, поддържайки пагинация и филтриране по дата.
3. **'**Add(string receiverId, [FromBody] string text)**'**
   1. Маршрут: 'add/receiver/{receiverId}'
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Позволява на потребителя да изпрати съобщение към указания получател. Идентификаторът на изпратеното съобщение се връща като резултат и се изпраща сигнал чрез SignalR хъба към клиентите, абонирани за съответната група.

Контролерът използва SignalR за да осигури взаимодействието в реално време между потребителите, улеснявайки комуникацията чрез съобщения. Взаимодействието чрез SignalR позволява на приложението да уведомява потребителите за нови съобщения без необходимостта от ръчно обновяване или проверка за нови съобщения.

**PostController**

Контролерът управлява функциите, свързани с публикациите в приложението Zest. Използва услугите 'IPostService', 'ICommunityService' и контекста на SignalR хъб 'IHubContext<DeleteHub>' за реализацията на операции като добавяне, премахване и извличане на публикации. Той е маркиран с атрибута '[Authorize]', което гарантира, че само автентицирани потребители могат да достъпват неговите методи. Освен това, използва атрибутите '[Route("api/[controller]")]' и '[ApiController]' за определяне на маршрутизацията и поведението на API.

**Зависимости**

- 'IPostService \_postService': Услуга за управление на публикации.

- 'IHubContext<DeleteHub> \_hubContext': Контекст на SignalR хъб за изпращане на уведомления за изтрити публикации.

- 'ICommunityService \_communityService': Услуга за управление на общности.

**Конструктор**

Конструкторът инициализира контролера с необходимите услуги за управление на публикации и общности.

**Методи**

1. **'**Find(int id)**'**
   1. Маршрут: '{id}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Извлича публикация по нейният идентификатор.
2. **'**Add(string title, [FromBody] string text, int communityId)**'**
   1. Маршрут: 'add/{title}/community/{communityId}'
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Добавя нова публикация с предоставените заглавие и текст в указаната общност.
3. **'**Remove(int postId)**'**
   1. Маршрут: 'remove/{postId}'
   2. HTTP Метод: 'HttpPut'
   3. Функционалност: Изтрива публикация по нейният идентификатор и изпраща уведомление чрез SignalR.
4. **'**GetByDate([FromRoute] DateTime lastDate, int communityId, int takeCount)**'**
   1. Маршрут: 'getByDate/{lastDate}/{communityId}/{takeCount}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Връща публикации според дата, общност и брой за извличане.
5. **'**GetByCommunity(int communityId)**'**
6. Маршрут: 'getByCommunity/{communityId}'
7. HTTP Метод: 'HttpGet'
8. Функционалност: Извлича всички публикации в указана общност.
9. **'** GetBySearch(string search, int takeCount, int communityId, [FromBody] int[]? skipIds)**'**
   1. Маршрут: 'getBySearch/{search}/{takeCount}/{communityId}'
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Връща публикации, които отговарят на зададени критерии за търсене, брой за извличане и общност.
10. **'** GetByTrending(int takeCount, int communityId,[FromBody] int[]? skipIds)**'**
    1. Маршрут: 'getByTrending/{takeCount}/{communityId}'
    2. HTTP Метод: 'HttpPost'
    3. Функционалност: Връща трендиращи публикации в указаната общност с опция за пропускане на някои по идентификатор.
11. GetByFollowed(int takeCount, [FromBody] int[]? skipIds)
    1. Маршрут: 'getByFollowed/{takeCount}'
    2. HTTP Метод: 'HttpPost'
    3. Функционалност: Връща публикации от акаунти, които текущият потребител следва, с опция за пропускане на някои по идентификатор.

Този контролер играе ключова роля в управлението на съдържанието в приложението, като предоставя потребителите с гъвкави опции за работа с публикации.

**PostResourcesController**

Контролерът управлява функциите, свързани със ресурсите (снимки и видеа) към публикациите в приложението Zest. Използва услугите 'IPostService', 'IPostResourcesService' за реализацията на операции като добавяне и извличане на ресурси към дадена публикация. Използва атрибутите '[Route("api/[controller]")]' и '[ApiController]' за определяне на маршрутизацията и поведението на API.

**Зависимости**

- 'IPostService \_postService': Услуга за управление на публикации.

- 'IPostResourcesService \_postResourceServic'e: Услуга за управление на ресурси към публикации.

**Конструктор**

Конструкторът инициализира контролера с необходимите услуги за управление на публикации и ресурсите към тях.

**Методи**

1. **'**UploadFile(int postId, IFormFileCollection postedFiles)**'**
   1. Маршрут: '"uploadFile/{postId}"'
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Качва колекция от файлове и прави връзка с публикацията, към която принадлежат.
2. **'**GetFile(string fileName)**'**
   1. Маршрут: 'get/{fileName}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Взима ресурс с даденото име от файловата система.
3. **'**GetPhotos(int postId)**'**
   1. Маршрут: 'getByPostId/{postId}'
   2. HTTP Метод: 'HttpGet'
   3. Функционалност: Връща ресурси към публикация с даден идентификатор

**SignalRGroupsController**

Контролерът управлява функциите, свързани със групите в SignalR хъбовете в приложението Zest. Използва услугите 'ISignaRService' за реализацията на операции като добавяне и премахване на дадена връзка към група. Използва атрибутите '[Route("api/[controller]")]' и '[ApiController]' за определяне на маршрутизацията и поведението на API.

**Зависимости**

-'ISignaRService \_signalRService': Услуга за управление на SigналR групи.

**Конструктор**

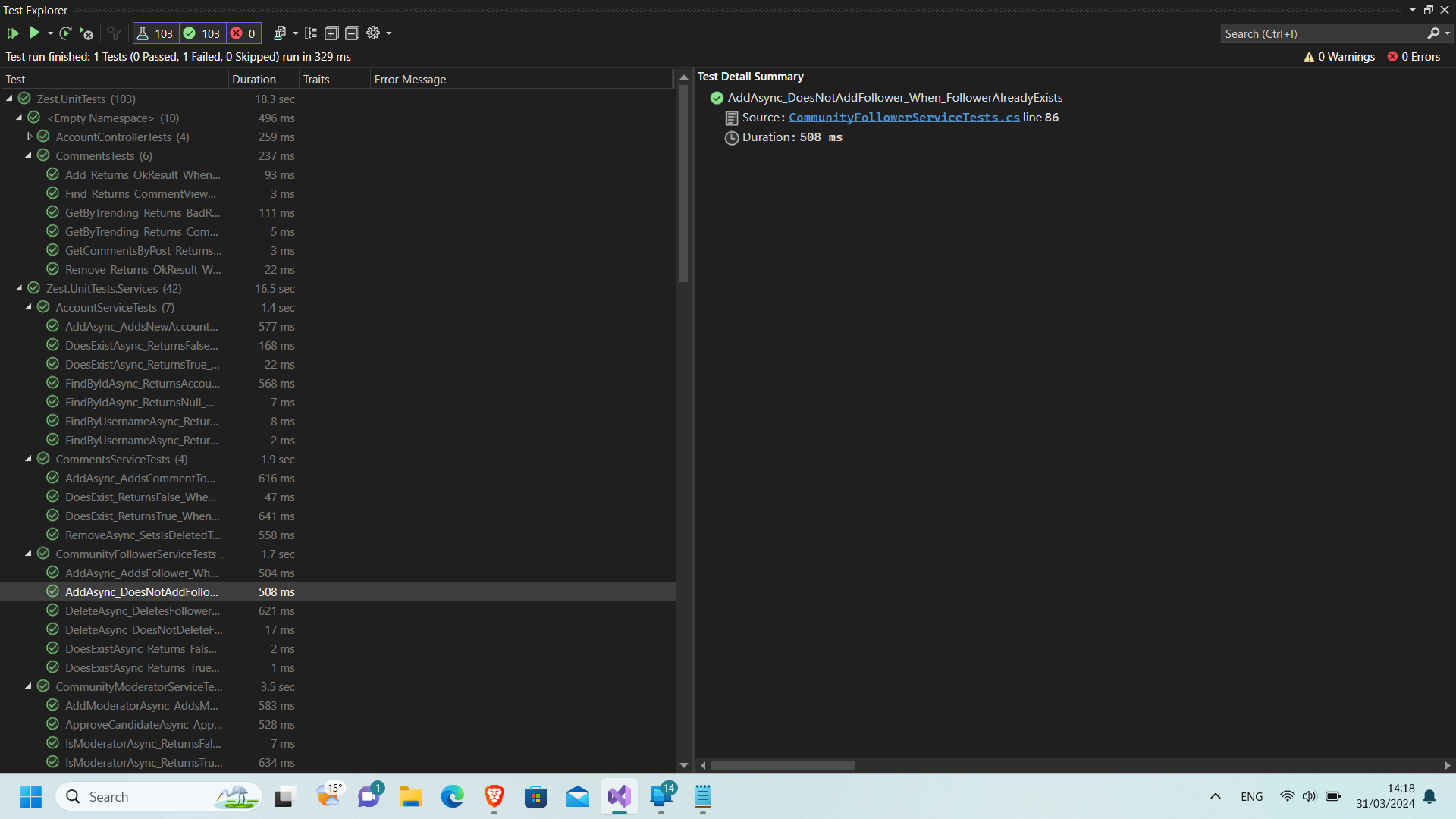
Конструкторът инициализира контролера с необходимите услуги за управление на групите в SignalR.

**Методи**

1. **'**AddConnectionToGroup(string connectionId, [FromBody]string[]? groupsId)**'**
   1. Маршрут:"addConnectionToGroup/{connectionId}"
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Добавя връзка към дадени групи.
2. **'**RemoveConnectionFromAllGroups(string connectionId)**'**
   1. Маршрут: "removeConnectionToGroup/{connectionId}"
   2. HTTP Метод: 'HttpPost'
   3. Функционалност: Премахва връзка от всички групи

Контролерът играе ключова роля в предоставянето на функционалности в реално време.

### Testing

Надеждността на приложението се подсигурява от 103 успешно завършили теста.

### SignalR Hubs

С цел по-добър UX, API на Zest предоставя възможност за връзка с SignalR хъбове, които предоставят функционалност в реално време. Хъбовете са разделени на три, всеки отговарящ за специфична задача.

1. **DeleteHub**
   1. Отговаря за нотификация при изтриване на публикации и коментари. Групите използват ключовота наименование pdd (post details delete), следвано от идентификатор на публикацията.
   2. Използване: Хъбът се инжектира в PostController и CommentsController и се използва при изтриване.
2. **LikesHub**
   1. Отговаря за нотификация при реакция на публикации и коментари. Групите използват ключовота наименование pdl (post details like), следвано от идентификатор на публикацията или само идентификатор на публикацията.
   2. Използване: Хъбът се инжектира в LikesController и се използва при добавяне на реакция.
3. **MessageHub**
   1. Отговаря за нотификация при изпращане на съобщения между потребителите. Осигурява взаимодействието в реално време между потребителите, улеснявайки комуникацията чрез съобщения. Взаимодействието чрез хъба позволява на приложението да уведомява потребителите за нови съобщения без необходимостта от ръчно обновяване или проверка за нови съобщения. За групите се иползва ключовото наименование chat, последвано от идентификаторите на потребителите в чата. Идентификаторите се сръвняват, за да може и двамата потребители да присъстват в една и съща група
   2. Използване: Хъбът се инжектира в MessageController и се използва при добавяне на съобщение.

### Мапиране

Zest използва AutoMapper за мапиране на DB моделите и DTOs.

## Zest Frontend

Клиентското приложение на Zest използва технологията за cross-platform native applications .Net MAUI. Технологията е избрана зареди нейното таргетиране на платформите Windows, Android, Tyzen и IOS. Трябва да се има впредвид, че технологията е сравнително нова и все още се развива. Приложението е тествано на Windows и Android. Използва се Shell application. Приложение следва MVVM модел.

### Страници

Приложението има общо 14 страници. Всяка от тях има своето предназначение. Има разлики в страниците в зависимост от платформата, която използва потребителя.

1. MainPage (ContentPage)
   1. Начална страница на приложението. Предоставя функционалност за влизане в акаунт в приложението чрез външен доставчик за автентикация.
2. AccountPage (ContentPage)
   1. Страница с детайли за текущия потребител. Предоставя функционалност за излизане на акаунта.
3. AddCommunityPage (ContentPage)
   1. Страница с функционалност за създаване на общност.
4. AddPostPage (ContentPage)
   1. Страница с функционалност за създаване на публикация към общност
5. ChatPage (ContentPage)
   1. Страница, която визуализира съобщенията на текущия потребител с негов приятел в мрежата.
6. CommentDetailsPage (ContentPage)
   1. Страница, която визуализира коментар и неговите отговори. Предоставя функционалност за харесване, изтриване и публикуване на отговор към съответен коментар.
7. CommuntiesPage (ContentPage)
   1. Страницата зарежда общностите в мрежата. Подрежда ги по даден филтър - All, Trending и Followed. Предоставя възможност за търсене на общност.
8. CommunityDetailsPage (ContentPage)
   1. Страницата визуализира детайлите на дадена общност, включително публикациите към нея, моито могат да се филтрират по Latest и Trending.
9. CommunityModeratorsPage (ContentPage)
   1. Страницата зарежда модераторите на дадена общност. Предоставя функционалност за одобряване или отхвърляне на кандидат-модератори (Функционалността е достъпна само за текущите модератори). Предоставя възможност за кандидатстване за модератор или премахване на текущия потребител като такъв.
10. FriendsPage (ContentPage)
    1. Страницата зарежда приятелите на текущия потребител. Предоставя възможност за търсене сред тях.
11. PostDetailsPage (ContentPage)
    1. Страницата визуализира дадена публикация и коментарите към нея. Предоставя функционалност за харесване и изтриване на публикацията или на някой от коментарите към нея.
12. PostsPage (ContentPage)
    1. Страницата зарежда публикации по даден филтър - Latest, Trending и Followed
13. UserDetailsPage (ContentPage)
    1. Страницата показва детайли за чуждите потребители в мрежата. Дава опция за последване на дадения потребител и зарежда негови следвани общности
14. UsersPage (ContentPage)
    1. Страницата зарежда другите потребители в мрежата. Предоставя възможност за търсене сред тях.

BindingContext на страниците сочи към съответния ViewModel.

### Навигация

За телефон, приложението се възползва от Tabs. За Desktop е създадено специално NavigationView (ContentView) с цел удобство за потребителите.

HTTP Services

В ViewModel се инжектират чрез DI нужните Http услуги за достъп да Zest API.

### SignalR

Клиентско приложение се възползва от предоставяната от API услуга за функционалност в реално време. Добавяне и премахване от групи обикновено става при навигация между страниците и зареждане на дадено съдържание.

## Authentication and Authorization

Тези услуги се предоставят от външен доставчик (Oauth2 by Okta). В тяхната система се осъществява контрол над потребителите от страна на администраторите на социалната мрежа. Пазенето на IdToken и AccessToken в клиентското приложение спазва всички препоръки на доставчика на услугата.

# Заключение

Системата Zest предоставя всички нужни функционалности за споделяне на научно съдържание и провежда на дискусии по дадена научна тема. Има потенциала да се превърне в основна платформа за комуникация в научната общност и да помогне за популяризиране на науката сред младите.

## Бъдещо развитие

#### Уеб приложение

Въпреки удобството, което .Net MAUI предоставя, голяма част от използващите социални мрежи предпочитат използването на уеб приложения. Зареди разделянето на системата на API и клиентско приложение, тази цел е осъществими в средносрочен план.

#### Real Time functionality

Въпреки че Zest предоставя такава функционалност за най-използваните методи в мрежата, разширяването на SignalR ще подобри значително UX.