File

```
src/app/services/user.service.ts
```

Description

Service that contains all the routes to our api regarding users

Index

Properties

- Private baseUrl
- Public identity
- Public token

Methods

- Public adminDeleteUser
- Public allusers
- Public checkFollow
- Public countAllHeroes
- Public countAllTeams
- Public countAllUsers
- Public deleteUser
- Public followUser
- Public getFollowersUsers
- Public getFollowUsers
- Public getIdentity
- Public getToken
- Public getTokenInfo
- Public getUser
- Public getUserByName
- Public isLoggedIn
- Public LoginUser
- Public newImg
- Public RegisterUser
- Public unFollowUser
- Public updatePass
- Public updateUser

Constructor

```
constructor(http: HttpClient)
```

Defined in src/app/services/user.service.ts:26

Constructor in which we inject httpClient to make http requests

Parameters :

Name	Type	Optional
http	`HttpClient`	No

Methods

Public adminDeleteUser

`adminDeleteUser(idUsu: number)`

Defined in src/app/services/user.service.ts:248

Delete a user

Parameters :

Name	Type	Optional
idUsu	number	No

Returns : `any`

Public allusers

`allusers()`

Defined in src/app/services/user.service.ts:236

Get all the users

Returns : `Observable<User[]>`

User[]

Public checkFollow

`checkFollow(idUsu: number, idUnfollow: number)`

Defined in src/app/services/user.service.ts:199

Check whether a user is being followed

Parameters :

Name	Type	Optional
idUsu	number	No
idUnfollow	number	No

Returns : `any`

any

Public countAllHeroes

`countAllHeroes()`

Defined in src/app/services/user.service.ts:282

Get heroes count

Returns : `Observable<any>`

count

Public countAllTeams

`countAllTeams()`

Defined in src/app/services/user.service.ts:271

Get teams count

Returns : `Observable<any>`

count

Public countAllUsers

`countAllUsers()`

Defined in src/app/services/user.service.ts:260

Get users count

Returns : `Observable<any>`

users

Public deleteUser

`deleteUser(idUsu: number, data)`

Defined in src/app/services/user.service.ts:153

Delete a user

Parameters :

Name	Type	Optional
idUsu	number	No
data		No

Returns : `any`

Public followUser

`followUser(ides)`

Defined in src/app/services/user.service.ts:165

Follow a user

Parameters :

Name	Optional
ides	No

Returns : `any`

Public getFollowersUsers

`getFollowersUsers(idUsu: number)`

Defined in src/app/services/user.service.ts:217

Get all the users who follows you

Parameters :

Name	Type	Optional
idUsu	number	No

Returns : `Observable<User[]>`

User[]

Public getFollowUsers

`getFollowUsers(idUsu: number)`

Defined in src/app/services/user.service.ts:208

Get all the users you follow

Parameters :

Name	Type	Optional
idUsu	number	No

Returns : `Observable<User[]>`

User[]

Public getIdentity

`getIdentity()`

Defined in src/app/services/user.service.ts:58

Get user from localstorage

Returns : `any`

identity

Public getToken

`getToken()`

Defined in src/app/services/user.service.ts:72

Get token from localstorage

Returns : `any`

token

Public getTokenInfo

`getTokenInfo()`

Defined in src/app/services/user.service.ts:86

translate the token to catch the expiration date

Returns : `any`

token expiration date

Public getUser

`getUser(idUsu: number)`

Defined in src/app/services/user.service.ts:119

Get user

Parameters :

Name	Type	Optional
idUsu	`number`	No

Returns : `Observable<User>`

user

Public getUserByName

`getUserByName(alias: string)`

Defined in src/app/services/user.service.ts:189

Get users by alias

Parameters :

Name	Type	Optional
alias	`string`	No

Returns : `Observable<User[]>`

user[]

Public isLoggedIn

`isLoggedIn()`

Defined in src/app/services/user.service.ts:104

Check token expiration time

Returns : `boolean`

Public LoginUser

`LoginUser(user)`

Defined in src/app/services/user.service.ts:47

Login User

Parameters :

Name	Optional
user	No

Returns : `any`

Public newImg

```
newImg(data)
```
Defined in src/app/services/user.service.ts:225

Update profile picture

Parameters :

Name	Optional
data	No

Returns : `any`

Public RegisterUser

```
RegisterUser(user)
```
Defined in src/app/services/user.service.ts:37

Register User

Parameters :

Name	Optional
user	No

Returns : `any`

Public unFollowUser

```
unFollowUser(idUsu: number, idUnfollow: number)
```
Defined in src/app/services/user.service.ts:177

Unfollow a user

Parameters :

Name	Type	Optional
idUsu	`number`	No
idUnfollow	`number`	No

Returns : `any`

Public updatePass

```
updatePass(idUsu: number, data)
```
Defined in src/app/services/user.service.ts:141

Update a user's password

Parameters :

Name	Type	Optional
idUsu	`number`	No
data		No

Returns : `any`

Public updateUser

```
updateUser(idUsu: number, user)
```
Defined in src/app/services/user.service.ts:128

Update a user

Parameters :

Name	Type	Optional
idUsu	`number`	No
user		No

Returns : `any`

Properties

Private baseUrl

Type : `string`

Default value : `environment.BASE_API_URL`

Defined in src/app/services/user.service.ts:18

Variable in which we store the url of our api

Public identity

Defined in src/app/services/user.service.ts:22

Variable in which we store the token info about the user

Public token

Defined in src/app/services/user.service.ts:26

Variable in which we store the token

```
import { Injectable } from "@angular/core";
import { environment } from "src/environments/environment";
import { HttpClient, HttpHeaders } from "@angular/common/http";
import { User } from '../models/user';
import { Observable } from 'rxjs';

/**
 * Service that contains all the routes to our api regarding users
 */
@Injectable({
  providedIn: "root",
})

export class UserService {
  /**
  * Variable in which we store the url of our api
  */
  private baseUrl: string = environment.BASE_API_URL;
  /**
  * Variable in which we store the token info about the user
  */
  public identity;
  /**
  * Variable in which we store the token
  */
  public token;

  /**
   * Constructor in which we inject httpClient to make http requests
   */
  constructor(private http: HttpClient) { }

  /**
  * Register User
  * @param {User} user
  */
  public RegisterUser(user) {
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    return this.http.post(this.baseUrl + '/auth/signup', user,
      { headers: headers });
  }

  /**
   * Login User
   * @param {User} user
   */
  public LoginUser(user) {
    // if toker equals null user takes the token property
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    return this.http.post(this.baseUrl + '/auth/signin', user,
      { headers: headers });
  }

  /**
   * Get user from localstorage
   * @returns identity
   */
  public getIdentity() {
    let identity = JSON.parse(localStorage.getItem('User'));
    if (identity != 'undefined') {
      this.identity = identity;
    } else {
      this.identity = null;
    }
    return this.identity;
  }

  /**
```

```
* Get token from localstorage
* @returns token
*/
 public getToken() {
   let token = localStorage.getItem('token');
   if (token != 'undefined') {
     this.token = token;
   } else {
     this.token = null;
   }
   return this.token;
 }

 /**
 * translate the token to catch the expiration date
 * @returns token expiration date
 */
 public getTokenInfo() {
   let token = (localStorage.getItem('token'));
   let payload;
   if (token) {
     // we cut the token from the point that interests us
     payload = token.split('.')[1];
     // we use window.atob which decodes a data string that has been encoded using base-64 encoding
     payload = window.atob(payload);
     return JSON.parse(payload);
   } else {
     return null;
   }
 }

 /**
 * Check token expiration time
 * @returns {boolean}
 */
 public isLoggedIn(): boolean {
   const user = this.getTokenInfo();
   if (user) {
     // return user.exp > (Date.now() / 1000) + (60 * 60);  // Token de una hora de duración
     return user.exp > Date.now() / 1000;
   } else {
     return false;
   }
 }

 /**
  * Get user
  * @param {number} idUsu
  * @returns {User} user
  */
 public getUser(idUsu: number): Observable<User> {
   return this.http.get<User>(`${this.baseUrl}/user/${idUsu}`);
 }

 /**
 * Update a user
 * @param {number} idUsu
 * @param {User} user
 */
 public updateUser(idUsu: number, user) {
   let headers = new HttpHeaders().set('Content-Type', 'application/json');
   // we add the token to the header
   let tokenAuth = (localStorage.getItem('token'));
   headers = headers.set("Authorization", `${tokenAuth}`);
   return this.http.put(`${this.baseUrl}/user/${idUsu}`, user, { headers: headers });
 }

 /**
 * Update a user's password
 * @param {number} idUsu
 * @param {any} data
 */
 public updatePass(idUsu: number, data) {
   let headers = new HttpHeaders().set('Content-Type', 'application/json');
   let tokenAuth = (localStorage.getItem('token'));
   headers = headers.set("Authorization", `${tokenAuth}`);
   return this.http.put(`${this.baseUrl}/user/newpass/${idUsu}`, data, { headers: headers });
 }

 /**
 * Delete a user
 * @param {number} idUsu
 * @param {any} data
 */
 public deleteUser(idUsu: number, data) {
   let headers = new HttpHeaders().set('Content-Type', 'application/json');
   let tokenAuth = (localStorage.getItem('token'));
```

```
    headers = headers.set("Authorization", `${tokenAuth}`);
    return this.http.post(`${this.baseUrl}/user/deleteUser/${idUsu}`, data,
      { headers: headers });
  }

  /**
   * Follow a user
   * @param {any} ides
   */
  public followUser(ides) {
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    let tokenAuth = (localStorage.getItem('token'));
    headers = headers.set("Authorization", `${tokenAuth}`);
    return this.http.post(`${this.baseUrl}/user/followUser`, ides,
      { headers: headers });
  }

  /**
  * Unfollow a user
  * @param {any} ides
  */
  public unFollowUser(idUsu: number, idUnfollow: number) {
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    let tokenAuth = (localStorage.getItem('token'));
    headers = headers.set("Authorization", `${tokenAuth}`);
    return this.http.delete(`${this.baseUrl}/user/unFollowUser/${idUsu}/${idUnfollow}`, { headers: headers });
  }

  /**
 * Get users by alias
 * @param {string} alias
 * @returns user[]
 */
  public getUserByName(alias: string): Observable<User[]> {
    return this.http.get<User[]>(`${this.baseUrl}/user/getUserByName/${alias}`);
  }

  /**
  * Check whether a user is being followed
  * @param {number} idUsu
  * @param {number} idUnfollow
  * @returns any
  */
  public checkFollow(idUsu: number, idUnfollow: number) {
    return this.http.get<any>(`${this.baseUrl}/user/checkFollow/${idUsu}/${idUnfollow}`);
  }

  /**
    * Get all the users you follow
    * @param {number} idUsu
    * @returns User[]
    */
  public getFollowUsers(idUsu: number): Observable<User[]> {
    return this.http.get<User[]>(`${this.baseUrl}/user/getFollowUsers/${idUsu}`);
  }

  /**
  * Get all the users who follows you
  * @param {number} idUsu
  * @returns User[]
  */
  public getFollowersUsers(idUsu: number): Observable<User[]> {
    return this.http.get<User[]>(`${this.baseUrl}/user/getFollowersUsers/${idUsu}`);
  }

  /**
  * Update profile picture
  * @param {any} data
  */
  public newImg(data) {
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    let tokenAuth = (localStorage.getItem('token'));
    headers = headers.set("Authorization", `${tokenAuth}`);
    return this.http.put(`${this.baseUrl}/user/newImg/user`, data, { headers: headers });
  }

  /**
   * Get all the users
   * @returns User[]
   */
  public allusers(): Observable<User[]> {
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    let tokenAuth = (localStorage.getItem('token'));
    headers = headers.set("Authorization", `${tokenAuth}`);
    return this.http.get<User[]>(`${this.baseUrl}/admin/allusers`, { headers: headers });
  }
  /**
    * Delete a user
    * @param {number} idUsu
    */
  public adminDeleteUser(idUsu: number) {
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    let tokenAuth = (localStorage.getItem('token'));
    headers = headers.set("Authorization", `${tokenAuth}`);
    return this.http.delete(`${this.baseUrl}/admin/${idUsu}`,
      { headers: headers });
  }

  /**
   * Get users count
   * @returns users
   */
  public countAllUsers(): Observable<any> {
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    let tokenAuth = (localStorage.getItem('token'));
    headers = headers.set("Authorization", `${tokenAuth}`);
    return this.http.get<any>(`${this.baseUrl}/admin/countAllUsers`, { headers: headers });
  }

  /**
   * Get teams count
   * @returns count
   */
  public countAllTeams(): Observable<any> {
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    let tokenAuth = (localStorage.getItem('token'));
    headers = headers.set("Authorization", `${tokenAuth}`);
    return this.http.get<any>(`${this.baseUrl}/admin/countAllTeams`, { headers: headers });
  }

  /**
   * Get heroes count
   * @returns count
   */
  public countAllHeroes(): Observable<any> {
    let headers = new HttpHeaders().set('Content-Type', 'application/json');
    let tokenAuth = (localStorage.getItem('token'));
    headers = headers.set("Authorization", `${tokenAuth}`);
    return this.http.get<any>(`${this.baseUrl}/admin/countAllHeroes`, { headers: headers });
  }
}
```