# Aktörer

**Lag ledare**

De kan registrera nya lag med medlemmar samt registrera vilka grenar, ålder och kön de tillhör. De kan även ta bort lag som de har registrerat.

**Sekreterare**

Handhåller Tävlingspoängen och räkna ut medelpoängen.

Sekreteraren kan skriva in och ändra poängen i systemet.

**Domare**

Domaren kan logga in och ändra i schemat, exempelvis vilka domare som ska döma vilka lag samt tider.

# Flöden

**Användningsfalls beskrivningar**

Användaren fyller i samtliga fält och klickar på registrera ny användare.

**Previllkor**:

* Användaren har gått in på webbsidan och hittat registrerings länken.

**Postvillkor**:

* Registrering av användare är lyckad.
* Alla data som är ifyllt har skickats till databasen(filen).

**Primära**:

* Bekräftelse på en lyckad registrering (Den perfekta världen)

**Alternativa flöden:**

* Fält som inte är korrekt ifyllt kastas ett undantag.
* Fält som är tomma kastas ett undantag.

# Klasspecifikationer

**Klass UserRegister**

Klassen är avsedd för aktörer som registrera sig i systemet innehållande fält, metoder, egenskaper och konstruktorer.

**Fält**

**string \_fullname;**

Innehåller för och efternamn på användaren.

**string \_password;**

Innhåller användarens lösenord.

**string \_password2;**

Innhåller användarens upprepade lösenord.

**string \_email;**

Innehåller användarens email.

**string \_actor;**

Innehåller användarens roll.

**string \_pin;**

Innehåller användarens personnummer.

**Egenskaper**

**String Fullname**

Ger fältet \_username ett värde av typen string samt ett krav på att namnet ska innehålla mellan 4-25 tecken.

**String Password**

Ger fältet \_password ett värde av typen string samt går igenom metoden PasswordIsValid ().

**String Password2**

Ger fältet \_password2 ett värde av typen string samt kollar så att värdet stämmer överens med Password.

**String Email**

Ger fältet \_email ett värde av typen string samt går igenom metoden

EmailIsValid ().

**String Actor**

Ger fältet \_actor ett värde av tre möjliga.

**String pin**

Ger fältet \_pin ett värde av typen string samt går igenom metoden

PinIsValid ().

**Konstruktorer**

**Register()**

Skapa ett nytt objekt av klassen Register.

Inparametrar är samtliga fält.

**Metoder**

**PasswordIsValid ()**

Kollar så att \_password innehåller mellan 6 till 25 tecken samt inkludera minst en siffra genom reguljära uttryck.

**EmailIsValid ()**

Kollar om \_email är en valid mail genom reguljära uttryck.

**PinIsValid()**

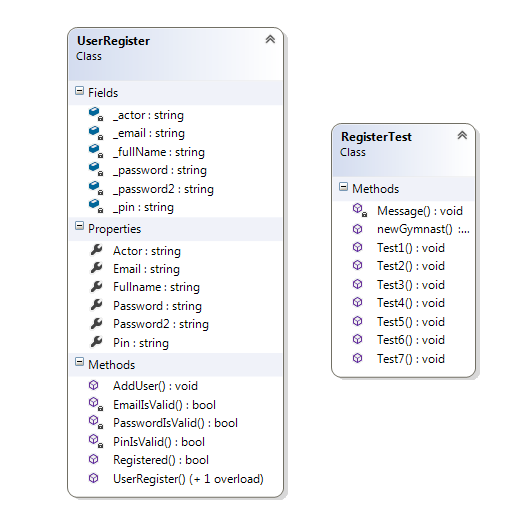
Kollar om \_pin är ett giltigt personnummer genom reguljära uttryck.

**AddUser()**

Lägger in samtlig information till en text fil

**Registered()**

Returnera ett boleanskt värd för att se om registreringen har lyckats.



**Testdokumentation**

Testningen går ut på att säkerställa att alla delar i klassen fungera som planerat och uppnår dokumenterade krav. Testningen kommer även se till att minimera eventuella buggar.

### Testsvit 1 – Förväntade resultat av klassen UserRegister

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nr** | **Typ av test** | **Testdata** | **Metod** | **Kommentar** | **Förväntat Resultat** |
| 1 | Kontroll av registrering | Namn = ”roye”  Lösenord = ”haxx0r”  Email =”roy@lnu.se”  Aktör=” lAg LeDaRE”  Pin="871131-2739" | Skapa en instans av klassen UserRegister och skapa en ny konstruktor som innehåller samtliga parametrar. |  | Ny Medlem registreras. |
| 2 | Kontroll av namn | Namn = ”roy” | Fullname() | Kräver minst 4  tecken | Inte godkänt |
| 3 | Kontroll av lösenord | Lösenord= ”haxxor” | Password() | Kräver en siffra minst | Inte godkänt |
| 4 | Kontroll av Email | Email=  ”roy.87@msn.com” | Email() | Både 2 och 3 bokstäver efter sista punkt ska godkännas | Email skapas |
| 5 | Kontroll av aktör | Aktör=”domare” | Actor() | Val av aktör är inte beroende av att vara gemener eller versaler | Aktör skapas |
| 6 | Kontroll av personnummer | Pin="871331-2739" | Pin() | Värdet överstrider antal månader ”13” och kommer inte att godkännas | Inte godkänt |
| 7 | Kontroll av upprepat lösenord | Lösenord=”password” | Password2() | Värdet har inte samma värde som password() och därför falera det | Inte godkänt |

### Testsvit 1 – Resultat av klassen UserRegister

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nr** | **Typ av test** | **Testdata** | **Metod** | **Förväntat Resultat** | **Verkliga**  **Resultat** |
| 1 | Kontroll av registrering | Namn = ”roye”  Lösenord = ”haxx0r”  Email =”roy@lnu.se”  Aktör=” lAg LeDaRE”  Pin="871131-2739" | Skapa en instans av klassen UserRegister och skapa en ny konstruktor som innehåller samtliga parametrar. | Ny Medlem registreras. | Godkänt |
| 2 | Kontroll av namn | Namn = ”roy” | Fullname() | Inte godkänt | Inte godkänt |
| 3 | Kontroll av lösenord | Lösenord= ”haxxor” | Password() | Inte godkänt | Inte godkänt |
| 4 | Kontroll av Email | Email=  ”roy.87@msn.com” | Email() | Email skapas | Godkänt |
| 5 | Kontroll av aktör | Aktör=”domare” | Actor() | Aktör skapas | Godkänt |
| 6 | Kontroll av personnummer | Pin="871331-2739" | Pin() | Inte godkänt | Inte godkänt |
| 7 | Kontroll av upprepat lösenord | Lösenord=”password” | Password2() | Inte godkänt | Inte godkänt |

**Sammanfattning av tester.**

Klassen är uppdelad i sviter och de testas främst genom deras metoder för att snabbt och enkelt kunna testa deras funktionalitet.

Testerna körs och returnera om testet är

godkänd eller inte godkänt genom en try- catch sats.

Samtliga tester är inkapslade i en metod som innehåller en try- catch sats för att utgöra om testet blir godkänt eller inte och där efter hämtar en metod som färgkodar testerna genom boolenskt värde sant eller falskt.

Testerna ger en bra visuell bild av samtliga tester i konsol fönstret