# Interfacce Sottosistemi

(in ordine alfabetico dei loro nomi)

**Aggiornato sottosistema Map.**
**Altre variazioni evidenziate in colore giallo. Da rimuovere per accettazione.**

Ogni gruppo è responsabile per la verifica che gli altri sottosistemi forniscano al proprio sottosistema tipi ed operazioni secondo i nomi e la lista di parametri convenuta. In caso di sistematici disallineamento da parte di qualche sottosistema, mi si riferisca ma non oltre o 15 gg dalla prima seduta d'esami.

Per modellare molteplicità di oggetti, si impieghi l'interfaccia List. In caso di diversa convenzione con altro sottosistema, darmene informazione.

È sempre disponibile la posizione di responsabile di System Integration & System Architect.

# Announcing

### Gestione

Nulla

### Offering

GC SVILUPPO DI LIVELLO SCADENTE

GC 16 gennaio 2019 TUTTORA OBSOLETO

Adattare nome dei metodi

Aggiunggere Announce getAnnounce() //Vedere Rules

boolean pubblic annuncioLocatore ( PhysicalApartment pA , UserId uId)

/* Crea un nuovo annuncio per un locatore (verrà validato da Rules con il loro metodo check) e restituisce un boolean di conferma o errore */

- ➔ pubblic renterAnnouce ( PhysicalApartment pA , Nickname uId)

ListaAnnunciLocatore visualizzaStoricoLocatore ( UserId uId)

// Restituisce una lista di annunci relativa al proprio storico

ListaAnnunciLocatore visualizzaAnnunciAttiviLocatore ( UserId uId)

//Restituisce una lista dei propri annunci ancora attivi (non prenotati)

ListaAnnunciLocatore visualizzaBachecaLocatori ( UserId uId)

//Restituisce una lista con tutti gli annunci dei locatori

boolean verificaDisponibilità (AnnuncioLocatore aL , Data d1 , Data d2)

/* Restituisce un boolean se l'annuncio riporta un appartamento disponibile o meno nell'intervallo di tempo tra le due date indicate */

//Piuttosto che Data, impiegherei una classe diversa.

void aggiornaDisponibilità (AnnuncioLocatore aL , Data d1 , Data d2)

// Modifica il calendario di un appartamento all'interno di un annuncio se l'affitto è andato a buon fine

// Piuttosto che Data, impiegherei una classe diversa.

**Interfaccia di contesto del sottosistema**

boolean verificaAnnuncio (AnnuncioLocatore aL )

/* Verifica che l'annuncio in questione rispetti le regole per essere creato importato dal sottosistema Rules. */

UserId getUserId ( Context c)

// Restituisce l'identificativo di un utente importato dal sottosistema UserProfile

void validaAnnuncioLocatoreSegnalato (AnnuncioLocatore aL)

/* Verifica la validità dell'annuncio segnalato almeno x volte (contatore definito da rules) importato dal sottosistema Rules */

Già fatto da Rules?|

float getMediaValutazione ( UserId uId)

// Restituisce la media di tutte le valutazioni di un utente importato dal sottosistema Evaluation

float getMediaValutazione ( PhysicalApartmentId pApt)

/* Restituisce la media di tutte le valutazioni di un appartamento importato dal sottosistema Evaluation */

void iniziaPrenotazione ( AnnuncioLocatore aL , Data d1 , Data d2 , UserId uId)

// Inizia la procedura di prenotazione dell'appartamento importato dal sottosistema Renting

## Search

Nulla

# ApartmentGrouping

GC SVILUPPO DI LIVELLO SCADENTE

viewApartGroups(ApartmentId: aId): List <Group>

// L'operazione tramite l'ID dell'appartamento restituisce la lista dei gruppi a cui lo stesso appartiene.

/*

Manca Group

Apartment si riferisce ad apt, di base o a quello attrezzato?

Possibili sottodomini di interesse: Announcing

viewUserGroups(Nickname: nickname): List <Group>

// L'operazione tramite l'ID dell'utente restituisce la lista di tutti i gruppi di cui i propri appartamenti fanno parte

INTERFACCIA DA IMPORTARE:

Renting Management

/* GC

Tramite questa si arriva all'appartamento fisico?

*/

getApartments(

Nickname: nickname): List<Apartment>

// Si richiede tramite ID di utente la lista degli appartamenti potenzialmente affittabili dell'utente.

/* GC Non mi è chiaro */

Evaluation

getAvgScore(subjectEval:Nickname): double **throws**

// Si richiede al sottosistema Evaluation management la valutazione

dell'appartamento da suo idApt.

/* Sembra essere un utente più che un appartamento */

/* GC

Manca la definizione della classe Group e di tutte le sue operazioni, comprese quelle che definiscono un nuovo gruppo, inseriscono/rimuovono un apt da un gruppo.

*/

# Evaluation

## Interfaccia esposta dal sottosistema Evaluation

- getPendingReviews():List<Evaluation>
  - Restituisce tutte le recensioni da far verificare da un amministratore; si pensa di implementare List<Evaluation> come un Vector<Evaluation>;
  - Sottosistemi interessati:People (Administration)
- getEvaluations(id:Nickname): List<Evaluation>
  - Restituisce le valutazioni riguardanti una specifica entità (SubjectId può essere il Nickname di un Locatore, Locatario o l'ApartmentID di un Appartamento)
  - Sottosistemi interessati: Announcing (Search, Offering), Filters, People (User Profile)
- getEvaluation(eval:EvalId): Evaluation
  - Restituisce un oggetto valutazione , con tutti i suoi dati, dato l'Id della stessa
  - Sottosistemi interessati: Announcing (Search, Offering), People (UserProfile, Administration)
- getAvgScore(subjectEvalId: *SubjectEvalId*): double
  - Restituisce il voto, un numero decimale compreso tra [1, 10], dato dalla media dei voti di tutte le recensioni su una certa entità subjectEvalId (*SubjectEval*è un'astrazione che può essere il Nickname di un Locatore o Locatario o l'ApartmentID di un Appartamento )
  - Sottosistemi interessati: Announcing (Search, Offering)
- reportOffensiveUser(offeniveUser: Nickname): void
  - Aggiunge un offensiveUser, che ha scritto una recensione offensiva, agli utenti segnalati.
  - Sottosistemi interessati: Administration, Rules
- createEval(subjectEvalId: *SubjectEvalId*): Evaluation
  - Crea e restituisce una nuova recensione sul dato "*subjectEvalId*".
  - Sottosistemi interessati: People (UserProfile), Renting Management (?)
- List <Nickname> getBadUsers()
  - Riporta la lista di uteti con più di <N> recensioni negative.
  - Sottosistemi interessati: Rules

## Interfaccia esposta dal Sotto-sottosistema ViewEvaluation

*Sottosistema che comprende la grafica per visualizzare le recensioni*

- viewEvaluations(subject: *SubjectId*): Evaluations
  - Restituisce ed elenca a schermo le valutazioni di un dato utente o appartamento.
  - Sottosistemiinteressati: Announcing (Search, Offering) , Roles Management, Admin
- viewAnEvaluation(eval:EvalId)
  - Mostra a schermo i dettagli di una singola valutazione, tipicamente scelta tra quelle mostrate a seguito di viewEvaluations.
  - Sottosistemi interessati: Announcing (Search, Offering) , Roles, Administration

# Interfacce da Importare

*Note: Sono tradotti in inglese alcuni nomi per coerenza con le altre interfacce; inoltre sono qui riportati riferimenti a ciò che ci venne consegnato e/o allo stato consolidato nelle interfacce avute in dote sulla ML.*

## UserProfile Management

*Aggiornato con le firme di UserProfile*

- getUserInfo(nickname:Nickname, UserInfoType type): UserInfo
    - Restituisce le informazioni dell'utente, dato il nickname

## Notification Interface

- sendNotification(dest:Nickname)
    - Invia una notifica a un utente

## Renting Interface

*Aggiornato con le firme di Renting*

- getTerminatedContracts(userNickname:Nickname, UserTypeuserType):List<Contract>
    - Ricerca e restituisce la lista dei contratti associati all'utente attualmente terminati

## Rules Interface

*Aggiornato con le firme di Rules Management*

- checkEvaluation(eval:EvalId): boolean

  Controlla che la recensione segua le regole di Rules Management.Esegue un controllo sul testo (es. di una recensione) per verificare la presenza di parole offensive(in caso affermativo, restituisce true)

## Roles Interface

*Aggiornato con le firme di Roles & Status Management*

- getUserRoles(in nickname:Nickname): List<UserRole>
    - Restituisce i ruoli dell'utente (es. Locatore, Locatario, Amministratore, RegisteredUser) e.g., {RENTER, TENANT, ADMINISTRATOR, REGISTERED_USER, GUEST}
- isTenant(nickname:Nickname):boolean
    - Permette di capire se un utente possiede il ruolo di Locatario
- isRenter(nickname:Nickname):boolean
    - Permette di capire se un utente possiede il ruolo di Locatore

## Filters Interface

*Aggiornato con firme di Filters versione Rel02*

- filterApartmentEvaluations (aEs: List<ApartmentEvaluation>, selectedFilters: List<Filter>): List<ApartmentEvaluation>

- sortApartmentEvaluations (aEs: List<ApartmentEvaluation>, sC: SortingCriterium): List<ApartmentEvaluation>
    - Restituiscono le recensioni di appartamenti filtrate o ordinate secondo certi criteri.
- filterUserEvaluations(uEs: List<UserEvaluation>, selectedFilters:List<Filter>): List<UserEvaluation>
- sortUserEvaluations(uEs: List<UserEvaluation>, sC:SortingCriterium): List<UserEvaluation>
    - Restituiscono recensioni di utenti  filtrate o ordinate secondo certi criteri.

## PhysicalApartment Management

*Aggiornato con firme di Physical Apartment*

/* GC Perché ci si rivolge e si richiede l'appartamento basico (a PhysicalApartment) e non quello attrezzato (a Renting)?

*/

- getApartment(aId:ApartmentId): Apartment
    - Restituisce un appartamento, datone l'id.
- getApartmentInfo(aId:ApartmentId): ApartmentInfo
    - Restituisce le informazioni su un appartamento, datone l'id. Sono da restituire almeno l'indirizzo e la zona.
- getApartmentSurfaceAddress(aId:ApartmentId): ApartmentSurfaceAddress

Restituisce l'indirizzo dell'appartamento dato l'id

Note:

- Ricerca recensione comprende la visualizzazione delle recensioni
- La ricerca di una lista di recensioni specifica viene effettuata da Filters

/* GC

## Tipi che questa interfaccia dovrebbe rendere disponibili

1. Evaluation
2. EvalId
3. *SubjectEvalId* Al momento, sembra che solo Evaluation usi ed esporti tale astrazione; quindi dovrebbe essere implementata dallo stesso sottosistema, utilizzando lea classe Nickname esposta da People e *Apartment* , i.e., [Basic] Apartment(o EquippedApartment?)
4. *SubjectEval* Iderm c.s., i.e.: al momento, sembra che solo Evaluation usi ed esporti tale astrazione; quindi dovrebbe essere implementata dallo stesso sottosistema, utilizzando le classi Tenant, Renter, Administrator, RegisteredUser esposte da People e *Apartment* ,  i.e., [Basic] Apartment (o EquippedApartment?)

   EquippedApartment dovrebbe eventualmente estendere Icons aggiungendo nuove specifiche icone onde invocare Map con queste.

*/

# Filters

## Interfaccia esposta dal sottosistema Filters

0. `List <ApartmentAnnouncement> filterApartmentAnnouncements (List<ApartmentAnnouncement> aAs, List<Filter> selectedFilters)`
   Il sistema mostra una lista di annunci di appartamenti filtrata secondo i filtri selezionati.
1. `List <ApartmentAnnouncement> sortApartmentAnnouncements (List<ApartmentAnnouncement> aAs, SortingCriterium sC)`
   Il sistema mostra una lista di annunci di appartamenti ordinata in base al criterio di ordinamento scelto.
2. `List <UserAnnouncement> filterUserAnnouncements (List <UserAnnouncement> uAs, List<Filter> selectedFilters)`
   Il sistema mostra una lista di annunci scritti da utenti filtrata secondo i filtri selezionati.
3. `List <UserAnnouncement> ordinaAnnunciUtente (List<UserAnnouncement> uAs, SortingCriterium sC)`
   Il sistema mostra una lista di annunci scritti da utenti ordinata in base al criterio di ordinamento scelto.
4. `List <ApartmentEvaluation> filterpartmentEvaluations (List <ApartmentEvaluation> , List<Filter> selectedFilters)`
   Il sistema mostra una lista di valutazioni dell'appartamento selezionato filtrata secondo i filtri selezionati.
5. `List <ApartmentEvaluation> sortpartmentEvaluations (List <ApartmentEvaluation> aEs, SortingCriterium sC)`
   Il sistema mostra una lista di valutazioni dell'appartamento selezionato ordinata in base al criterio di ordinamento scelto.
6. `List <UserEvaluation> filterUserEvaluations (List <UserEvaluation> uEs, List <Filter> selectedFilters)`
   Il sistema mostra una lista di valutazioni riguardo l'utente selezionato filtrata secondo i filtri selezionati.
7. `List <UserEvaluation> sortUserEvaluations (List <UserEvaluation> uEs, SortingCriterium sC)`
   Il sistema mostra una lista di valutazioni sull'utente selezionato ordinata in base al criterio di ordinamento scelto.

## Operazioni/Dati richiesti ad altri sotto-sistemi:

0. Sotto-sistema "Announcing"
   a. Classe `ApartmentAnnouncement`
   b. Classe `UserAnnouncement`
1. Sotto-sistema "Evaluation"
   a. Classe `ApartmentEvaluation`
   b. Classe `UserEvaluation`

e implementazione di relative List.

# Maps

## INTERFACCIA ESPOSTA

•**geocodeAddress**( SurfaceAddress surfAdd ): LatLong
/* L'operazione converte l'indirizzo di superficie ricevuto come parametro nelle relative coordinate geografiche; */
•Sottosistemi interessati: Physical Apartment mgt;

// *Icon* è una astrazione specializzabile variamente (e.g., tipo, colore, con stelle).
Sottosistemi interessati: Announcing, Renting;
•**insertIcon**(Icon i): void
// L'operazione consente l'inserimento di una determinata icona in persistenza;
•**removeIcon**(Icon i): void
// L'operazione effettua la rimozione di una determinata icona;
•**showIcon**(~~LatLong point,~~ Icon i): MapView
/* L'operazione mostra sulla mappa l'icona i (e.g., appartamento). In relazione all'icona passata, è possibile mostrare varie informazioni; e.g., il tipo di risorsa (appartamento), la sua qualità o valutazione, stato di affitto. */
•~~showIcon(SurfaceAddress surfAdd, Icon i): MapView~~
•// L'operazione mostra sulla mappa all'indicato surfAdd la icona i;

•**showIcons**( List <Icon> iconL~~List<IconedLatLong~~> ~~iconedPointList~~): MapView
// L'operazione mostra sulla mappa le icone posizionate alle loro coordinate.

~~•showIcons( List< IconedSurfaceAddress > iconedSurfaceAddresses): MapView~~
~~/* L'operazione mostra sulla mappa le icone ai surface address corrispondenti passati come parametro */~~
•**highlightIconsByZone**( LatLong center, double radius, IconType ~~String~~ iconType): MapView
// L'operazione evidenzia su mappa tutte le icone del tipo iconType presenti nella zona di ricerca definita.

TIPI DI DATI ESPORTATI:
•**LatLong**: classe che identifica un punto sulla mappa, possiede come attributi:
// double latitudine;
// double longitudine;
•**MapView**: classe che identifica la mappa da visualizzare, possiede come attributi:
// LatLong center;
•int zoom;
•List<Icon> iconList

•//vettore che identifica i punti contrassegnati sulla mappa;

•**Icon**: classe che identifica un'icona utilizzata per essere posizionata su mappa, possiede come attributi: può possedere attributi, quali:

- •LatLOng latLong // proprie coordinate
- •IconID iconID; //identifica l'icona
- •IconType iconType; //rappresenta il tipo di risorsa (appartamento, camera, …)
- •Status status; //libero, occupato, ..
- •Evaluation evalutation; //numero di stelle o un colore
- •Image img; //immagine assegnata all'icona
- • Object o // l'oggetto che rappresenta, e.g., un AppartamentId.

~~•**IconedSurfaceAddress**: classe che associa ad un'icona la sua posizione su mappa dettata dall'indirizzo presente nell'oggetto di tipo SurfaceAddress, possiede come attributi~~
~~// Icon i~~
~~// SurfaceAddress surfAddr~~

~~IconedLatLong: classe che mantiene l'assocazione tra un'icona rappresentante una risorsa e la sua posizione in base alle coordinate, avrà come attributi:~~
~~Icon icon~~
~~LatLong point~~

# People

// Chi gestisce i Guest (utente non registrato)? Chi realizza i suoi casi d'suo?

Include i seguenti tre sottosistemi

# Administration

INCOMPLETO. INSUFFICIENTE

Il sottosistema **Administration mgt** svolge le mansioni di amministrazione del sistema, che sono la raccolta di report da parte degli altri sottosistemi e l'esecuzione di operazioni di eliminazione e modifica di informazioni non idonee o errate. Presenta la seguente interfaccia:

| | |
|---|---|
| Segnatura | boolean report() <overloaded><br>report(nickname Nickname, nicknameVictim Nickname, description Description)<br>report(nickname Nickname, ApartmentId apartamentId, description Description)<br>report(nickname Nickname, AnnounceId announceId, description Description)<br>report(nickname Nickname, TextId textId, Description description)<br>//GC???TextId ??<br>report(Nickname nickname, Description description)<br>// Di che tipo appartamento si tratta? |
| Descrizione | Invia un report (segnalazione descrittiva) su un'entità del sistema |
| Ritorno | True se il report è stato inviato correttamente// GC Inserire eccezioni<br>False altrimenti |
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | void signal() <overloaded><br>signal(Nickname nickname, SignalOpU op)<br>signal(ApartementId ApartementId, SignalOpAp op) |

| | |
|---|---|
| | signal(AnnounceId announceId, SignalOpAn op)<br>signal(TextId textId, SignalOpT op)<br>// Idem c.s. |
| Descrizione | Chiede l'esecuzione di un'operazione amministrativa su un'entità del sistema |
| Ritorno | Nessuno |
| Eccezioni | Solleva eccezioni in caso di errore |

Nota: le diverse enumerazioni SignalOpX servono ad indicare l'azione che si vuole compiere all'interno dell'operazione signal()

/* 20190116 Rules (e forse altri subsistemi) vanno informati della decisione presa in conseguenza della rihiesta di ban. In particolare, se si è rinunciato ("derigato") al ban (e.g., per qualche utente importante; se non lo si fa, loro continueranno a segnalare per ogni azione negativa fatta dallo stesso. D'altro canto, Rules dovrebbe avere anche un secondo livello di segnalazione; e.g., superBanRequest, quando un utente "importante" supera il 2° livello di ban).

// GC X??? Siamo ancora a questo?

//GC Salvo casi banali, non è bene usare un parametro per distinguere diverse azioni da svolgere in una operazione. In caso, usare operazioni diverse.

Note:

Le categorie di report e le operazioni di signal nei vari contesti sono da definire in accordo con le richieste degli altri sottosistemi.

// GC X??? Se non voi, chi le definisce?

Administration mgt richiede agli altri sottosistemi di mettere a disposizione metodi per cancellarne i diversicontenuti (appartamenti, annunci, valutazioni, utenti).

// GC Bisogna essere più specifici Quale sottosistema deve rendere disponibile quale operazione o tipo?

# Roles & Status

Il sottosistema **Roles& Status mgt** si occupa della gestione del ruolo e dello status degli utenti nel sistema. I ruoli all'interno del sistema sono:

- ~~**Guest**~~
- **RegisteredUser**: utente che non possiede né appartamenti, né contratti di affitto.
- **Renter (Locatore)**: proprietario di appartamenti. Può essere anche Locatario.

- **Tenant (Locatario** o Affittuario): può essere anche Locatore.

Lo **status** di un utente nel sistema è la composizione dei seguenti stati:

- **ACTIVE**: utente che ha utilizzato il sistema nel "tempo recente".
- **INACTIVE**: utente che non ha utilizzato il sistema nel tempo recente.
- **CANCELLED**: utente eliminato dal sistema di cui ancora si conservano i dati.
- **BANNED**: utente bandito, cioè che non può più accedere alla piattaforma.

Presenta la seguente interfaccia:

| | |
|---|---|
| Segnatura | Roles getRoles(Nickname nickname) |
| Descrizione | Permette di ottenere i ruoli svolti dall'utente nel sistema |
| Ritorno | Oggetto Roles contenente alcuni ruoli, ciascuno denotante un ruolo dell'utente |
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | Status getStatus(Nickname nickname) |
| Descrizione | Permette di ottenere lo status dell'utente nel sistema |
| Ritorno | Oggetto Status contenente la denotazione dello status dell'utente |
| Eccezioni | Nickname null |

| | |
|---|---|
| Segnatura | voidmakeARenter(Nickname nickname) |
| Descrizione | Rende l'utente <nickname> un locatore. |
| Ritorno | True se il cambio di ruolo ha avuto successo // GC Eccezioni<br>False altrimenti |
| Eccezioni | |

| | |
|---|---|
| Segnatura | void removeRentership(Nickname nickname) |
| Descrizione | Toglie all'utente <nickname> il ruolo di locatore se lo è. |
| Ritorno | True se il cambio di ruolo è effettivamente avvenuto e con successo<br>False altrimenti. // GC Meglio Eccezioni? |
| Precondizione | nickname è non null |
| Eccezioni | |

| | |
|---|---|
| Segnatura | boolean makeATenant(Nickname nickname) |
| Descrizione | Rende l'utente <nickname> un locatario. |
| Ritorno | True se il cambio di ruolo ha avuto successo<br>False altrimenti // GC Meglio Eccezioni? |
| Precondizione | nickname è non null |

| | |
|---|---|
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | boolean removeTenantship(Nickname nickname) |
| Descrizione | Toglie all'utente <nickname> il ruolo di locatario |
| Ritorno | True se il cambio di ruolo è effettivamente avvenuto e con successo<br>False altrimenti // GC Meglio Eccezioni? |
| Precondizione | nickname è non null |
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | boolean changeUserStatus(Nickname nickname, UserStatus status) |
| Descrizione | Cambia lo status corrente dell'utente <nickname> in status |
| Ritorno | True se il cambio di status ha avuto successo<br>False altrimenti // GC Meglio Eccezioni? |
| Precondizione | nickname è non null |
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | boolean isRenter(Nickname nickname) |
| Descrizione | Permette di capire se un utente possiede il ruolo di locatore |
| Ritorno | True se l'utente è locatore<br>False altrimenti |
| Precondizione | nickname è non null |
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | boolean isTenant(Nickname nickname) |
| Descrizione | Permette di capire se un utente possiede il ruolo di locatario |
| Ritorno | True se l'utente è locatario<br>False altrimenti |
| Precondizione | nickname è non null |
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | boolean isActive(Nickname nickname) |
| Descrizione | Permette di capire se un utente possiede lo status di attivo |
| Ritorno | True se l'utente è in stato ATTIVO<br>False altrimenti |
| Precondizione | nickname è non null |
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | boolean isInactive(Nickname nickname) |
| Descrizione | Permette di capire se un utente possiede lo status di inattivo |
| Ritorno | True se l'utente è in stato INATTIVO<br>False altrimenti |
| Precondizione | nickname è non null |
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | boolean isCanceled(Nickname nickname) |
| Descrizione | Permette di capire se un utente possiede lo status di cancellato |
| Ritorno | True se l'utente è in stato CANCELLATO<br>False altrimenti |
| Precondizione | nickname è non null |
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | boolean isBanned(Nickname nickname) |
| Descrizione | Permette di capire se un utente possiede lo status di bannato |
| Ritorno | True se l'utente è in stato BANNATO<br>False altrimenti |
| Precondizione | nickname è non null |
| Eccezioni | Nessuna |

# UserProfile&Services

Il sottosistema **UserProfile&Services mgt** si occupa della gestione delle informazioni degli utenti nel sistema, nonché della loro creazione ed eliminazione. Presenta la seguente interfaccia:

| | |
|---|---|
| Segnatura | BasicUserInfo getBasicUserInfo (Nickname nickname) |
| Descrizione | Permette di ottenere i dati base dell'utente |
| Ritorno | Oggetto BasicUserInfo contenente le informazioni dell'utente |
| Eccezioni | Nessuna //GC nichname o type null |

| | |
|---|---|
| Segnatura | RestrictedUserInfo getRestrictedUserInfo (Nickname nickname) |
| Descrizione | Permette di ottenere i dati sensibili dell'utente |
| Ritorno | Oggetto RestrictedUserInfo contenente le informazioni dell'utente |
| Eccezioni | Nessuna //GC nichname o type null |

| | |
|---|---|
| Segnatura | boolean doesNicknameExist(Nickname nickname) |
| Descrizione | Verifica se il nickname è già in uso nel sistema |
| Ritorno | True se il nickname è già utilizzato da un utente nel sistema |
| | False altrimenti |
| Eccezioni | Nessuna //GC nichname null |

| | |
|---|---|
| Segnatura | boolean doesTaxCodeExist(TaxCode cf) |
| Descrizione | Verifica se il codice fiscale cf è già in uso nel sistema |
| Ritorno | True se il codice fiscale è già utilizzato da un utente nel sistema |
| | False altrimenti |
| Eccezioni | Nessuna |

| | |
|---|---|
| Segnatura | void createUser(Nickname nickname, UserInfo userInformation) |
| Descrizione | Inserisce un nuovo utente nel sistema //GC Definire la classe UserInfo |
| Ritorno | Nessuno |
| Eccezioni | Solleva eccezioni in caso di errore |

| | |
|---|---|
| Segnatura | void cancelUser(Nickname nickname) |
| Descrizione | Cancella logicamente un utente dal sistema. I dati utente verranno comunque conservati |
| Ritorno | Nessuno |
| Eccezioni | Solleva eccezioni in caso di errore |

| | |
|---|---|
| Segnatura | void deleteUser(Nickname nickname) throws … |
| Descrizione | Elimina fisicamente un utente dal sistema. I dati utente sono definitivamente rimossi dal sistema applicativo |
| Ritorno | Nessuno |
| Eccezioni | Solleva eccezioni in caso nickname è nullo o non debba essere eliminato dal sistema |

public class UserInfo

La classe UserInfo contiene una copia delle informazioni dell'utente. Potrà essere specializzata in diverse sottoclassi, come BasicUserInfo, RestrictedUserInfo.

## public class UserInfoType

Da realizzare

# PhysicalApartment (o BasicApartment?)

Valutare se sia il caso di definire classi abstract *Apartmente* *ApartmentId* per tutti da cui derivare BasicApartment e BasicApartmentId, rispettivamente, per questo sottosistema e poi da esse EquippedApartment ed EquippedApartmentId in Renting.

**Elenco delle Funzionalità da esportare:**

public ApartmentId **insertApartment(**SurfaceAddress, Plant, Images, Description, Renter, LatLog**) throws** NewApartmentInclusionException
//SurfaceAddress include tutte le informazioni compreso scala, interno e piano.
// Images può essere null

public ApartmentId **inserApartment(**SurfaceAddress, Plant, Images, Description, Renter**) throws** NewApartmentInclusionException
//Uses Map to get coordinates

**public void** showApartmentAsIcon (ApartmentIda, Map m, *Icon* i)
//GC Mostra a con la sua LatLog su m come i)
//GC altre invocazioni di Map
//GC è Apt che invoca Map, non l'incontrario.

/* GC
Converrebbe definire la classe **Apartment**.

Solo se tale classe fosse utile anche ad altri sottosistemi, allora la si potrebbe inserire in un package di base, accessibile da altri oltre che da PhysicalApt.
La classe dovrebbe essere comprensiva di attributi SurfaceAddress, Plant, Images, (Short &Full)Description, UserId, UserStatus, LatLog).

ApartmentStatus: mi vengono in  mente AVAILABLE, RENTED, IN_MANTAINANCE.

Ogni istanza  di Apartment andrebbe poi inserita da PhysicalApt nel sistema in modo da poterla gestire. Per inserire un apt., l'utentenickname dovrebbe avere il ruolo di Renter: in generale, per altre operazioni, potrebbe essere quello di Owner, di un procuratore di vendita, un agente ….

Creazione:

Apartment a = **new** Apartment (SurfaceAddress s, Plant p, Images is, Description d, UserId renter, ApartmentStatus aS [, LatLog cs]);
//GC Il costruttore genera anche un Id per l'appartamento.

ApartmentId aId = a.getId();

//GC Inserimento
**try** …
insertApartment**(**aId**);**
**catch**(…);
*/

**Import** SurfaceAddress, Plant, Images, Description, UserId, ApartmentStatus, LatLog, Room, Service, ApartmentId;

**class Apartment {**
private SurfaceAddress s;
private Plant p;
private Images is;
private Description d;
private UserId renter;
private ApartmentStatus aS;
private LatLog cs;
private ApartmentId aId;
private List <Service> services
private List <Room> rooms

Room r
// GC …

Apartment (SurfaceAddress s, Plant p, List <Room> rooms, List <Service> services, Images is, Description d, UserId renter, private ApartmentStatus aS, LatLog cs);
/*GC La lista services potrebbe essere dedotta automaticamente da rooms e quindi non inclusa fra i parametri */
Apartment (SurfaceAddress s, Plant p, List <Room> rooms, List <Service> services, Images is, Description d, UserId renter, ApartmentStatus aS);

void**modify (**SurfaceAddress s**) throws** ModifyApartmentException

void **modify (**LatLog c**) throws** ModifyApartmentException

void**modify (**Plant p**) throws** ModifyApartmentException
void**modify (**Images i**) throws** ModifyApartmentException
void**modify (**Description d**) throws** ModifyApartmentException

ApartmentId **getId** (Apartment aId);
Description **getDescription** ()

- Restituisce oggetto di classe Description che contiene le informazioni, comprensive di ShortDescription e Full Description, sull'appartamento.

ApartmentSurfaceAddress**getSurfaceAddress**():

- Restituisce l'indirizzo dell'appartamento

**getRooms** (): List <Room>

- Restituisce elenco di ambienti costituenti l'appartamento

**getServices** (): List <Service>

- Restituisce elenco dei servizi per l'appartamento

**getLatLog** (): LatLog

- Restituisce asttributo cs LatLog (latitudine e longitudine)

// GC …
**}**

**//GC Sottosistema PhysicalApt**
**Import** Apartment

void **modifyApartment(**ApartmentId, SurfaceAddress**) throws** ModifyApartmentException

void **modifyApartment(**ApartmentId, LatLog**) throws** ModifyApartmentException

void **modifyApartment(**ApartmentId, Plant**) throws** ModifyApartmentException
void **modifyApartment(**ApartmentId, Images**) throws** ModifyApartmentException
void **modifyApartment(**ApartmentId, Description**) throws** ModifyApartmentException

void **removeApartment(**ApartmentId**) throws** RemoveApartmentException

void insertApartment**(**ApartnebtId**) throws** InsertApartmentException**;**

**getApartment**( ApartmentId i): Apartment

- Restituisce oggetto di classe Apartment con tutte le info dell'appartamento, ~~foto~~ e servizi collegate, dato l'id
- Sottosistemi interessati: Announce Management

/*GC La pianta deve essere parte dell'appartamento. Direi lo stesso per le foto.
*/

List <ApartmentDescriptor>**getApartmentsByRenter**(UserId renter) **throws** UserTypeException;

- Restituisce elenco di brevi descrizioni degli appartamenti gestiti, dato id del gestore; **`ApartmentDescriptor`** dovrebbe includere **`ApartmentId`** e uno **`ShortDescriptor`**, e.g., un breve testo, sull'appartamento, magari un sommario di **`Description`**. Se di interesse, si chiamerà poi **getApartment**( ApartmentId i);
- Sottosistemi interessati: Admin, Annoumcing ~~Filters~~

**getApartmentSurfaceAddress**(ApartmentId i): ApartmentSurfaceAddress

- Restituisce l'indirizzo dell'appartamento dato l'id
- Sottosistemi interessati: Map Management

**getServices**(): List <Service>

- Restituisce elenco dei servizi generali previsti dal sistema
- Sottosistemi interessati: Renting Management, Filters, Announce Management, Admin

/* GC
Tipi possibili di servizi di un appartamento fisico sono ad esempio i seguenti:
{Bagno (Doccia SI|NO, Vasca SI|NO, …), Gas (DiCittà | Bombolone | Bombola), Cucina (Macchina del Gas SI|NO, Frigorifero SI |NO, Lavastoviglie SI | NO …), Riscaldamento (Gas | Elettrico | Altro) (MisuratoreConsunoSingolaUnità SI | NO), Condizionamento (MisuratoreConsunoSingolaUnità SI | NO) , Termostato, Timer, …};
Per applicazioni in altri domini, e.g., Catasto, tali serbizi dovrebbero essere nullo per default.

In effetti, mi pare:
1) che serva una classe Servizio per ottenere flessibilità / manutenibilità / portabilità.
2) L'insieme dei servizi è, per l'applicazione, statico o dinamico? Se statico basta definirlo in una semplice classe: questa dovrà essere però ricompilata ad ogni cambiamento dell'insieme. Altrimenti, se è dinamico, esso dovrà essere gestito a tempo di esecuzione (con include, get, modify, remove dei suoi elementi): in tal caso, userei l'interfaccia List che è uno standard de facto dell'applicazione.
*/

**getServicesOfApartment**(ApartmentId i): List <Service>

- Restituisce elenco dei servizi per un appartamento
- Sottosistemi interessati: Renting Management, Filters, Announce Management

/* GC
List è l'interfaccia che, de facto, stiamo usando come standard in tutto il sistema per modellare una pluralità di oggetti. Poi ciascun sottosistema se la implementerà come preferisce.*/

**getDescriptionOfApartment**(ApartmentId i): Description

- Restituisce oggetto di classe Description che contiene le informazioni, comprensive di ShortDescription e Full Description, su un appartamento, dato l'id. Short description può essere anche essenziale: e.g., Loft, villetta, appartamento condominiale. Full Description può essere breve ma potrà essere esteso da RentingMgt tramite modify .
- Sottosistemi interessati: Renting Management (e AnnouncingMgt?)

/* GC
Pensate alla implementazione in questa fase è SBAGLATO? Non vincolatevu a una stringa fin da ora. Come riprova, è appena sorta la esigenza comporre Description come ShortDescription e FullDescription.
*/

/* GC

Announcing userà l'appartamento fisico, cioè con gli attributi e i metodi disponibili ; non possono inventarselo. Potranno modificare la vs. descrizione iniziale si limiterà al fisico, per quanto necessario.
*/

**getLatLogOfApartment**(ApartmentId i): LatLog

- Restituisce classe LatLog(latitudine e longitudine)

/* GC Dobbiamo immaginare che in Map si entri con un SurfaceAddress e non con un ApartmentId. Map è un sottosistema di base e perciò non dovrebbe dipendere dal concetto di appartamento (ma avere al più quello di icona di un appartamento e forse un relativo colore atto a esprimerne la valutazione effettuata da Evaluation); quindi, dovremmo evitare che Map chiami il sottosistema PhysicalApt allo scopo di trasformare un appartamento nell'indirizzo di superficie dello stesso e poi da questo indirizzo ricavare le coordinate. Dato che se un appartamento fisico esiste allora esso ha coordinate (vi deve essere navigabilità da Appartamento a Coordinata) allora manteniamo qui in PhysicalApt la operazione **getLatLogOfApartment**(ApartmentId i): LatLoge lasciamo a Map la operazione **getLatLog**(SurfaceAddress s): LatLog.
*/

/* GC
Per una definizione dinamica dei servizi utilizzare List, impiegando
la classe Service: List <Service>.
Ovviamente una implementazione dovrà realizzare tutte le operazioni previste dalla interfaccia List.
*/

/* GC
A livello di sottosistema, se si ritenesse operare su List troppo oneroso o rischioso, si potrebbe realizzare la classe Servicesda rrendere accessibile solo al ruolo Admin:

Import Service, ServiceId;
**void insertService**(Service service)**throws** ServiceException;
**void updateService**(Service newServiceVersion) **throws** ServiceException;
**void deleteService**(ServiceId) **throws** ServiceException;

- Prima di eliminare il sistema verifica che il servizio non è ancora impiegato da alcun appartamento (una funzione privata canDelete(ServiceId p)). Questa dovrebbe essere invocata dal chiamante. In tal caso contrario, si solleva una eccezione.
- Sottosistemi interessati: Admin

**void** cancelServiceFromApartment(ApartmentId i, ServiceId s) **throws** ServiceException;

- Rimuove il servizio passato come parametro dell'elenco dei servizi di un appartamento
- Sottosistemi interessati: Renting Management, Filters, Announce Management

** è il locatore che inserisce, modifica, i servizi forniti o non più forniti in un appartamento presente nel sistema e pertanto anche quest non sono operazioni esportabili ad altri sottosistemi.

/* GC In generale, se un'operazione resa disponibile da un sottosistema è utile anche o solo a un attore del sistema, allora l'attore stesso, tramite la classe View che lo accoppia allo specifico caso d'uso, invocherà il controllore del medesimo caso d'uso, il quale si limiterà a semplicemente invocare l'operazione esposta dal sottosistema. */

## Classi da definire

/*GC Dico classi, così se un domani cambierà la implementazione, non bisognerà cambiare le correlate classi applicative. */

Apartment

ApartmentId

ApartmentSurfaceAddress

RenterId (Classechepensodebbaessererealizzata a cura di RentMgt)

Service

ServiceId

Characteristics: List <Characteristic>;

/* GC

Characteristic serve per raggruppare (ApartmentGroupingMgt) appartamenti secondo caratteristiche comuni. Una Characteristic assume valore in un insieme evidentemente dinamico, ad esempio esempio: List <{Palazzo Gotico;  Palazzo Umbertino; Mobili Moderni; Materassi Nuovi}>. Esso deve essere gestito. Il gruppo PhysicalApartment può anche farlo nascere vuoto ma deve rendere disponibile le operazioni della lista affinché il sottosistema ApartmentGroupingMgt possa invocarle.

*/

Description

LatLog: (double, double)

- Altre eventuali

/*GC
Se partecipano alla interfaccia di PhysycalApartment allora i nomi vanno definiti ora; se non fatto da altri sottosistemi le classi vanno definite qui
*/

## Elenco delle Funzionalità da importare:

/*GC

MEMO. L'appartamento fisico è stato esplicitamente specificato in classe per essere riutilizzabile in altre eventuali applicazioni (e.g., Gestione Catasto, Pagamento tassa su case in base a loro caratteristiche fisiche (superficie totale, superficie bagni, superficie cucina, superfici di balconi e terrazze …).

*/

/* GC

Resta aperta una questione che verificherò. Esiste un modello di appartamento che include quello fisico ed è un appartamento da affittare come visto e gestito dal sistema di renting (e.g., AppartamentoDaAffittareOVendere, ApartmentToRentOrForSale)?

Di certo ci sono dei servizi che tipicamente non si indicano per catasto, compracendita o tasse, quali WiFi, TV et similia, numero di letti per camera,tipi di letti.

Alcune operzioni sono riservate a specifici ruoli. Quindi PhysicalApartment deve importare dal sottosistema UserMgt le classi e l'operazione di cui appresso:

UserRoles getUserRoles(UserId u): UserRoles

Restituisce i ruoli dell'utente u (es. RENTER, TENANT, ADMINISTRATOR …)

Questo degli user roles serve, peraltro, per sollevare eccezioni in caso di operazioni riservate. Bisognerebbe identificare al più presto chi può fare che cosa.

*/

# Renting

Sottosistemi con cui si interagisce: Announcing, UserProfile, Evaluating, PhysicalApartment.

RentingContract è una specializzazione di Contract.

1. List<RentingContract>getActiveContracts(Nickname renterNickname, UserTypeuserType): throws USER_ID_EXCEPTION, USER_TYPE_EXCEPTION
   INPUT: nickname dell'utente e tipologia di utente (locatore o locatario)
   COSA FA: Ricerca i contratti associati all'utente attualmente attivi
   OUTPUT: Lista dei contratti attivi
   ECCEZIONI: Sollevamento di una eccezione xxx se il nickname non esiste oppure yyy se non è associato ad un utente del tipo userType

2. List<RentingContract>getActiveContracts(EquippedApartmentId equippedApartmentId) throws EQ-APT_ID_EXCEPTION:
   INPUT: identificativo dell'appartamento
   COSA FA: Ricerca i contratti associati all'appartamento attualmente attivi
   OUTPUT: Lista dei contratti attivi
   ECCEZIONI: Sollevamento di una eccezione EQ_APT_ID_EXCEPTION se l'identificativo dell'appartamento non corrisponde ad alcun appartamento

3. List<RentingContract>getTerminatedContracts(Nickname userNickname, UserTypeuserType) throws USER_ID_EXCEPTION, USER_TYPE_EXCEPTION:
   INPUT: nickaname dell'utente e tipologia di utente (locatore o locatario)
   COSA FA: Ricerca i contratti associati all'utente attualmente terminati
   OUTPUT: Lista dei contratti terminati
   ECCEZIONI: Sollevamento di una eccezione USER_ID_EXCEPTION se il nickname non esiste oppure USER_TYPE_EXCEPTIONse non è associato ad un utente del tipo userType

4. List<RentingContract>getTerminatedContracts(EquippedApartmentId eApartmentId) throws EQ_APT_ID_EXCEPTION:
   INPUT: identificativo dell'appartamento
   COSA FA: Ricerca i contratti associati all'appartamento attualmente terminati
   OUTPUT: Lista dei contratti terminati

ECCEZIONI: Sollevamento di una eccezione APT_ID_EXCEPTION se l'identificativo dell'appartamento non ha un appartamento associato

5. booleanhasActiveContracts (Nickname userNickname) throws USER_ID_EXCEPTION
   INPUT: identificativo dell'utente
   COSA FA: Controlla la presenza di almeno un contratto attivo per l'utente (sia come locatario che come locatore)
   OUTPUT: TRUE or FALSE
   ECCEZIONI: Sollevamento di una eccezione USER_ID_EXCEPTIONse il nickname non esiste

6. boolean hasBeenHere(Nickname tenantNickname, EquippedApartmentId eApartmentId) throws USER_ID_EXCEPTION, USER_TYPE_EXCEPTION, EQ_APT_ID_ECEPTION:
   INPUT: nickname del locatario e identificativo dell'appartamento
   COSA FA: Controlla la presenza di un contratto tra il locatario e l'appartamento
   OUTPUT: TRUE or FALSE
   ECCEZIONI:
   - Sollevamento di una eccezione USER_ID_EXCEPTION se il nickname non esiste oppure USER_TYPE_EXCEPTION senon è associato ad un utente di tipo locatario
   - Sollevamento di una eccezione se l'identificativo dell'appartamento non ha un appartamento associatoAPT_ID_EXCEPTION
     /* GC PER LE ECCEZIONI CONTINUARE COME SOPRA. SE NON DEFINITE I NOME, GLI ALTRI NE USERANNO ALTRI E I SISTEMI NON SI INTEGRERANNO */
7. List<Nickname>getTenants(EquippedApartmentId eApartmentId):
   INPUT: identificativo dell'appartamento
   COSA FA: Ricerca tutti i locatari associati attualmente all'appartamento
   OUTPUT: Lista dei locatari nell'appartamento
   ECCEZIONI: Sollevamento di una eccezione se l'identificativo dell'appartamento non ha un appartamento associato

8. List<Nickname>getHousmates(Nickname tenantNickname, EquippedApartmentId eApartmentId)
   INPUT: nickname del locatario e identificativo dell'appartamento
   COSA FA: Ricerca gli attuali colocatari del loctario identificato da tenantNickname nell'appartamento identificato da apartmentId
   OUTPUT: Lista dei colocatari

ECCEZIONI:

- Sollevamento di una eccezione se il nickname non esiste oppure non è associato ad un utente di tipo locatario
- Sollevamento di una eccezione se l'identificativo dell'appartamento non ha un appartamento associato
- Sollevamento di una eccezione se il locatario non ha alcun ha alcun contratto attivo associato nell'appartamento identificato da apartmentId

9. List<Nickname>getTenantsHistory(EquippedApartmentId eApartmentId):
   INPUT: identificativo dell'appartamento
   COSA FA: Ricerca tutti i locatari attualmente o precedentemente associati all'appartamento
   OUTPUT: Lista dei locatari che hanno affittato l'appartamento
   ECCEZIONI: Sollevamento di una eccezione se l'identificativo dell'appartamento non ha un appartamento associato

10. List<EquippedApartmentInfo>getRentedApartments(Nickname renterNickname):
    INPUT: identificativo del locatore
    COSA FA: Ricerca di tutti gli appartamenti attualmente affittati
    OUTPUT: Lista degli appartamenti affittati
    ECCEZIONI: Sollevamento di una eccezione se il nickname non esiste oppure non è associato ad un utente di tipo locatore
    /*GC
    Piu' che appartamenti dovrebbe restituirne Info brevi, fatte ad esempio da EquippedApartmentId& ShortDocumentation, come già altrove definite (dove??); poi, se di interesse, si cercano dettagli utilizzando lo ID
    */

11. List<EquippedApartmentInfo>getUnRentedApartments(Nickname renterNickname):
    INPUT: identificativo del locatore
    COSA FA: Ricerca di tutti gli appartamenti attualmente non affittati (appartamento in cui tutte le stanze sono libere)
    OUTPUT: Lista degli appartamenti non affittati //GC Vedere nota sopra
    ECCEZIONI: Sollevamento di una eccezione se il nickname non esiste oppure non è associato ad un utente di tipo locatore

12. ContractgetContractData(RentedContractIdcontractId)
    INPUT: identificativo del contratto

COSA FA: Ricerca i dati di un contratto
OUTPUT: Contratto associato al contractId
ECCEZIONI: Sollevamento di un'eccezione se non vi è alcun contratto associato a contractId

/* GC Manca la Creazione e gestione delle classi EquippedApartment ed EquippedApartmentId. Questa utilizza la classe Apartment (o BasicApartment) come definita da PhysicalApartment. Dovete accordarvi con loro per definire cosa modellano loro e cos'altro modellate voi (di sicuro a voi spettano la aggiunta di wifi, telefono …)

*/

## Metodi da importare dagli altri sottosistemi

//Da modificare secondo Sue disposizioni; attendiamo che i sottosistemi da cui //importiamo le operazioni comunichino le dovute modiche

**/* GC Non io, ma voi e gli stakeholder di altri sottosistemi dovete raggiungere l'accordo */**

UserProfilemgt:

- getUserInfo(Nickname nickname, UserInfoType type): UserInfo

PhysicalApartment& Services mgt:

- getApartmentsByRenter(RenterId id): List <Apartment>

Evaluation Subsystem:

- getEvaluations(SubjectId id): Evaluations
- getAvgScore(SubjectId id): Double
- viewEvaluations(subjectId id): Evaluations
- viewAnEvaluations(EvalIdid)

# Rules

*Le regole di FERSA parametrizzate:*

*(Ogni valore è parametrico; dipende dalla configurazione del sistema)*

- intNUMERO_MAX_RECENSIONI_NEGATIVE
  Quando un utente supera la soglia indicata dal numero massimo di recensioni negative, diviene 'sospetto'.
  Esempio : 5
- int NUMERO_MAX_DEROGHE_BAN
  Quando un utente "importante" supera, su richiesta di Administration, la soglia indicata dal numero massimo di deroghe, diviene "SUPER_SOSPETTO".
  Esempio : 4

- intNUMERO_MAX_ANNUNCI_UTENTE
  Ogni utente ha un tot di numero di annunci che può pubblicare.
  Esempio : 10
- intNUMERO_MAX_PAROLE
  Ogni testo inserito, che sia un annuncio o una valutazione, ha un limite sulla lunghezza ovvero sul conteggio delle parole.
  Esempio : 200
- STATO_SOSPETTO_UTENTE
  A ogni utente è associato un livello di affidabilità che inizialmente è pari a 100 e decresce per ogni violazione ammessa dall'utente nel sito FERSA. Quando un utente raggiunge un livello di affidabilità che non supera la prevista soglia minima, egli diviene 'sospetto'.
  Esempio : STATO_SOSPETTO_UTENTE < 30
- STATO_NORMALE_UTENTE
  Un utente è creato in stato NORMALE e lo mantiene fintanto che non diviene SOSPETTO o BANNATO.
- STATO_BANNATO_UTENTE
  Un utente può divenire BANNATO tramite la decisione del back office. Un utente bannato non ha più permesso d'accesso nella piattaforma utilizzando la stessa email.
- PREZZO_VALIDO
  Un range di valori minimo e massimo del prezzo ammissibile.
  Esempio :min 100.00 , max 10000.00 $
- PREZZO_M^2
  Un range di valori minimo e massimo del prezzo al metro quadro ammissibili.
- GRAVITA_DI_PAROLA: NULLA; LIEVE; MEDIA; ALTA
  Ogni parola non ammissibile ha un peso diverso. Questo viene utilizzato nel calcolo dell'ammissibilità del testo e nel cambio di affidabilità dell'utente.
  Esempio :
  ALTA : -10
  MEDIA : -5
  LIEVE : -3
  NULLA: 0
- INAMMISSIBILITA
  Un testo per ogni nuova parola inammissibile in esso contenuta avrà un valore di inammissibilità sempre più alto. Superata una fissata soglia di inammissibilità, il testo sarà essere respinto

*Interfacce esposte dal Sotto-Sistema (firma di ciascuna delle operazioni esposte):*

## *Operazioni Esportate:*

1. boolean checkAnnounce(AnnounceId)

   L'interfacciache controlla se l'annuncio può essere pubblicato, ovvero se è conforme alle regole di sistema.

   **Sottosistemi interessati: Announcing**

   Operazioni interne :

   - checkText(Text): boolean (controlla se il testo può essere pubblicato, per lunghezza e per parole ammissibili)
   - checkMedia(Media): boolean (controlla se il video può essere pubblicato)
   - checkImage(Image): boolean (controlla se l'immagine può essere pubblicato)
   - checkPrice(Int) : boolean (controlla se il prezzo è valido)
   - checkNumMaxPublishedAnnounces(User_ID) : boolean (controlla se l'utente ha raggiunte il limite di annunci pubblicabili)
   - checkRentTime(Int) : boolean (controlla se la durata dell'affitto scritta è nei limiti imposti dalla legge)

2. ValidatedData getValidatedData(Nickname nickname)

   Si inviano a UserProfile i dati validati dell'utente:nickname,email, dati anagrafici.

   **Sottosistemi interessati: User Profile**

   Operazioni interne :

   - checkUserInfo(UserID): booleanVerifica se la registrazione dell'utente è valida, ovvero se i dati inseriti sono conformi alle regole di sistema.
   - checkNickname(Text) :boolean (controlla se il nickname non contiene parole proibite)
   - checkEmail(Text): boolean (controlla se l'email è già stata usata e se contiene parole proibite)
   - checkVitalData(Text) : boolean (controlla dati anagrafici : maggiore età. etc)

3. boolean checkEvaluation(EvalId)

   Controlla se la recensione può essere pubblicata, ovvero se è conforme alle regole di sistema.

   **Sottosistemi interessati: Evaluation**

   - checkText(String)

4. boolean checkSignal(SignalIdsegnalazione)

   Controlla se la segnalazione avvenuta è valida. La segnalazione è un oggetto che ha : motivazione segnalazione, id utente segnalante, id utente segnalato.

   **Sottosistemi interessati: Announcing, Evaluation**

   Operazione interna :

   - booleancheckUserwithNegativeEvaluation(Nickname)
     Effettua un controllo sugli utenti che hanno superato un alto tasso di recensioni negative, cioè il back office decide se bannare o meno l'utente.
   - Affidability checkAffidability(Nickname) : controlla lo stato dell'utente e lo pone come 'sospetto' a seguito delle segnalazioni. Il Back Office del sistema deciderà del destino dell'utente (se bannarlo o ripristinarlo a normale).

5. List <Rule> showRules()

   Prevede la visualizzazione di tutte le regole del sistema.

   Le regole di FERSA,essendo parametrizzate, devono poter essere visibili agli altri sottosistemi che ne fanno uso in caso di necessità.

   **Sottosistemiinteressati: Announcing, Evaluation, User Profile, Role , People**

**void derogateBan (importantUser Nickname);**
/* 20190116 A seguire la richiesta di ban, l'amministratore risponde chiedendo di derogare in quanto utente special */

*Operazioni Importate:*

***User Profile&Role& Status management***

1. List <UserRole>getUserRoles(Nickname nickname)

Permette di ottenere i ruoli svolti dall'utente nel sistema

2. UserStatusgetUserStatus(Nickname nickname)

Permette di ottenere lo status dell'utente nel sistema

3. booleanchangeUserStatus(Nickname nickname, UserStatus uStatus)

Cambia lo status corrente dell'utente <nickname> in status

4. booleanisUserActive(Nickname nickname)

Permette di capire se un utente possiede lo status di ATTIVO

5. booleanisUserInactive(Nickname nickname)

Permette di capire se un utente possiede lo status INATTIVO

6. booleanisUserCanceled(Nickname nickname)
   Permette di capire se l'utente è stato CANCELLATO
7. booleanisUserBanned(Nickname nickname)
   Permette di capire se l'utente è stato BANNATO

***Evaluation***

8. void reportOffensiveUser(Nickname nickname)
   Aggiunge un utente (che ha scritto recensione offensiva o che ha raggiunto il numero massimo di recensioni negative) agli utenti segnalati
9. SignalInfo getSignalInfo(SignalId)
   Permette di ottenere le informazioni relative alla segnalazione : Utente segnalante, utente segnalato, e motivazione segnalazione

***Announcing***

10. SignalInfo getSignalInfo(SignalId)
    Permette di ottenere le informazioni relative alla segnalazione : Utente segnalante, utente segnalato, e motivazione segnalazione
11. AnnouncegetAnnounceInfo(AnnounceID, Nickname)
    Permette di ottenere tutte le informazioni dell'annuncio e i relativi media presenti in esso.

Classi da definire (o importare? Da dove?)

ValidatedData
SignalIds
Signal

# Fine Interfacce Sottosistemi 20190105-01