Documento di architettura

Scopo del documento **1**
Diagramma delle classi **2**
Utente 2
Gestione Evento 4
Gestione Immagini 4
Gestione Utente 4
Gestione Autenticazione 5
Diagramma delle classi complessivo 7
Codice in OCL **8**
createEvent 8
editAvailability / editEventCoverImage 8
markParticipationToEvent / likeComment 9
sendNotificationForEditedEvent 10
sendFollowRequest 10
acceptFollowReques 10
sendVerificationEmail / sendVerificationSms 11
Diagramma delle classi con codice OCL 12

Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto E-20 usando diagrammi delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL). Nel precedente documento è stato presentato il diagramma degli use case, il diagramma di contesto e quello dei componenti. Ora, tenendo conto di questa progettazione, viene definita l'architettura del sistema dettagliando da un lato le classi che dovranno essere implementate a livello di codice e dall'altro la logica che regola il comportamento del software. Le classi vengono rappresentate tramite un diagramma delle classi in linguaggio UML. La logica viene descritta in OCL perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

Diagramma delle classi

Nel presente capitolo vengono presentate le classi previste nell'ambito del progetto E-20. Ogni componente presente nel diagramma dei componenti diventa una o più classi. Tutte le classi individuate sono caratterizzate da un nome, una lista di attributi che identificano i dati gestiti dalla classe e una lista di metodi che definiscono le operazioni previste all'interno della classe. Ogni classe può essere anche associata ad altre classi e, tramite questa associazione, è possibile fornire informazioni su come le classi si relazionano tra loro.
Riportiamo di seguito le classi individuate a partire dai diagrammi di contesto e dei componenti. In questo processo si è proceduto anche nel massimizzare la coesione e minimizzare l'accoppiamento tra classi.

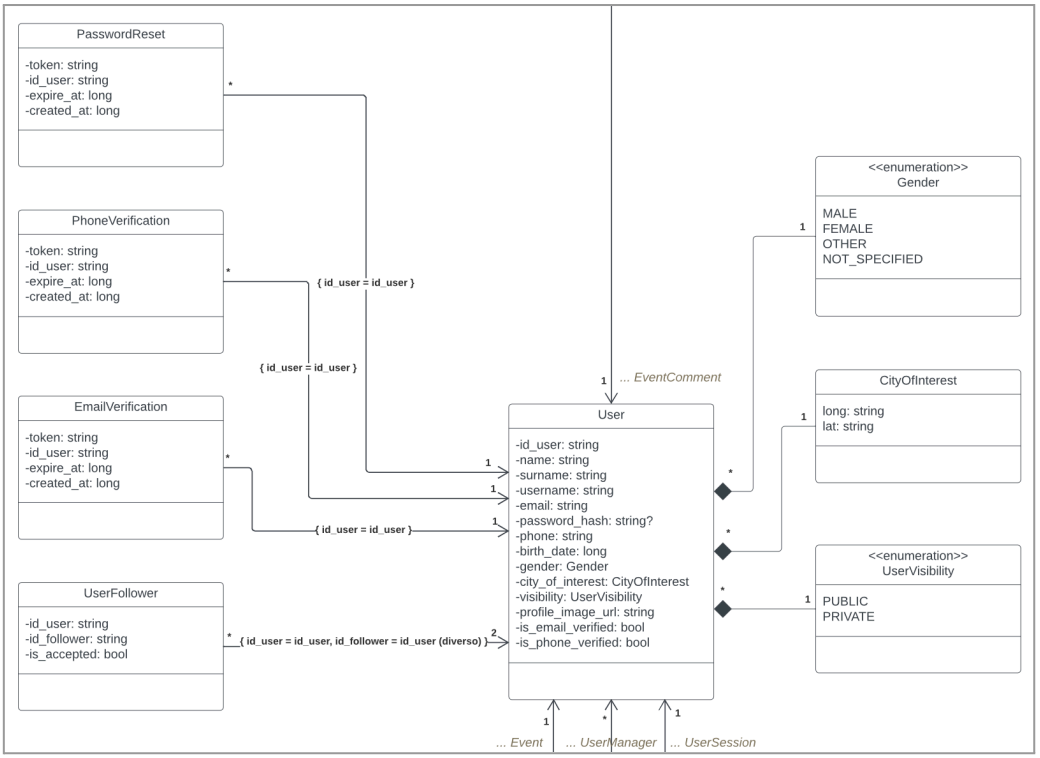
Utente

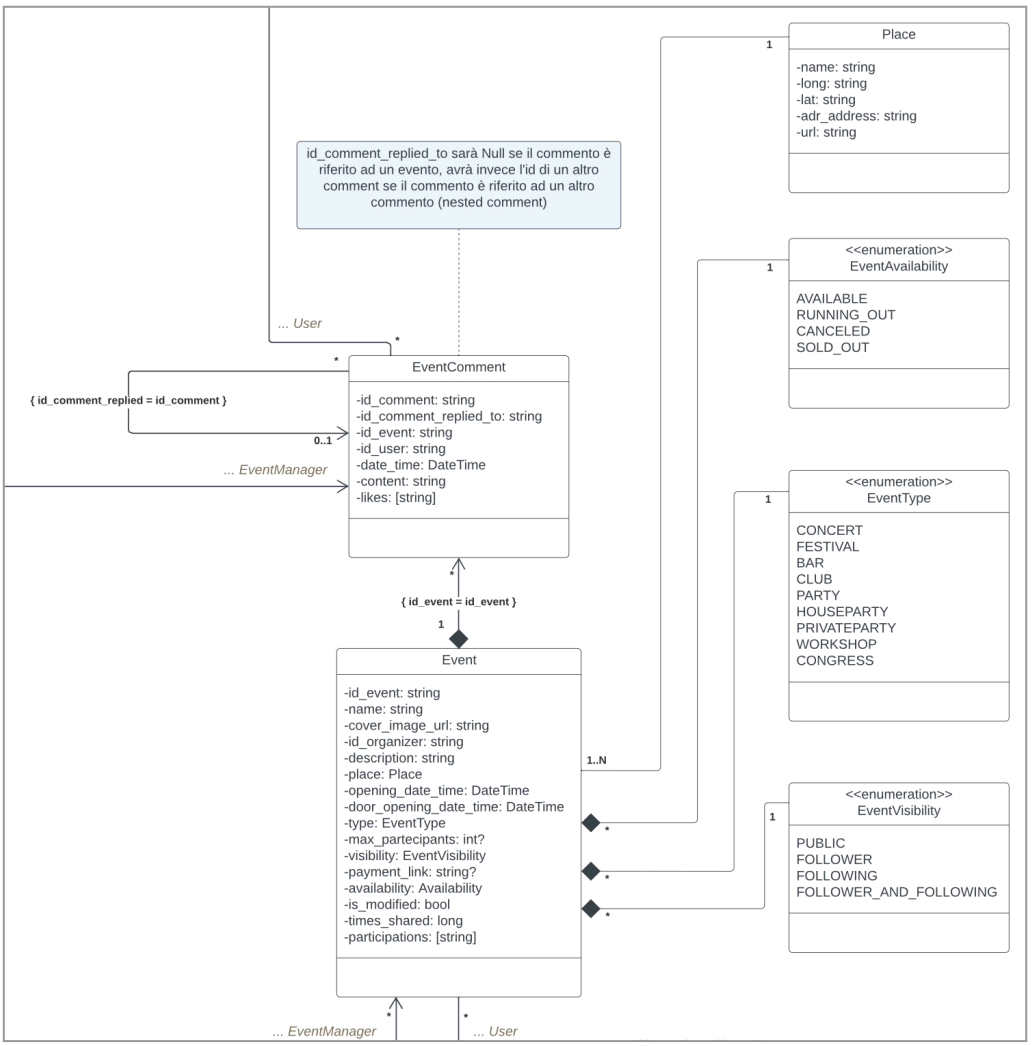
Analizzando i tre diagrammi del contesto proposti nel documento precedente sono state individuate due tipologie di utente: autenticato e non autenticato.

L'utente non autenticato e' colui che non ha ancora effettuato il login o la registrazione. Le funzionalità dell'utente non autenticato sono limitate alla visualizzazione della bacheca eventi e alla ricerca degli eventi.

Il campo *id_user* nella classe User costituisce un identificatore univoco per ciascun utente. Nel caso in cui un utente non sia autenticato, il valore di *id_user* sarà nullo. In presenza di tale condizione, l'utente verrà privato dell'accesso a determinate funzionalità, come la creazione di un nuovo evento (Event) o la possibilità di inserire commenti nella sezione dei commenti di un evento (EventComment).

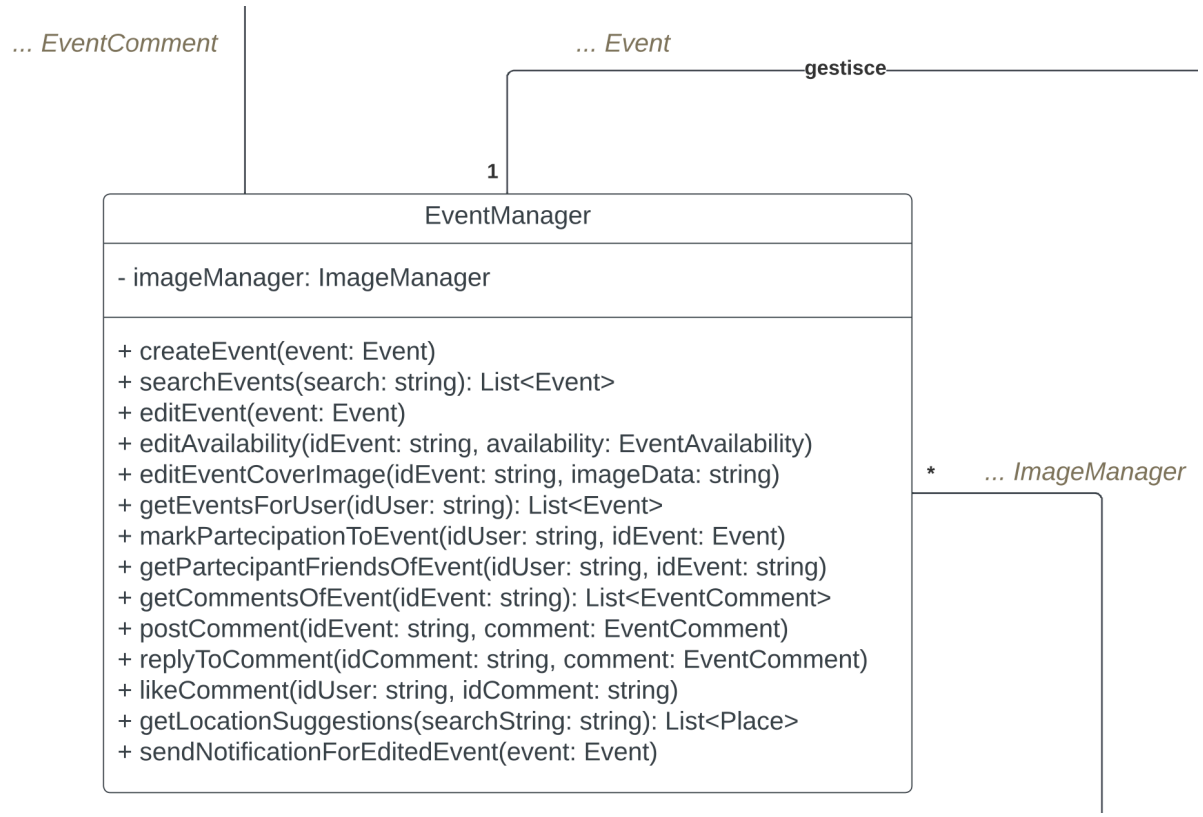
Le classi PasswordReset, PhoneVerification, EmailVerification sono classi che memorizzano temporaneamente un *token* utile alla verifica dell'identita' dell'utente. Queste classi memorizzano temporaneamente il token fino all'orario stabilito nel campo *expire_at*.


PasswordReset
-token: string
-id_user: string
-expire_at: long
-created_at: long
PhoneVerification
-token: string
-id_user: string
-expire_at: long
-created_at: long
EmailVerification
-token: string
-id_user: string
-expire_at: long
-created_at: long
UserFollower
-id_user: string
-id_follower: string
-is_accepted: bool
{ id_user = id_user }
{ id_user = id_user }
{ id_user = id_user }
{ id_user = id_user, id_follower = id_user (diverso) }
... EventComment
User
-id_user: string
-name: string
-surname: string
-username: string
-email: string
-password_hash: string?
-phone: string
-birth_date: long
-gender: Gender
-city_of_interest: CityOfInterest
-visibility: UserVisibility
-profile_image_url: string
-is_email_verified: bool
-is_phone_verified: bool
... Event
... UserManager
... UserSession
<<enumeration>>
Gender
MALE
FEMALE
OTHER
NOT_SPECIFIED
CityOfInterest
long: string
lat: string
<<enumeration>>
UserVisibility
PUBLIC
PRIVATE


id_comment_replied_to sarà Null se il commento è riferito ad un evento, avrà invece l'id di un altro comment se il commento è riferito ad un altro commento (nested comment)
... User
EventComment
-id_comment: string
-id_comment_replied_to: string
-id_event: string
-id_user: string
-date_time: DateTime
-content: string
-likes: [string]
{ id_comment_replied = id_comment }
... EventManager
{ id_event = id_event }
Event
-id_event: string
-name: string
-cover_image_url: string
-id_organizer: string
-description: string
-place: Place
-opening_date_time: DateTime
-door_opening_date_time: DateTime
-type: EventType
-max_partecipants: int?
-visibility: EventVisibility
-payment_link: string?
-availability: Availability
-is_modified: bool
-times_shared: long
-participations: [string]
... EventManager
... User
Place
-name: string
-long: string
-lat: string
-adr_address: string
-url: string
<<enumeration>>
EventAvailability
AVAILABLE
RUNNING_OUT
CANCELED
SOLD_OUT
<<enumeration>>
EventType
CONCERT
FESTIVAL
BAR
CLUB
PARTY
HOUSEPARTY
PRIVATEPARTY
WORKSHOP
CONGRESS
<<enumeration>>
EventVisibility
PUBLIC
FOLLOWER
FOLLOWING
FOLLOWER_AND_FOLLOWING

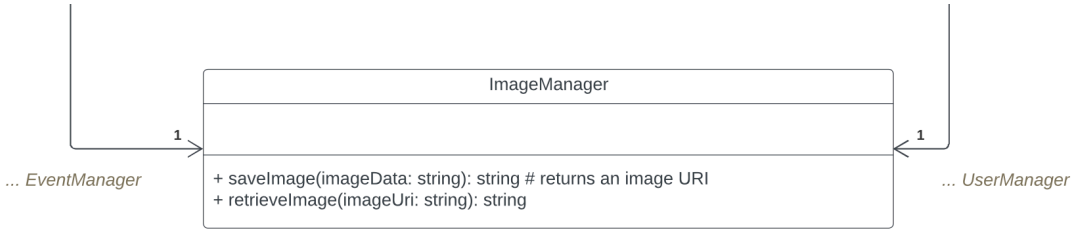
Gestione Evento

La classe EventManager si occupa della gestione di alcune funzionalita' legate alla classe Event.



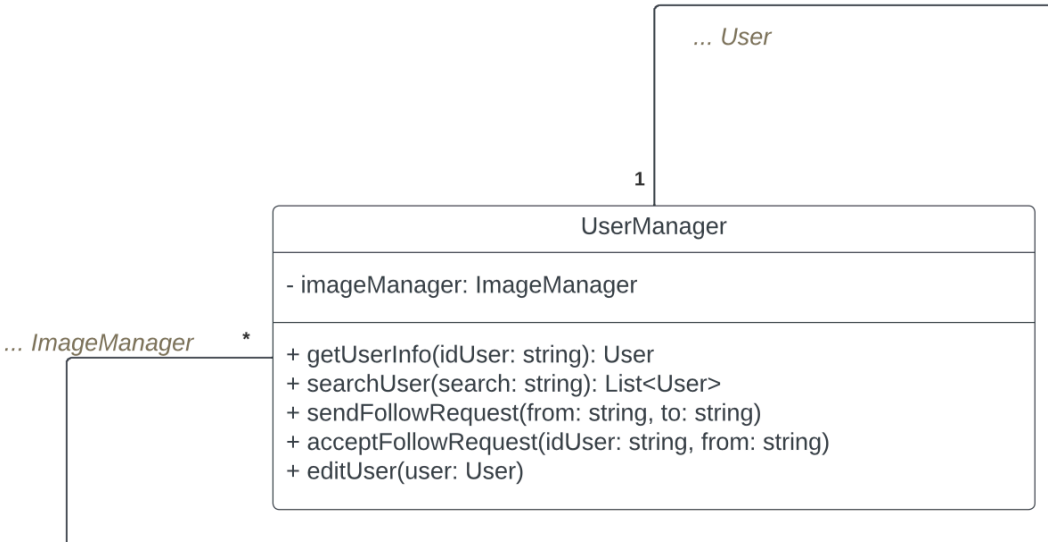
Gestione Immagini

Nel diagramma di contesto analizzato è presente un sistema paritario "S3" il quale si occupa della gestione delle immagini profilo dell'utente e delle immagini di copertina degli eventi. Per questo motivo e' stata individuata la classe ImageManager, la quale salva l'url delle immagini in una stringa.



Gestione Utente

La classe UserManager si occupa della gestione di alcune funzionalità legate all'utente, come l'inserimento/modifica dell'immagine profilo, sistema di follow/follower e modifica delle informazioni dell'utente.



Gestione Autenticazione

Nel diagramma di contesto analizzato sono presenti 3 sistemi paritari "Apple OAuth", "Facebook OAuth" e "Google OAuth" ma è anche presente la possibilità di autenticarsi attraverso credenziali a scelta.

Per gestire l'autenticazione e' stata quindi creata la classe AuthenticationManager. L'autenticazione avviene attraverso l'inserimento della mail o numero di telefono e della password corretti.

AuthenticationManager si occupa della registrazione di un nuovo utente. La registrazione puo' avvenire tramite inserimento manuale di tutti i campi necessari specificati nel **RF8** oppure tramite accesso all'account Google, Facebook o Apple e le ulteriori informazioni necessarie. In questo ultimo caso la web-app si interfaccerà con il sistema esterno per la gestione dell'autenticazione.

AuthenticationManager si occupa anche dell'operazione di verifica del numero di telefono e della mail dell'utente. Senza la verifica delle credenziali, l'utente non potra' effettuare l'accesso.

La classe UserSession si occupa della memorizzazione temporanea delle informazioni utili per la visualizzazione degli eventi nei dintorni dell'utente.
Le informazioni verranno cancellate alla scadenza della durata massima di una sessione di autenticazione.

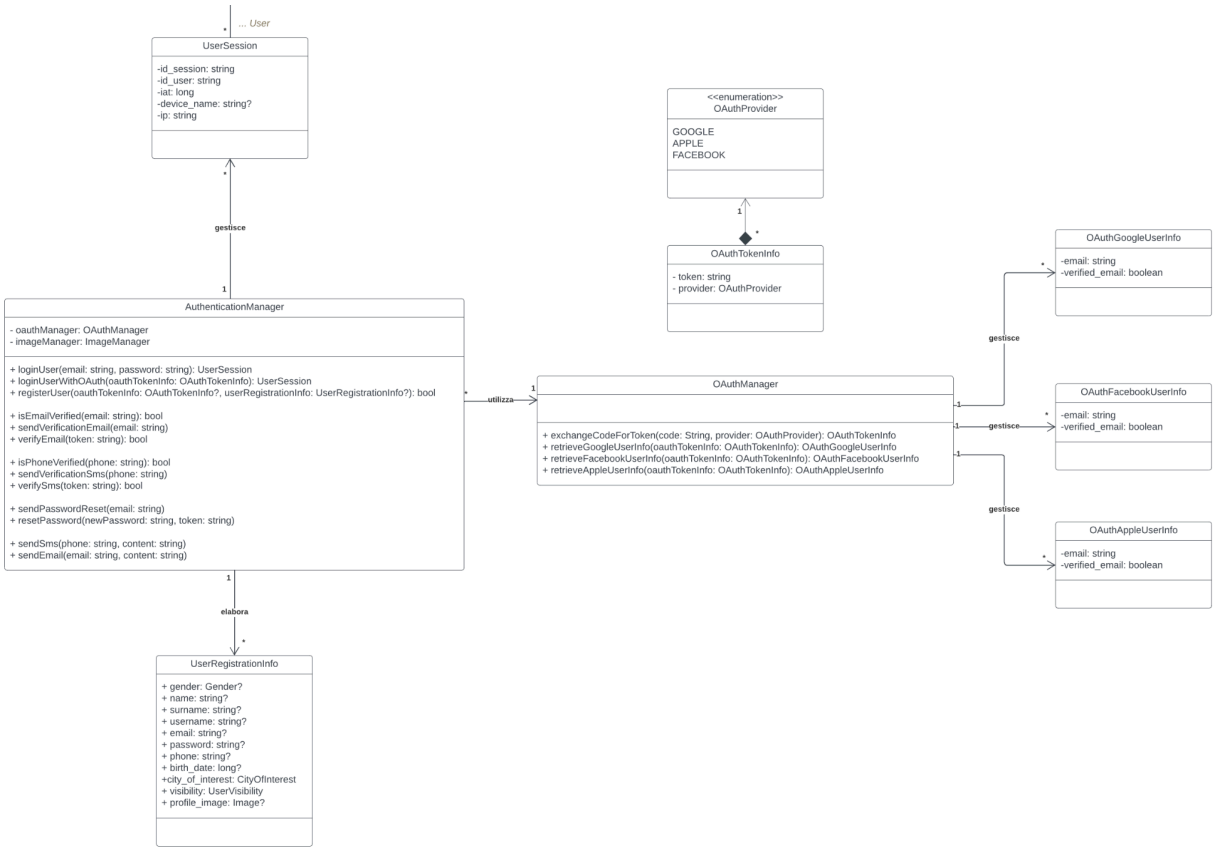

... User
UserSession
-id_session: string
-id_user: string
-iat: long
-device_name: string?
-ip: string
gestisce
AuthenticationManager
- oauthManager: OAuthManager
- imageManager: ImageManager
+ loginUser(email: string, password: string): UserSession
+ loginUserWithOAuth(oauthTokenInfo: OAuthTokenInfo): UserSession
+ registerUser(oauthTokenInfo: OAuthTokenInfo?, userRegistrationInfo: UserRegistrationInfo?): bool
+ isEmailVerified(email: string): bool
+ sendVerificationEmail(email: string)
+ verifyEmail(token: string): bool
+ isPhoneVerified(phone: string): bool
+ sendVerificationSms(phone: string)
+ verifySms(token: string): bool
+ sendPasswordReset(email: string)
+ resetPassword(newPassword: string, token: string)
+ sendSms(phone: string, content: string)
+ sendEmail(email: string, content: string)
utilizza
elabora
UserRegistrationInfo
+ gender: Gender?
+ name: string?
+ surname: string?
+ username: string?
+ email: string?
+ password: string?
+ phone: string?
+ birth_date: long?
+city_of_interest: CityOfInterest
+ visibility: UserVisibility
+ profile_image: Image?
<<enumeration>>
OAuthProvider
GOOGLE
APPLE
FACEBOOK
OAuthTokenInfo
- token: string
- provider: OAuthProvider
OAuthManager
+ exchangeCodeForToken(code: String, provider: OAuthProvider): OAuthTokenInfo
+ retrieveGoogleUserInfo(oauthTokenInfo: OAuthTokenInfo): OAuthGoogleUserInfo
+ retrieveFacebookUserInfo(oauthTokenInfo: OAuthTokenInfo): OAuthFacebookUserInfo
+ retrieveAppleUserInfo(oauthTokenInfo: OAuthTokenInfo): OAuthAppleUserInfo
gestisce
gestisce
gestisce
OAuthGoogleUserInfo
-email: string
-verified_email: boolean
OAuthFacebookUserInfo
-email: string
-verified_email: boolean
OAuthAppleUserInfo
-email: string
-verified_email: boolean

Diagramma delle classi complessivo

Codice in OCL

In questo capitolo è descritta in modo formale la logica prevista nell'ambito di alcune operazioni di alcune classi. Tale logica viene descritta in Object Constraint Language (OCL) perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

createEvent

Gli utenti autenticati sono abilitati a inserire un nuovo evento attraverso diverse voci di dati. È necessario che tutti i campi richiesti siano correttamente popolati e privi di valori vuoti. Tale condizione è stata formalizzata utilizzando una precondizione nell'OCL (Object Constraint Language).
È importante notare che il campo *id_event* verrà generato automaticamente durante la creazione di un nuovo evento.
Inoltre, i campi *payment_link* e *max_partecipant* hanno la possibilità di essere impostati come valori nulli.

```
Unset
context EventManager::createEvent(event:Event)
pre: event.name != null, event.cover_image_url != null, event.id_organizer
!= null, event.description != null, event.place != null,
event.opening_date_time != null, event.type != null, event.visibility !=
null, event.payment_link != null
```

editAvailability / editEventCoverImage

Nella classe EventManager è presente anche il metodo *editAvailability / editEventCoverImage*.
Prima di procedere alla modifica di un parametro, quale sia un enum oppure un immagine, è necessario controllare che esiste un evento il quale id corrisponda a quello passato per parametro. Questa condizione è rappresentata da una

precondizione nell'OCL. Nella postcondizione dell'OCL e' rappresentato invece la modifica della variabile interessata.

```
Unset
context EventManager::editAvailability(idEvent: string, availability:
EventAvailability)
pre: self.event->exists(e | e.idEvent = idEvent)
post: self.event->any(e | e.idEvent = idEvent).availability = availability
```

```
Unset
context EventManager::editEventCoverImage(idEvent: string, imageData:
string)
pre: self.event->exists(e | e.idEvent = idEvent)
post: self.event->any(e | e.idEvent = idEvent).imageData = imageData
```

markParticipationToEvent / likeComment

Nella classe EventManager, sono inclusi i metodi *markParticipationToEvent* e *likeComment*. Prima di aggiungere un nuovo *idUser*, è essenziale effettuare una verifica per assicurarsi che esista un evento o un commento corrispondente e che l'utente non sia già stato incluso nell'elenco dei partecipanti o dei 'like' relativi a un commento. Questa condizione è formalmente espressa come precondizione nel linguaggio OCL (Object Constraint Language).

La postcondizione, invece, si occupa dell'aggiunta di un nuovo elemento all'elenco delle persone che partecipano a un evento o hanno espresso apprezzamento ('like') per un commento.

```
Unset
context EventManager::markParticipationToEvent(idUser: String, idEvent:
String)
pre: self.event->exists(e | e.idEvent = idEvent and
e.partecipation->excludes(idUser))
post: self.event->any(e | e.idEvent =
idEvent).participation->includes(idUser)
```

```
Unset
context EventManager::likeComment(idUser: String, idComment: String)
pre: self.event->exists(c | c.idComment = idComment and
c.likes->excludes(idUser))
post: self.event->any(c | c.idComment = idComment).likes->includes(idUser)
```

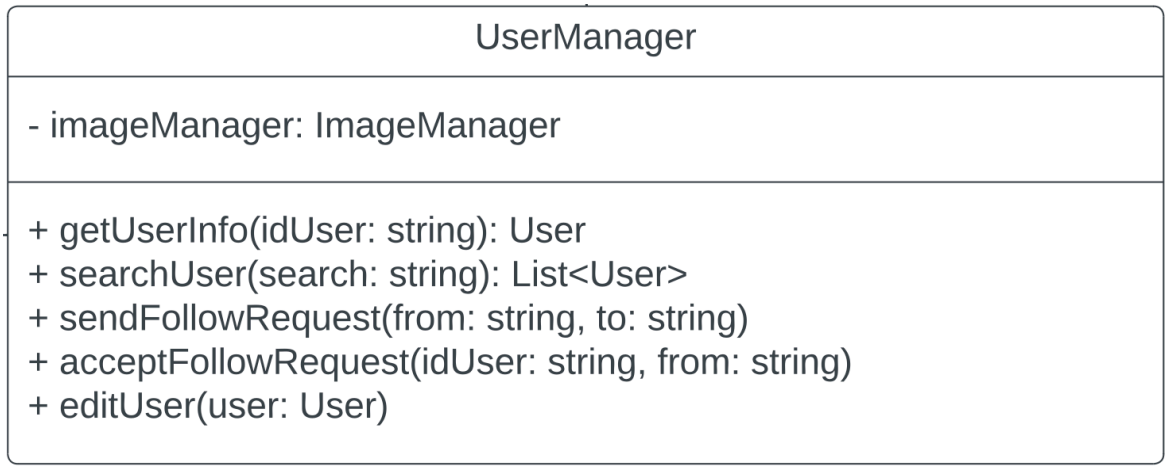
sendNotificationForEditedEvent

Nella classe EventManager e' incluso il metodo *sendNotificationForEditedEvent*, il quale si occupa di inviare una notifica per mail / sms quando viene modificato un evento. Questa condizione e' espressa con una precondizione nell'OCL.

```
Unset
context EventManager::sendNotificationForEditedEvent(event: Event)
pre: event.is_modified = true
```

sendFollowRequest

Nella classe UserManager è presente il metodo *sendFollowRequest*, il quale invia una richiesta di follow tra 2 utenti diversi. Questa condizione e' rappresentata da una precondizione nel linguaggio OCL.




```
Unset
context UserManager::sendFollowRequest(from: string, to: stirng)
pre: from != to
```

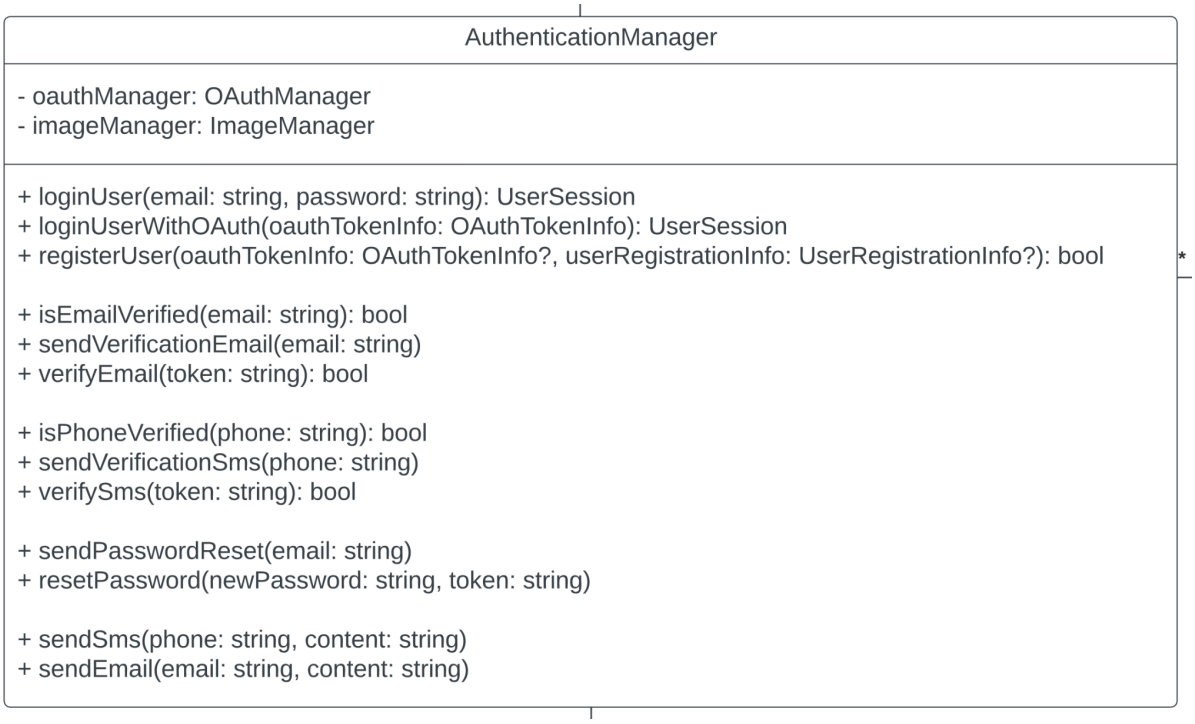
acceptFollowReques

Nella classe UserManager e' presente il metodo *acceptFollowRequest*, il quale accetta una richiesta di follow da parte di un utente. Ogni volta che viene accettata una richiesta di follow, bisogna cambiare il valore della variabile is_accepted della richiesta. Questa condizione e' rappresentata nella postcondizione nel linguaggio OCL.

```
Unset
context UserManager::acceptFollowRequest(idUser: String, from: String)
pre: self.request->exists(r | r.to = idUser and r.from = from)
post: self.request->any(r | r.to = idUser and r.from = from).is_accepted =
true
```

sendVerificationEmail / sendVerificationSms

Nella classe AuthenticationManager sono presenti i metodi *sendVerificationEmail* e *sendVerificationSms*. Questi metodi si occupano della verifica della mail e del numero di telefono di un utente. Nel caso la mail / numero di telefono e' gia' stato verificato, non è necessario l'invio di una notifica di verifica. Questa condizione e' espressa come precondizione nel linguaggio OCL.




```
Unset
context AuthenticationManager::sendVerificationEmail(email: String)
pre: User.allInstances()->exists(u | u.email = email and not
u.is_email_verified)
```

```
Unset
context AuthenticationManager::sendVerificationSms(email: String)
pre: User.allInstances()->exists(u | u.phone = phone and not
u.is_email_verified)
```

Diagramma delle classi con codice OCL