Documento di Architettura

Progetto:
4SPOT

Titolo del documento:
Documento di architettura

Document info:

Doc. name	Documento di Architettura
Doc. number	*D3*
Description	*Il documento include diagrammi delle classi e codice in OCL.*

Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto *4SPOT* 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*.

1. Diagramma delle classi

Nel presente capitolo vengono presentate le classi previste nell'ambito del progetto *4SPOT*.
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 di loro.
Riportiamo di seguito le classi individuate a partire dai diagrammi di contesto e dei componenti.
In questo processo si è preceduto anche nel massimizzare la coesione e l'accoppiamento tra classi.

1.1 Pagina di accesso, sistema di autenticazione, pagina di registrazione, pagina di recupero password

In base a RF3, per usufruire dei servizi della piattaforma è necessario effettuare una procedura di login, tramite la Pagina di Login.
La pagina di Login è stata identificata come una classe, che consenta l'accesso per gli Stakeholder. di questa classe è stata individuata anche una sottoclasse per il login degli User non Supervisor.
Dalla pagina di login è possibile accedere anche alle pagine per la registrazione e per il recupero della password, ciascuna rappresentata da una classe a sè stante.
Le pagine di login e registrazione interagiscono con la classe AuthenticationSystem, che si occupa appunto dell'autenticazione degli Stakeholder nel sistema.

Pagine di Accesso

Le seguenti pagine sono le interfacce con cui l'Utente effettua l'accesso per poter sfruttare il servizio.

Pagina di Login

La Pagina di Login è la pagina di default verso la quale un Client non riportante un token valido viene reindirizzato.
Dalla Pagina di Login si può accedere con le proprie credenziali o richiedere la Pagina di Recupero della Password.

Pagina di Login Utente

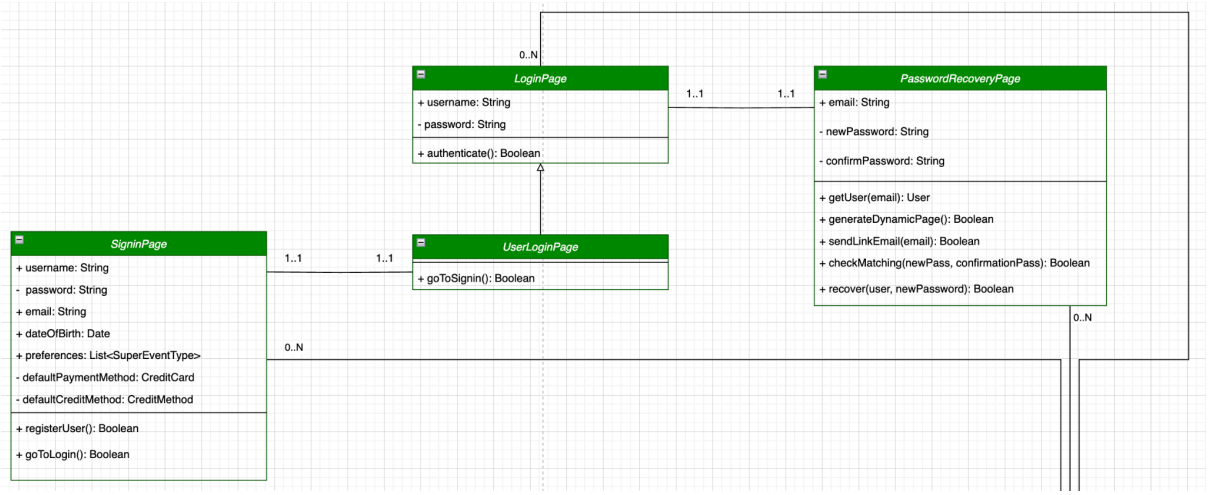
La Pagina di Login Utente è una sottoclasse della Pagina di Login, utilizzata specificatamente per l'accesso di un Utente (un Gestore non può utilizzarla).
La Pagina di Login Utente, oltre a fornire le funzionalità che eredita dalla superclasse, permette di passare alla Pagina di Signin (si noti che l'account di un Gestore non può essere creato da suddetta pagina).

Pagina Signin

La Pagina di Signin è la pagina utilizzata da un Client che vuole creare un Account.
Dalla Pagina di Signin Utente si può richiedere la Pagina di Login Utente.

Pagina di Recovery della Password

La Pagina di Recovery della Password permette ad uno Stakeholder con un Account di recuperare le proprie credenziali.
Dalla Pagina di Recovery della Password lo Stakeholder può richiedere la Pagina di Login al Server.

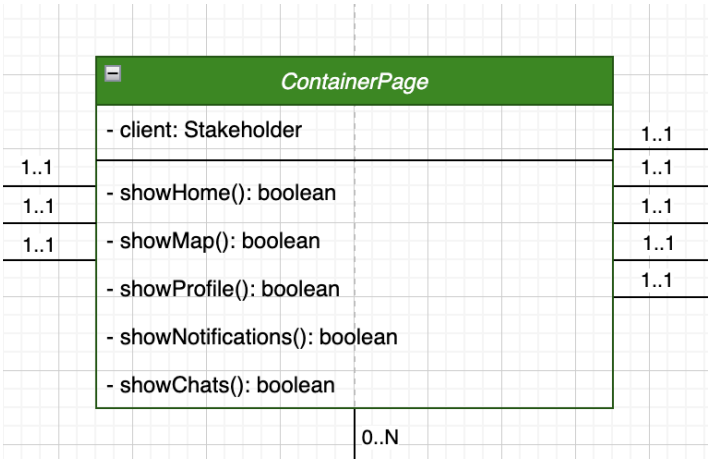


Pagine di Navigazione

Con l'espressione "Pagine Statiche" si indicano tutte quelle pagine "uniche" per ogni Utente. Ad esempio, un'istanza della classe HomePage è l'unica ad essere utilizzata dallo stesso Utente, nonostante al suo interno più Eventi differenti possano essere visualizzati.

Container Page

La container Page rimane attiva quando una delle pagine Home Page, Map Page, Profile Page, Notifications Page o Chats Page. Funge da "cornice" per le altre pagine visitate durante la normale navigazione dell'App (ad eccezione delle Pagine di Accesso).



Home Page

L'HomePage è l'interfaccia in cui l'Utente può visualizzare gli Eventi proposti scorrendo la pagina.

Questa è stata individuata come una classe, che interagisce, oltre che con il CoreSystem, anche con il sistema di raccomandazione degli Eventi, che vengono visualizzati dall'Utente secondo specifici criteri di rilevanza.

Map Page

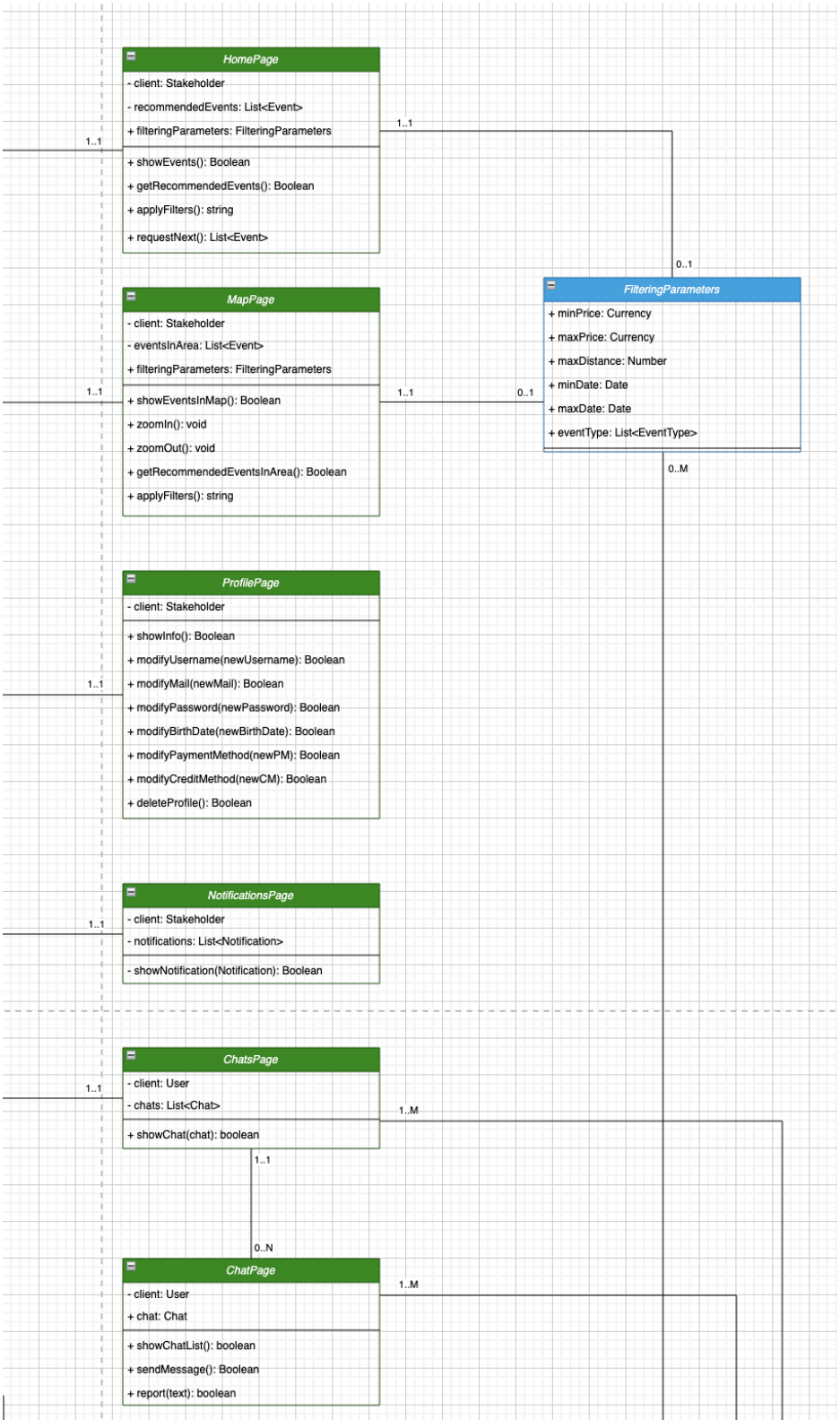
La Map Page è la pagina con cui l'Utente può cercare gli Eventi in una data area. Nel caso in cui si utilizzi geolocalizzazione, l'Utente può ottenere gli Eventi intorno a lui. Inoltre, l'Utente può specificare dei parametri di filtraggio che il sistema utilizzerà per selezionare gli Eventi da indicare nella Mappa.

Profile Page

La Profile Page è la pagina alla quale l'Utente può accedere ai propri dati, inclusi gli Eventi pubblicati, gli altri Utenti che segue e dai quali è seguito e altre informazioni indicate nella classe Utente (Stakeholder). Si noti che ogni Utente può accedere solo alla propria Pagina Profilo.

Notifications Page

La Notification Page è la pagina visualizzata dall'Utente che intende accedere alle notifiche inviategli dal Sistema.



User Page

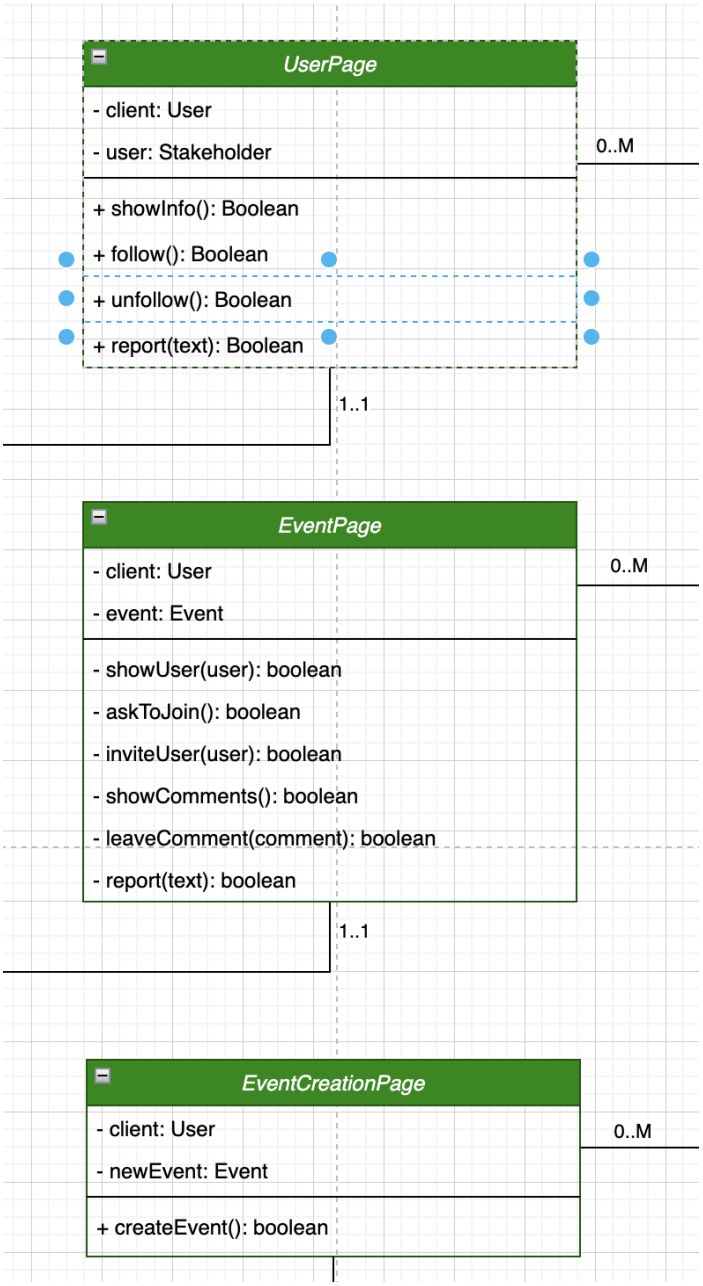
La Pagina Utente è associata ad ogni Utente ed è la pagina visualizzata da chiunque cerchi di accedere al profilo del suddetto (a parte lo stesso, il quale ha a disposizione la sua Profile Page).

Event Page

La Event Page è associata ad un Evento particolare. Dalla Event Page ogni Stakeholder può accedere alle informazioni dell'Evento in questione ed effettuare operazioni come iscriversi, richiedere di partecipare o lasciare un commento.

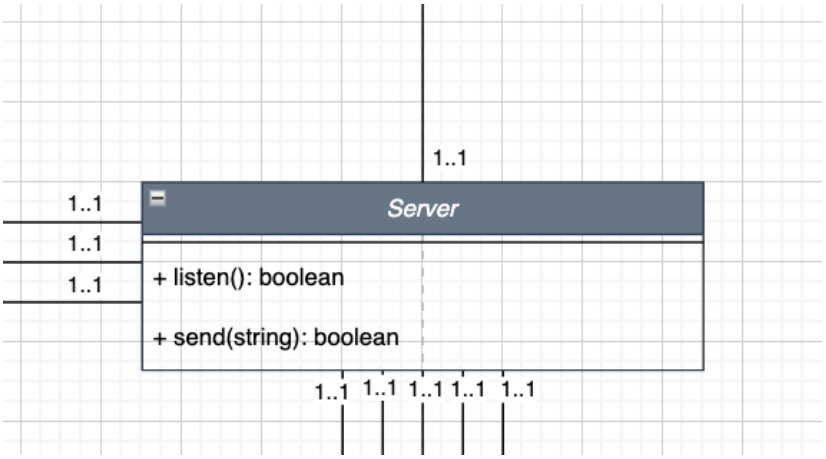
Event Creation Page

Dalla Event Creation Page un Utente può creare un Evento, così come specificato nei requisiti funzionali.


UserPage
- client: User
- user: Stakeholder
0..M
+ showInfo(): Boolean
+ follow(): Boolean
+ unfollow(): Boolean
+ report(text): Boolean
1..1
EventPage
- client: User
- event: Event
0..M
- showUser(user): boolean
- askToJoin(): boolean
- inviteUser(user): boolean
- showComments(): boolean
- leaveComment(comment): boolean
- report(text): boolean
1..1
EventCreationPage
- client: User
- newEvent: Event
0..M
+ createEvent(): boolean

Server

La classe Server si occupa di gestire l'ascolto delle richieste da parte dei client, utilizzare i vari sistemi di Back-End per effettuare le operazioni necessarie e rispondere ai client nel modo appropriato.



Sistemi di Back-End

Le classi qui descritte sono sistemi interfacciati al Server, che permettono di svolgere task specifici.

Sistema di Gestione Informazioni Utente

Il Sistema di Gestione Informazioni Utente è un sistema che consente la registrazione e la modifica di informazioni riguardanti l'Utente. Interagisce con il database per operazioni di recupero o registrazione di dati e con il sistema di verifica dei metodi di pagamento.

Sistema di Gestione dei Token

Il Sistema di gestione dei Token permette di generare e verificare i token inviati dai client in modo da assicurare il riconoscimento e l'autenticazione di questi.

Sistema di Validazione del Metodo di Pagamento

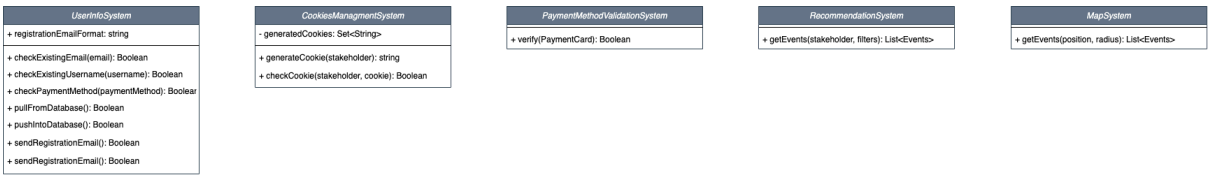
Il Sistema di Validazione del Metodo di Pagamento è un sistema che si occupa di verificare la validità dei metodi di pagamento inseriti dall'Utente per il proprio profilo interagendo con l'API di PayPal.

Sistema di Raccomandazione

Il Sistema di Raccomandazione permette di generare un insieme di Eventi "raccomandati" all'utente in base alla storia passata di quest'ultimo. Inoltre, tiene conto dei parametri di filtraggio inseriti dallo stesso.

Sistema di Geolocalizzazione Eventi (Map System)

Il Sistema di Geolocalizzazione Eventi permette di generare un insieme di Eventi in base alla localizzazione dell'Utente e ai parametri di filtraggio inseriti dallo stesso.



Classi Stakeholder

Le Classi Stakeholder sono utilizzate per organizzare le varie informazioni inerenti ad uno Stakeholder, così come definito nei requisiti funzionali. Tutte le seguenti classi ereditano dalla superclasse Stakeholder e sono memorizzate nel DB.

Stakeholder

La classe Stakeholder è la superclasse utilizzata per inserire le informazioni comuni a tutti i tipi di Stakeholder. Da un certo punto di vista tale classe può essere pensata come astratta, in quanto non viene mai realmente salvata un'istanza di Stakeholder che non sia anche un'istanza di una sua sottoclasse. In particolare, gli attributi di questa classe sono quelli necessari alla gestione della richiesta di login da parte di un Client.

Utente

La classe Utente, sottoclasse di Stakeholder, è specializzata per contenere le informazioni inerenti ad un singolo Utente. Permette a chi accede di svolgere azioni riservate al solo Utente (così come descritto nei RF).
Sebbene erediti da Stakeholder, User è a sua volta una classe astratta. Nella pratica, ogni User è istanza della classe Business oppure della classe Personal (non entrambe).

Business

Business è la classe che identifica un Utente con Account Business, così come descritto nei RF. L'unica aggiunta rispetto a User è l'attributo *IVA*, necessario affinché l'account venga registrato.

Personal

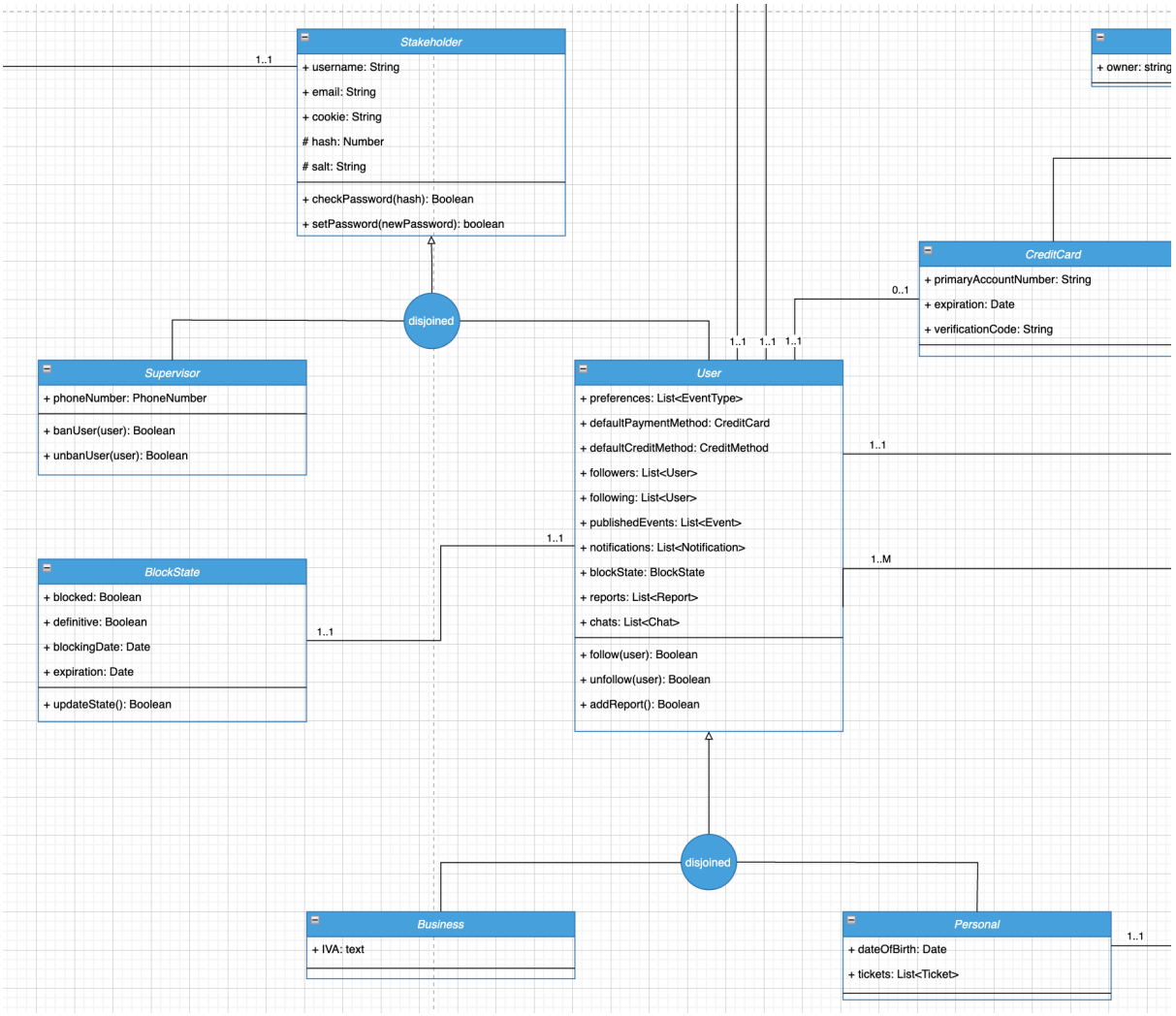
Personal è la classe che identifica un Utente con Account Personal, così come descritto nei RF.
Qui vengono aggiunti gli attributi dateOfBirth (necessaria per consentire all'Utente di accedere ad Eventi che richiedono un'età minima) e tickets, una lista di biglietti posseduti dall'Utente.

Supervisor

Supervisor è l'altro diretto discendente della classe Stakeholder. Un'istanza di tale classe identifica univocamente un Gestore del Sistema.

Block State

La classe Block State, univoca per ogni Utente...,



Metodi di pagamento

Metodo di pagamento (Credit Method)

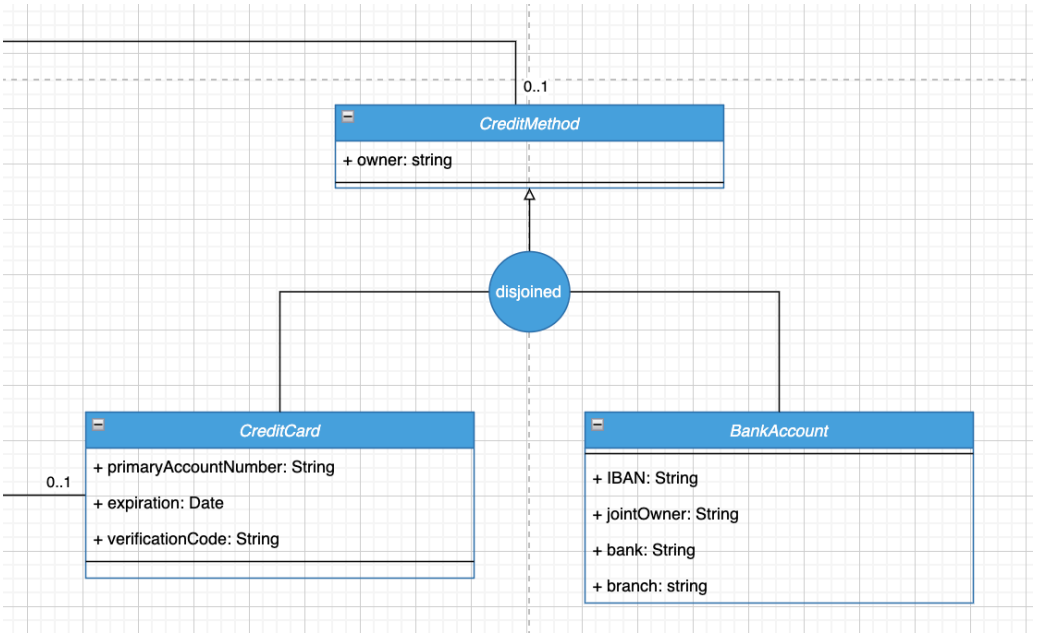
È la superclasse che racchiude tutte le possibilità di pagamento. Ha solo l'attributo 'owner'.

Carta di Credito

Classe racchiudente le fondamentali caratteristiche di una carta di credito per il pagamento. Il sistema esegue un check per verificare che esista e non sia scaduta.

Conto Bancario

Rappresenta un normale conto bancario. A differenza della classe precedente, non garantisce un controllo di tutti i campi.



Classi Evento

Event

La classe Evento racchiude tutti gli attributi associati allo stesso, in modo da garantire un accesso semplice. Inoltre, ha associate altre classi, come Event Type, per facilitarne la classificazione, o Currency, per aggiungere informazioni.
Come si può notare, Event è una classe molto estesa; in effetti, insieme a Stakeholder, ha un ruolo chiave nell'applicazione.

Requested Event

Data la documentazione, è stato necessario aggiungere Requested Event, nonostante non abbia alcun attributo in più. Tuttavia, in questo modo viene facilitato il riconoscimento, senza aggiungere altri attributi alla classe Evento, di per sé già molto estesa.

Evento Privato

Un Evento privato eredita da Event, con la sola differenza di tenere una lista di invitati.

Event Type

Per EventType si indica una tipologia di Evento, salvata su DB dal gestore (o comunque non aggiornabile dall'utente). Ogni Evento appartiene almeno ad una categoria Event Type.

Super Event Type

Non è altro che una macroclasse di Tipologie di Eventi. Un esempio è quello dello 'Sport', che racchiude discipline come la 'Pallavolo' all'interno.

Currency

Ha due attributi: state e amount. Serve per riconoscere le valute.

Location

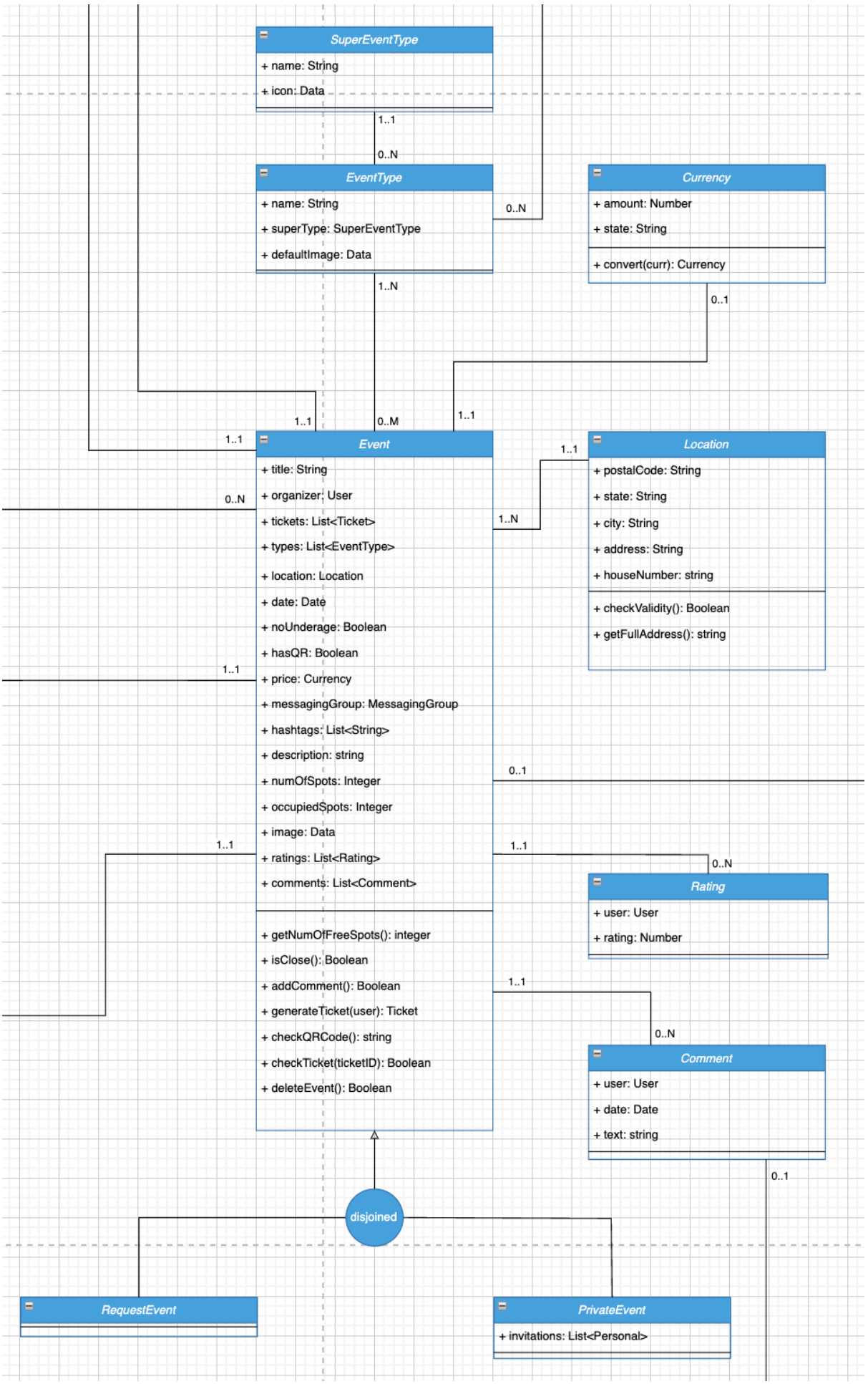
È una classe fondamentale, necessaria alla localizzazione degli Eventi (e al corretto funzionamento della Mappa).

Rating

Rating consente di racchiudere le informazioni di rating (per l'appunto) associate all'Evento e fornite da calcoli che interessano l'interazione degli altri utenti, come da documentazione.

Comment

Comment mantiene al suo interno gli attributi necessari per la corretta gestione dei commenti ad un Evento, compres testo, data e profilo commentante.


SuperEventType
+ name: String
+ icon: Data
1..1
0..N
EventType
+ name: String
+ superType: SuperEventType
+ defaultImage: Data
0..N
Currency
+ amount: Number
+ state: String
+ convert(curr): Currency
0..1
1..N
1..1
0..M
1..1
1..1
Event
+ title: String
+ organizer: User
+ tickets: List<Ticket>
+ types: List<EventType>
+ location: Location
+ date: Date
+ noUnderage: Boolean
+ hasQR: Boolean
+ price: Currency
+ messagingGroup: MessagingGroup
+ hashtags: List<String>
+ description: string
+ numOfSpots: Integer
+ occupiedSpots: Integer
+ image: Data
+ ratings: List<Rating>
+ comments: List<Comment>
+ getNumOfFreeSpots(): integer
+ isClose(): Boolean
+ addComment(): Boolean
+ generateTicket(user): Ticket
+ checkQRCode(): string
+ checkTicket(ticketID): Boolean
+ deleteEvent(): Boolean
0..N
1..1
1..N
Location
+ postalCode: String
+ state: String
+ city: String
+ address: String
+ houseNumber: string
+ checkValidity(): Boolean
+ getFullAddress(): string
1..1
0..N
1..1
0..1
1..1
0..N
Rating
+ user: User
+ rating: Number
1..1
0..N
Comment
+ user: User
+ date: Date
+ text: string
0..1
disjoined
RequestEvent
PrivateEvent
+ invitations: List<Personal>

Notifiche

Notifica

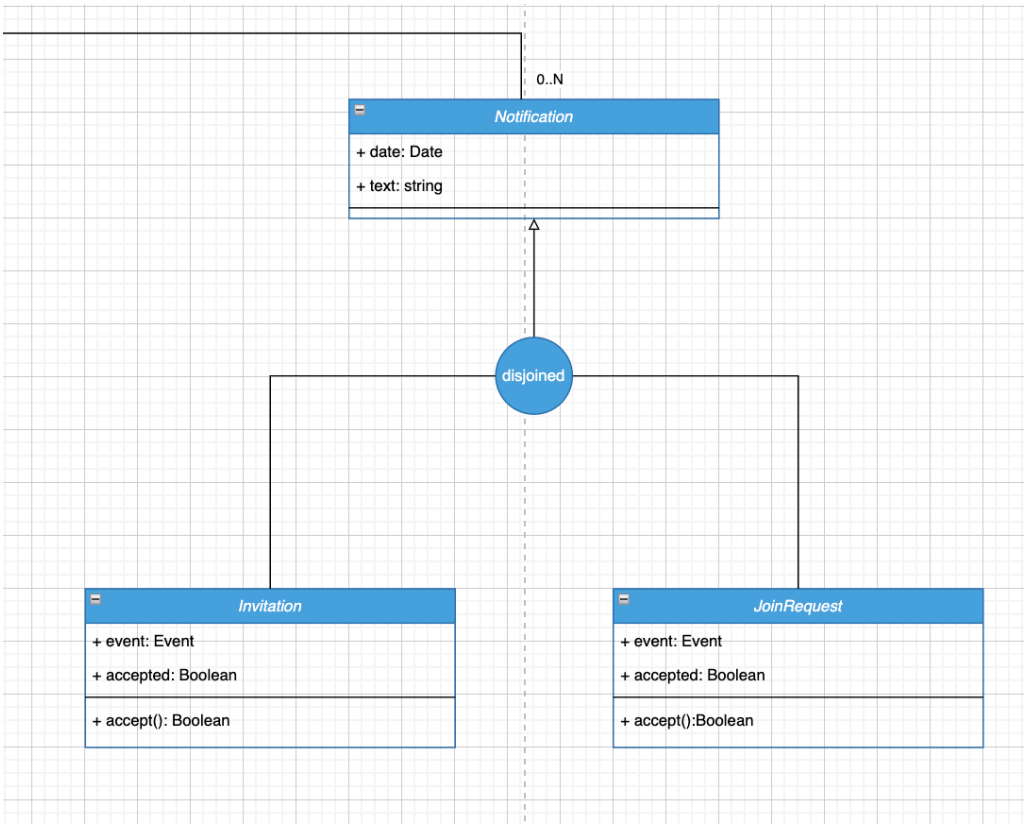
La classe serve a mantenere nel database informazioni per quanto riguarda la comunicazione fra sistema e Utente.

Invito

L'invito di partecipazione ad un Evento viene inviato dall'Organizzatore all'Utente per mezzo delle notifiche. Per questo, una Invitation è una notifica a parte.

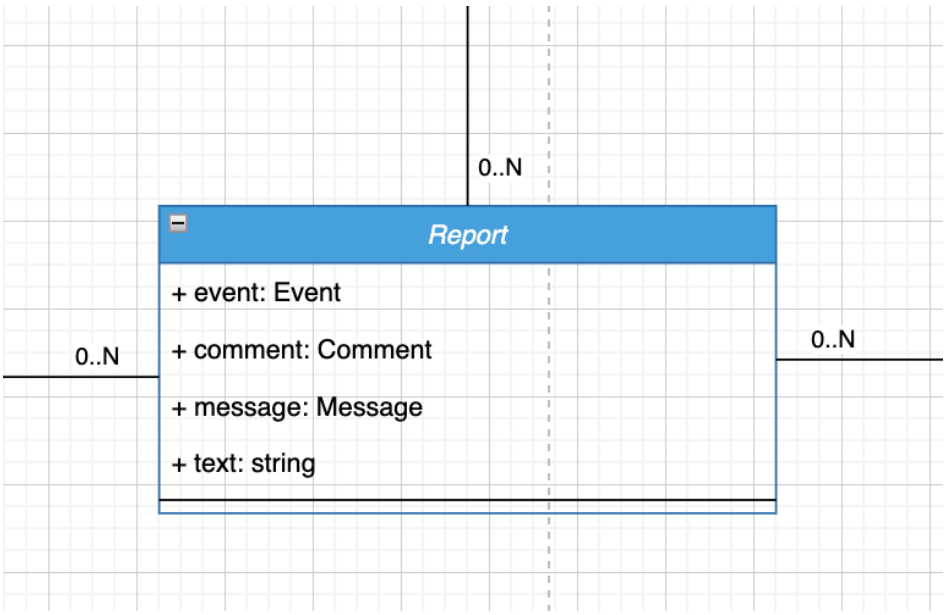
Richiesta di partecipazione (Join request)

Così come un Invito, una Join Request gode di caratteristiche proprie



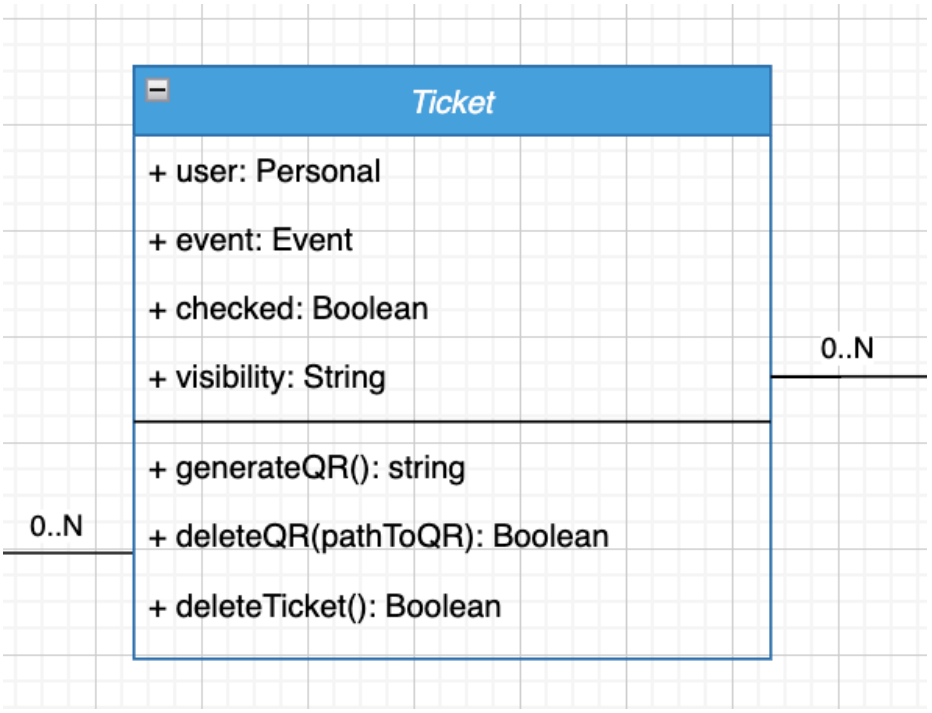
Report

Nel DB vengono salvate informazioni di tutti i report prodotti da Utenti, associati all'Utente oggetto dell'esposto.



Biglietti

La classe Tickets ha il semplice scopo di rendere comoda la gestione delle partecipazioni e iscrizioni agli Eventi.



Chat

Message

Un messaggio è semplicemente una corrispondenza tra due Utenti, un mittente e un destinatario.

Chat

Una Chat è un insieme di Messaggi fra lo stesso gruppo di Utenti.

Messaging Group

Quando una Chat interessa più di due Utenti, si parla di Messaging Group. Può essere associato ad un Evento.

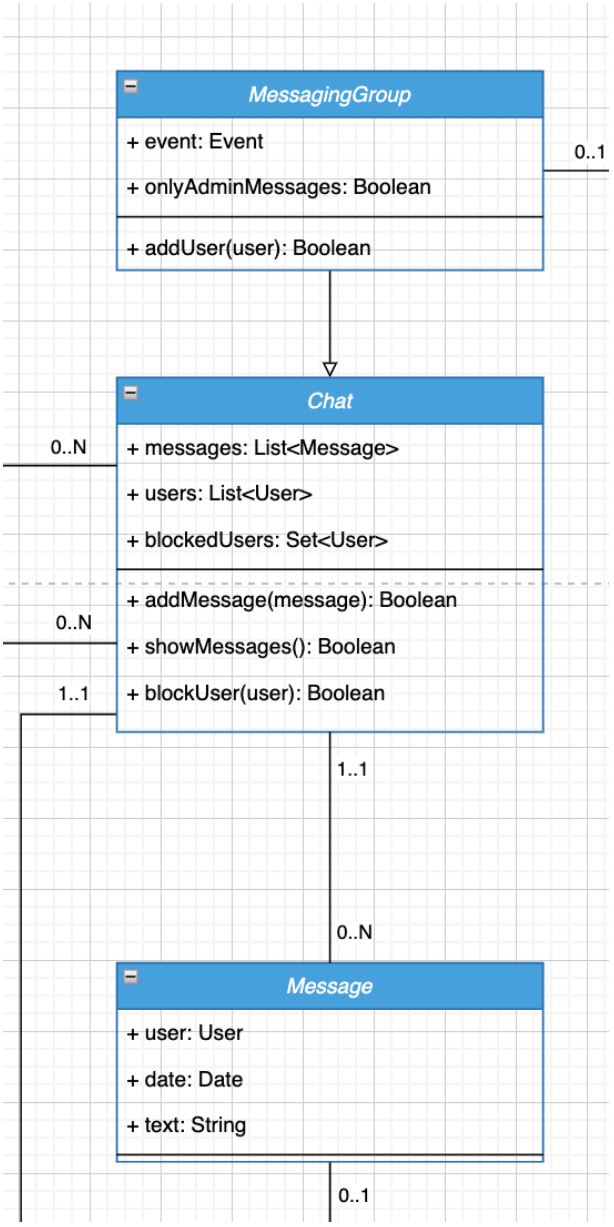
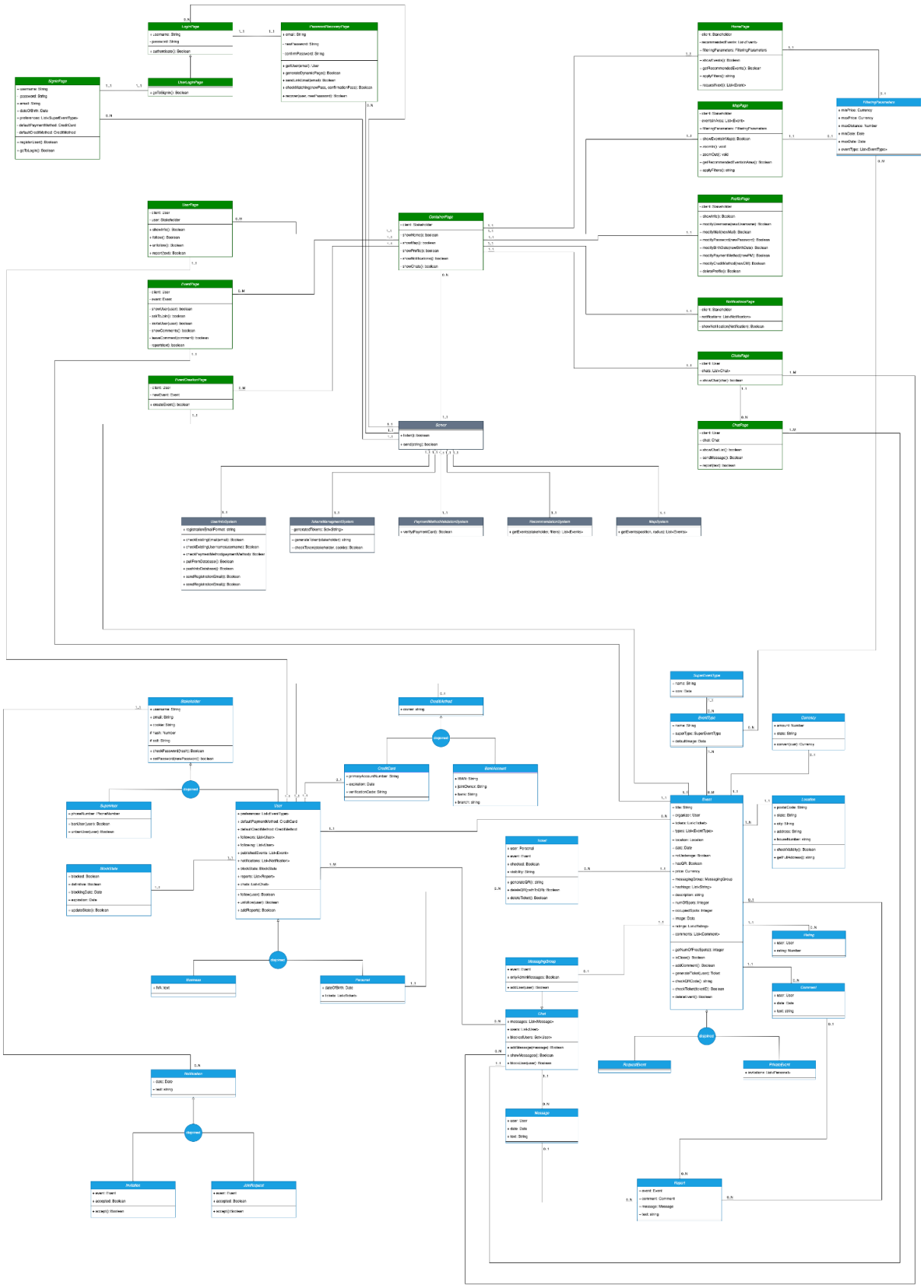


Diagramma delle classi complessivo

Riportiamo di seguito il diagramma delle classi con tutte le classi fino ad ora presentate.



2. Codice in Object Constraint Language

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.

```
Unset

context User
  inv : (self.preferences -> size()) >= 3
  inv : self.followers -> excludes(self)
  inv : self.following -> excludes(self)

context User::follow(user)
  pre: self.following -> excludes(user)
  post: self.following -> includes(user)

context User::unfollow(user)
  pre: self.following -> includes(user)
  post: self.following -> excludes(user)

context Supervisor::banUser(user, definitive, duration)
  pre : (user.BlockState.expired == NULL) ||
    (user.BlockState.expired < currentDate())
  pre : user.BlockState.definitive = False

  post : user.BlockState.definitive = definitive
  post : user.BlockState.blockingDate = currentDate()
  post : user.BlockState.expiration = currentDate() + duration

context Supervisor::unbanUser(user)
  pre : (user.BlockState.expiration != NULL) &&
    (user.BlockState.expiration >= currentDate())
  pre : user.BlockState.definitive = False

  post : user.BlockState.definitive = False
  post : user.BlockState.blockingDate = NULL
  post : user.BlockState.expiration = NULL

context BlockState
  inv : (self.blockingDate == NULL) || (self.blockingDate <= currentDate())
  inv : (self.expiration == NULL) || (self.expiration >= self.blockingDate)

context Event
  inv : self.types -> notEmpty()
```

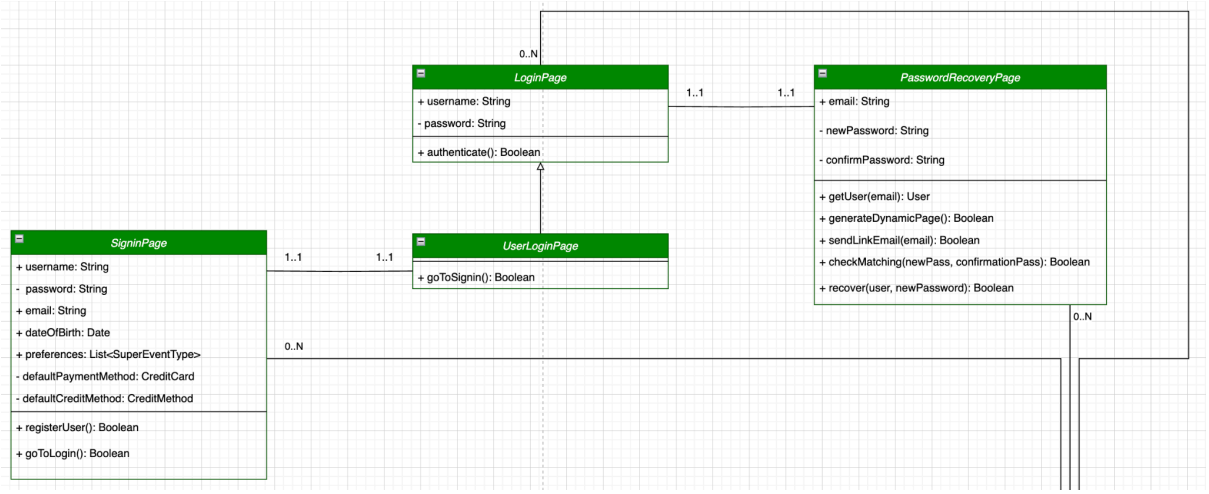
```
  inv : self.price.value >= 0.0
  inv : (self.participants -> size()) <= self.numOfSpots
  inv : self.pathsToHTML -> notEmpty()

context Event::generateTicket(personalUser)
  pre : self.numOfSpots > (self.participants -> size())
  pre : self.participants -> excludes(personalUser)

  post : self.participants -> includes(personalUser)
  post : self.tickets -> notEmpty()
  post : personalUser.tickets -> notEmpty()

context Event::checkQRCode()
  pre : self.hasQRCode = True
  pre : self.participants -> notEmpty()

context Event::checkTicket(ticketID)
  pre : self.tickets -> notEmpty()

context Chat
  inv : self.users -> size() >= 1

context Chat::addMessage(requester, message)
  post : self.messages -> notEmpty()
  post : self.messages -> includes(message)

context Chat::blockUser(requester, target)
  pre : requester.blockedList -> excludes(target)
  post : requester.blockedList -> includes(target)

context MessagingGroup::addUser(user)
  pre : self.users -> excludes(user)
  post : self.users -> includes(user)

context Message:
  inv : self.date <= currentDate()

context JoinRequest
  inv : self.date <= currentDate()
  inv : self.user != self.event.organizer

context JoinRequest::accept()
  pre : self.accepted = False
  post : self.accepted = True

context Invitation
  inv : self.user != self.event.organizer
  inv : self.date <= currentDate()
```

```
context Invitation::accept()
  pre : self.accepted = False
  post : self.accepted = True

context Notification
  inv : self.date <= currentDate()

context SuperNotification
  inv : self.date <= currentDate()

context Comment
  inv : self.date <= currentDate()

context CookiesManagementSystem::generateCookie(stakeholder)
  post : self.generatedCookies -> notEmpty()
  post : self.generatedCookies -> includes(Pair<stakeholder, cookie>)

context SigninPage::registerUser(username, email, password, ...)
  pre : Register.users.usernames -> excludes(username)
  pre : Register.users.emails -> excludes(email)

context Server::waitConnection(port)
  pre : self.availablePorts[port] = True
  post : self.availablePorts[port] = True

context FilteringParameters
  inv : self.minPrice <= self.maxPrice
  inv : self.minDate <= self.maxDate

context UserPage::follow()
  pre : self.client != self.user
  pre : self.client.following -> excludes(user)
  pre : self.user.followers -> excludes(client)
  post : self.client.following -> includes(user)
  post : self.user.followers -> includes(client)

context UserPage::unfollow()
  pre : self.client != self.user
  pre : self.client.following -> includes(user)
  pre : self.user.followers -> includes(client)
  post : self.client.following -> excludes(user)
  post : self.user.followers -> excludes(client)

context EventCreationPage::createEvent()
  pre : self.client.publishedEvents -> excludes(newEvent)
  post : self.client.publishedEvents -> includes(newEvent)
```

3. Diagramma delle classi con codice OCL

Riportiamo infine il diagramma delle classi con tutte le classi fino ad ora presentate ed il codice OCL individuato.

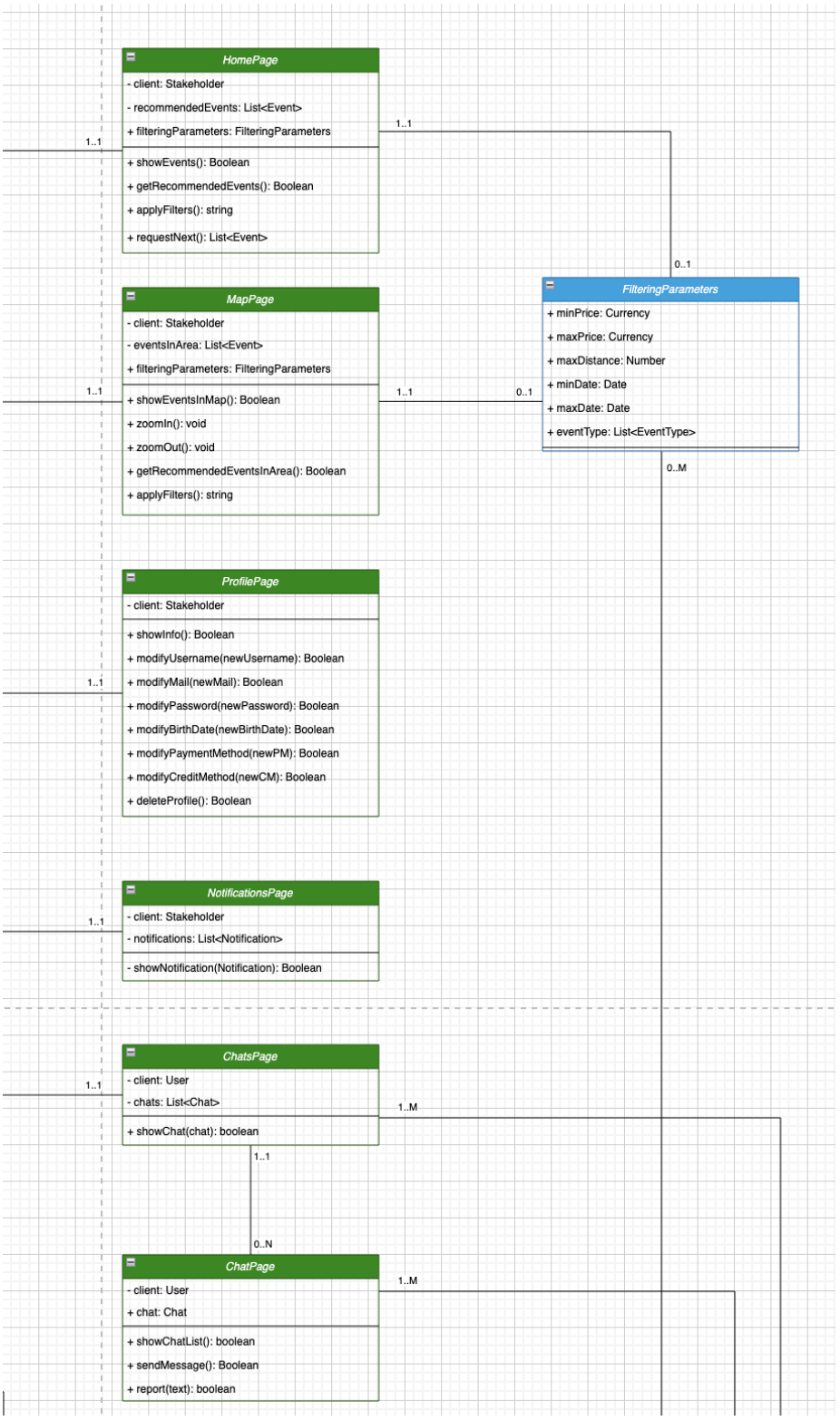
Pagine di Accesso



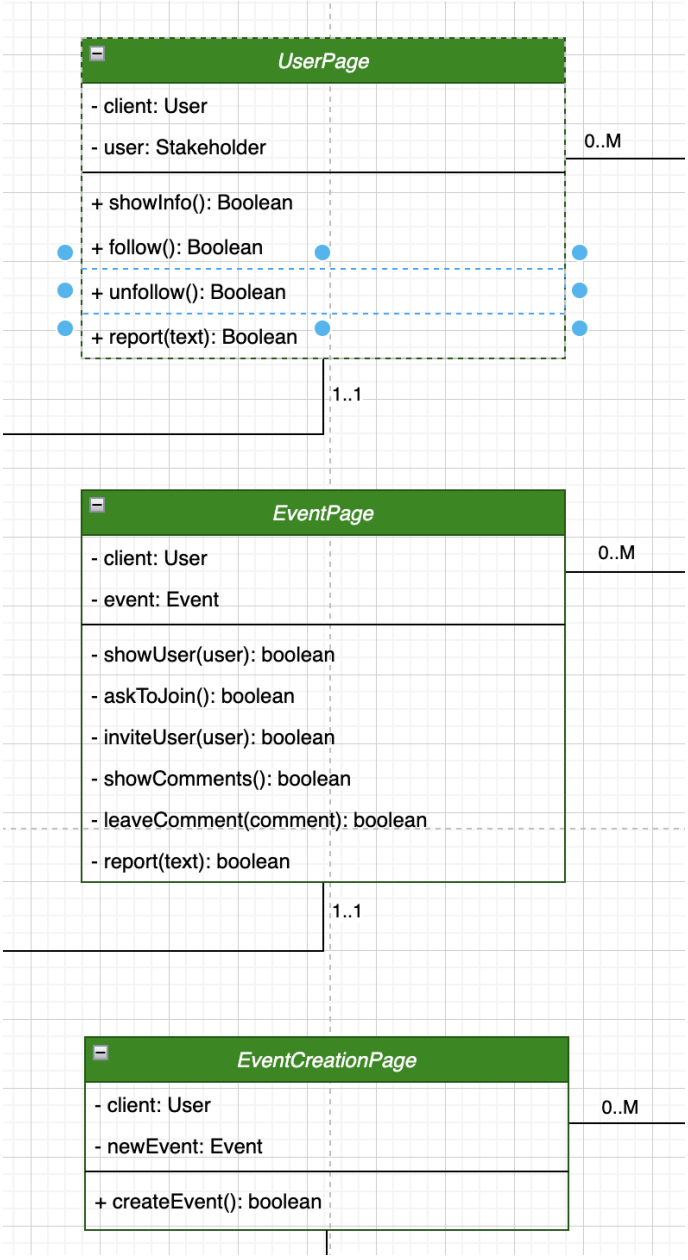
Unset

```
context SigninPage::registerUser(username, email, password, ...)
  pre : Register.users.usernames -> excludes(username)
  pre : Register.users.emails -> excludes(email)
```

Pagine di Navigazione



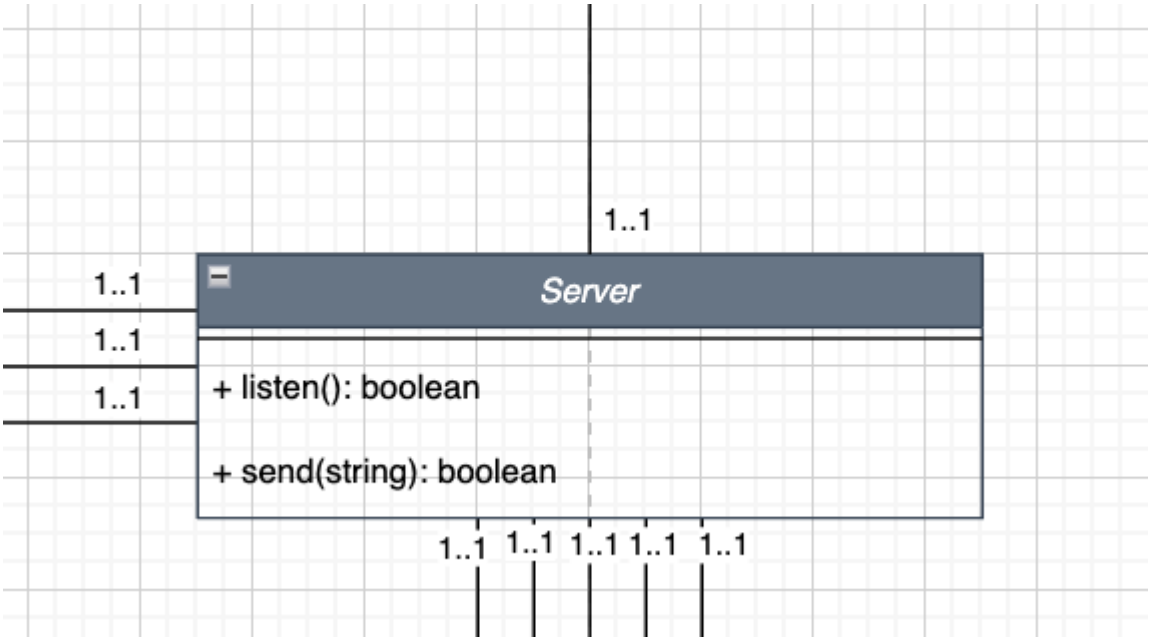
```
Unset
context FilteringParameters
  inv : self.minPrice <= self.maxPrice
  inv : self.minDate <= self.maxDate
```




```
Unset
context UserPage::follow()
  pre : self.client != self.user
  pre : self.client.following -> excludes(user)
  pre : self.user.followers -> excludes(client)
```

```
  post : self.client.following -> includes(user)
  post : self.user.followers -> includes(client)

context UserPage::unfollow()
  pre : self.client != self.user
  pre : self.client.following -> includes(user)
  pre : self.user.followers -> includes(client)
  post : self.client.following -> excludes(user)
  post : self.user.followers -> excludes(client)

context EventCreationPage::createEvent()
  pre : self.client.publishedEvents -> excludes(newEvent)
  post : self.client.publishedEvents -> includes(newEvent)
```

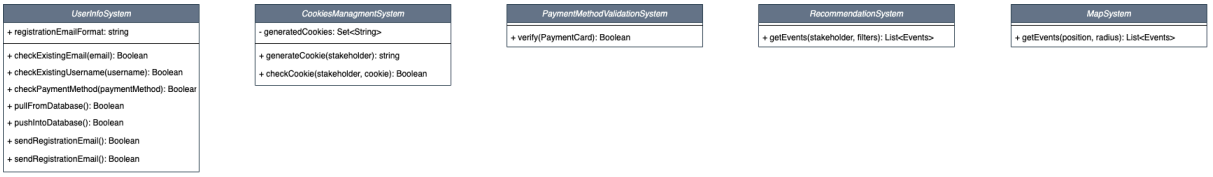
Server



Unset

```
context Server::waitConnection(port)
  pre : self.availablePorts[port] = True
  post : self.availablePorts[port] = True
```

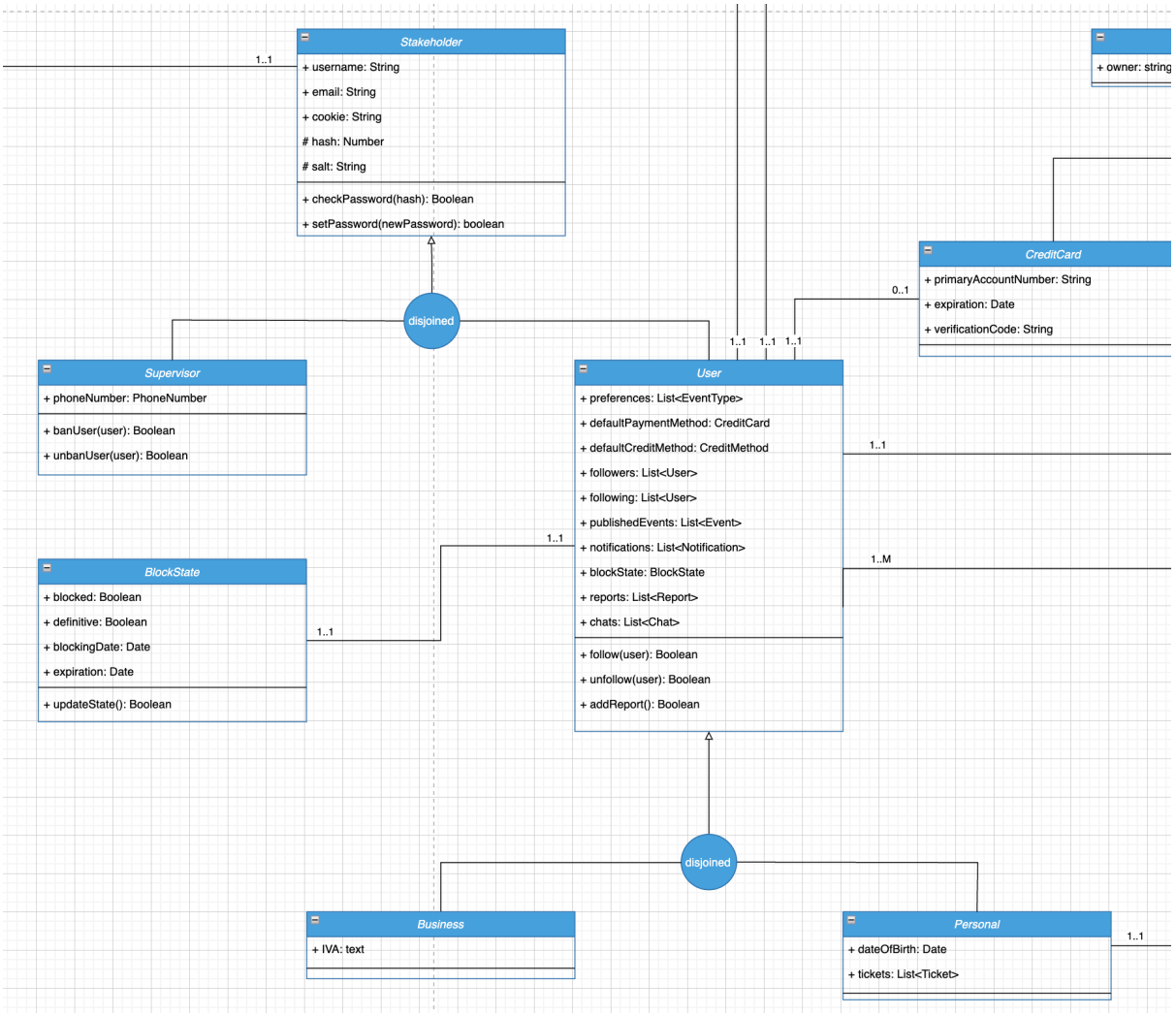
Sistemi di Back-End



Unset

```
context CookiesManagementSystem::generateCookie(stakeholder)
  post : self.generatedCookies -> notEmpty()
  post : self.generatedCookies -> includes(Pair<stakeholder, cookie>)
```

Classi Stakeholder

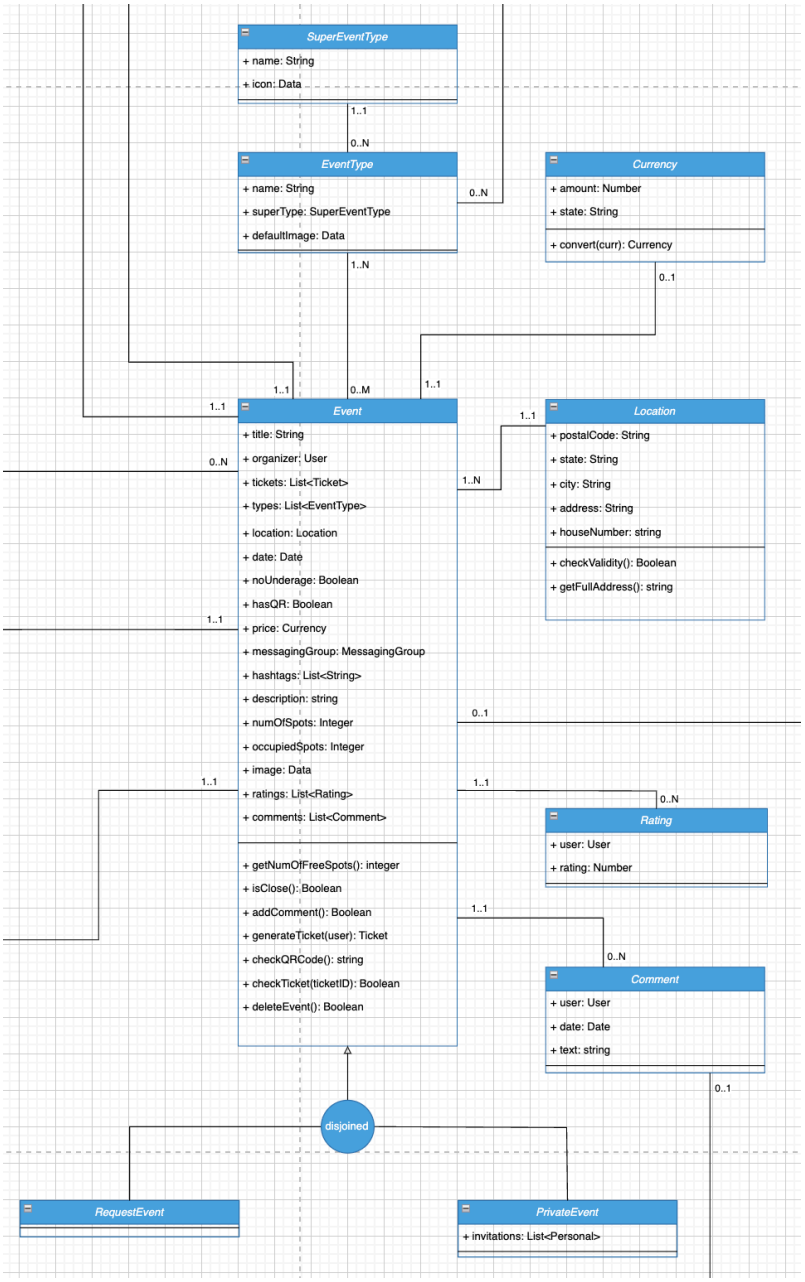


Unset

```
context User
  inv : (self.preferences -> size()) >= 3
  inv : self.followers -> excludes(self)
  inv : self.following -> excludes(self)

context User::follow(user)
  pre: self.following -> excludes(user)
  post: self.following -> includes(user)

context User::unfollow(user)
  pre: self.following -> includes(user)
  post: self.following -> excludes(user)

context Supervisor::banUser(user, definitive, duration)
```

```
  pre : (user.BlockState.expired == NULL) ||
    (user.BlockState.expired < currentDate())
  pre : user.BlockState.definitive = False

  post : user.BlockState.definitive = definitive
  post : user.BlockState.blockingDate = currentDate()
  post : user.BlockState.expiration = currentDate() + duration

context Supervisor::unbanUser(user)
  pre : (user.BlockState.expiration != NULL) &&
    (user.BlockState.expiration >= currentDate())
  pre : user.BlockState.definitive = False

  post : user.BlockState.definitive = False
  post : user.BlockState.blockingDate = NULL
  post : user.BlockState.expiration = NULL

context BlockState
  inv : (self.blockingDate == NULL) || (self.blockingDate <= currentDate())
  inv : (self.expiration == NULL) || (self.expiration >= self.blockingDate)
```

Classi Evento

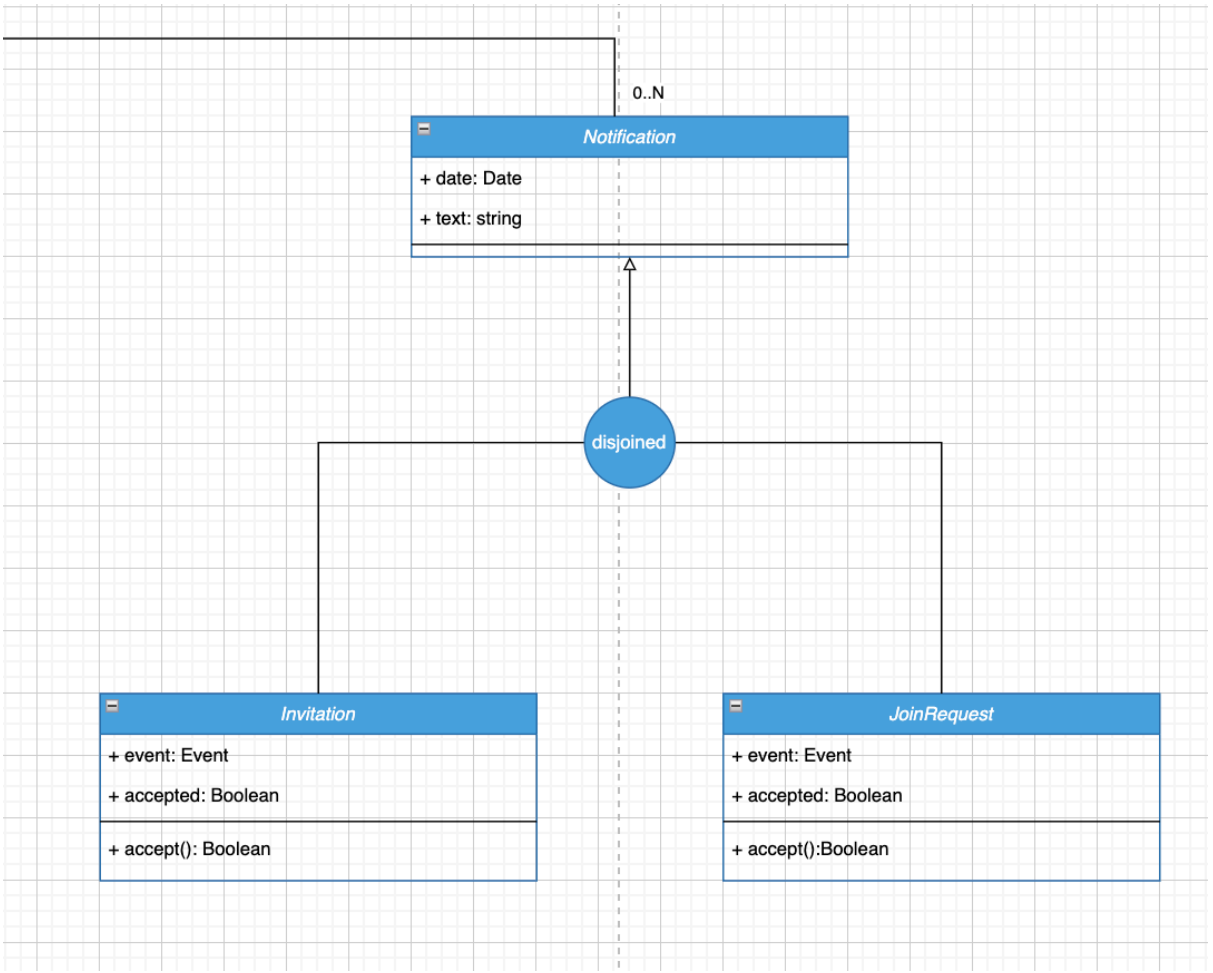


Unset

```
context Event
  inv : self.types -> notEmpty()
  inv : self.price.value >= 0.0
  inv : (self.participants -> size()) <= self.numOfSpots
  inv : self.pathsToHTML -> notEmpty()

context Event::generateTicket(personalUser)
  pre : self.numOfSpots > (self.participants -> size())
  pre : self.participants -> excludes(personalUser)
```

```
  post : self.participants -> includes(personalUser)
  post : self.tickets -> notEmpty()
  post : personalUser.tickets -> notEmpty()

context Event::checkQRCode()
  pre : self.hasQRCode = True
  pre : self.participants -> notEmpty()

context Event::checkTicket(ticketID)
  pre : self.tickets -> notEmpty()

context Comment
  inv : self.date <= currentDate()
```

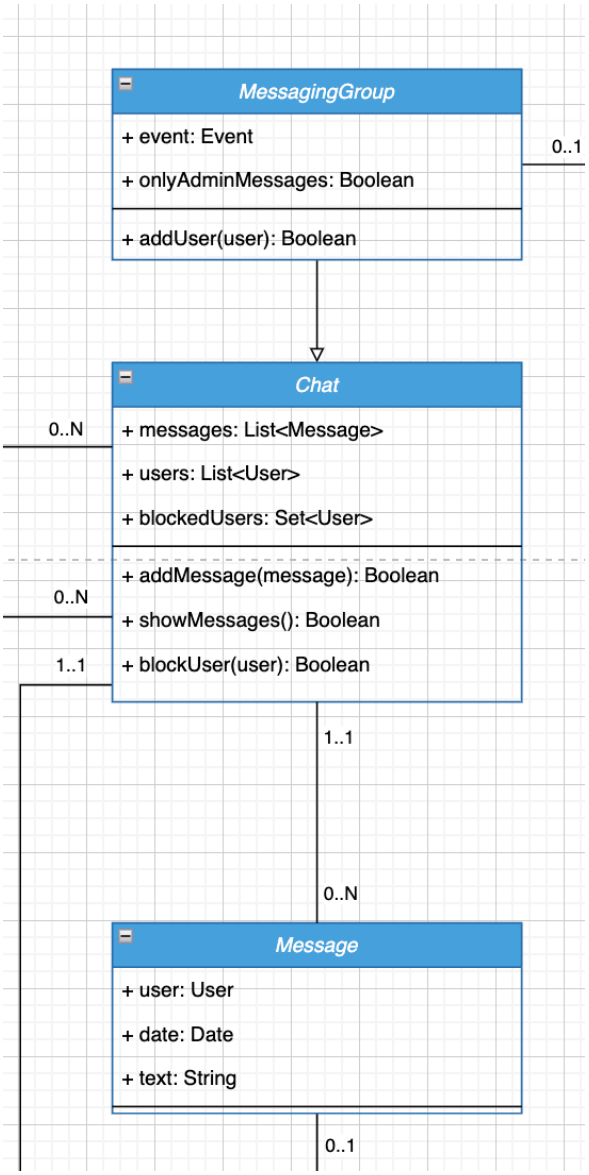
Notifiche



Unset

```
context JoinRequest
  inv : self.date <= currentDate()
  inv : self.user != self.event.organizer

context JoinRequest::accept()
  pre : self.accepted = False
  post : self.accepted = True

context Invitation
  inv : self.user != self.event.organizer
  inv : self.date <= currentDate()

context Invitation::accept()
  pre : self.accepted = False
  post : self.accepted = True

context Notification
  inv : self.date <= currentDate()

context SuperNotification
  inv : self.date <= currentDate()
```

Chat



Unset

```
context Chat
  inv : self.users -> size() >= 1

context Chat::addMessage(requester, message)
  post : self.messages -> notEmpty()
  post : self.messages -> includes(message)

context Chat::blockUser(requester, target)
  pre : requester.blockedList -> excludes(target)
  post : requester.blockedList -> includes(target)
```

```
context MessagingGroup::addUser(user)
  pre : self.users -> excludes(user)
  post : self.users -> includes(user)

context Message:
  inv : self.date <= currentDate()
```