

Ingegneria del Software

UNITN - Dipartimento di Ingegneria e Scienza dell'Informazione
Anno Accademico 2022-2023
Battocchio N. - Bocchi G. - Miazzo L.



**UNIVERSITY
OF TRENTO**

CarMeeTiN

— Documento di architettura



Versione del documento

Versione	Data	Modifiche
0.1	30/11/2022	Creazione documento e aggiunta diagramma delle classi
0.2	3/12/2022	Aggiunta paragrafo OCL
0.3	4/12/2022	Correzione diagramma delle classi
1.0	6/12/2022	Aggiunta diagramma complessivo e correzioni minori

INDICE

SCOPO DEL DOCUMENTO	5
<hr/>	
1. DIAGRAMMA DELLE CLASSI	6
1.1 Utenti e sistemi esterni	7
1.1.1 UtenteGenerico	7
1.1.2 UtenteAnonimo	7
1.1.3 UtenteAutenticato	7
1.2 GestoreAccount	9
1.2.1 AuthHelper	9
1.2.2 UtenteRecuperoPassword	10
1.2.3 Generatore QR	11
1.3 Mapbox API	12
1.3.1 Posizione	12
1.4 Interfaccia database	12
1.4.1 DatabaseHelper	12
1.5 Gestore eventi	14
1.5.1 Evento	14
1.5.2 StatoEvento	15
1.6 Gestore garage	15
1.6.1 TipoAuto	15
1.6.2 Auto	16
1.7 Pagine Principali del Sito Web	17
1.7.1 Home	17
1.7.2 Evento	17
1.7.3 Profilo Utente	18
1.8 Diagramma delle classi completo	19
<hr/>	
2. CODICE IN OBJECT CONSTRAINT LANGUAGE	20
2.1 Stato Evento	20
2.2 Partecipanti Evento	20
2.3 Crea Evento	21
2.4 Iscrizione Evento	21
2.5 Disiscrizione Evento	21
2.6 Aggiorna Evento	22
2.7 Posizione	22
2.8 TipoAuto	22
2.9 Auto	23

2.10 Crea Auto	23
2.11 Aggiorna Auto	23
2.12 Rimuovi Auto	24
2.13 DatabaseHelper	24
2.14 EmailHelper	24
2.15 AuthHelper	25
2.16 Utente Recupera Password	25
2.17 Utente Anonimo	25
2.18 Login	26
2.19 Utente Autenticato	26
2.20 Aggiorna Utente	27
2.21 Aggiorna Password	27
2.22 Logout	27

3. DIAGRAMMA DELLE CLASSI CON CODICE OCL	28
---	-----------

SCOPO DEL DOCUMENTO

Il presente documento riporta la definizione dell'architettura del progetto 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 CarMeeTiN.

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.

1.1 Utenti e sistemi esterni

Analizzando il diagramma di contesto realizzato nel documento di Specifica dei Requisiti si nota la presenza di due attori: “**Utente anonimo**” (RF 1) e “**Utente autenticato**” (RF 2).

L'attore “**Utente anonimo**” è colui che utilizza il sito web per visualizzare gli eventi esistenti e le pagine profilo degli utenti già registrati.

Una volta autenticato diventa “**Utente autenticato**” e può modificare il proprio account, creare eventi, iscriversi a eventi esistenti e modificare il proprio garage.

Da questi sono state individuate 3 classi:

- **1.1.1 UtenteGenerico**

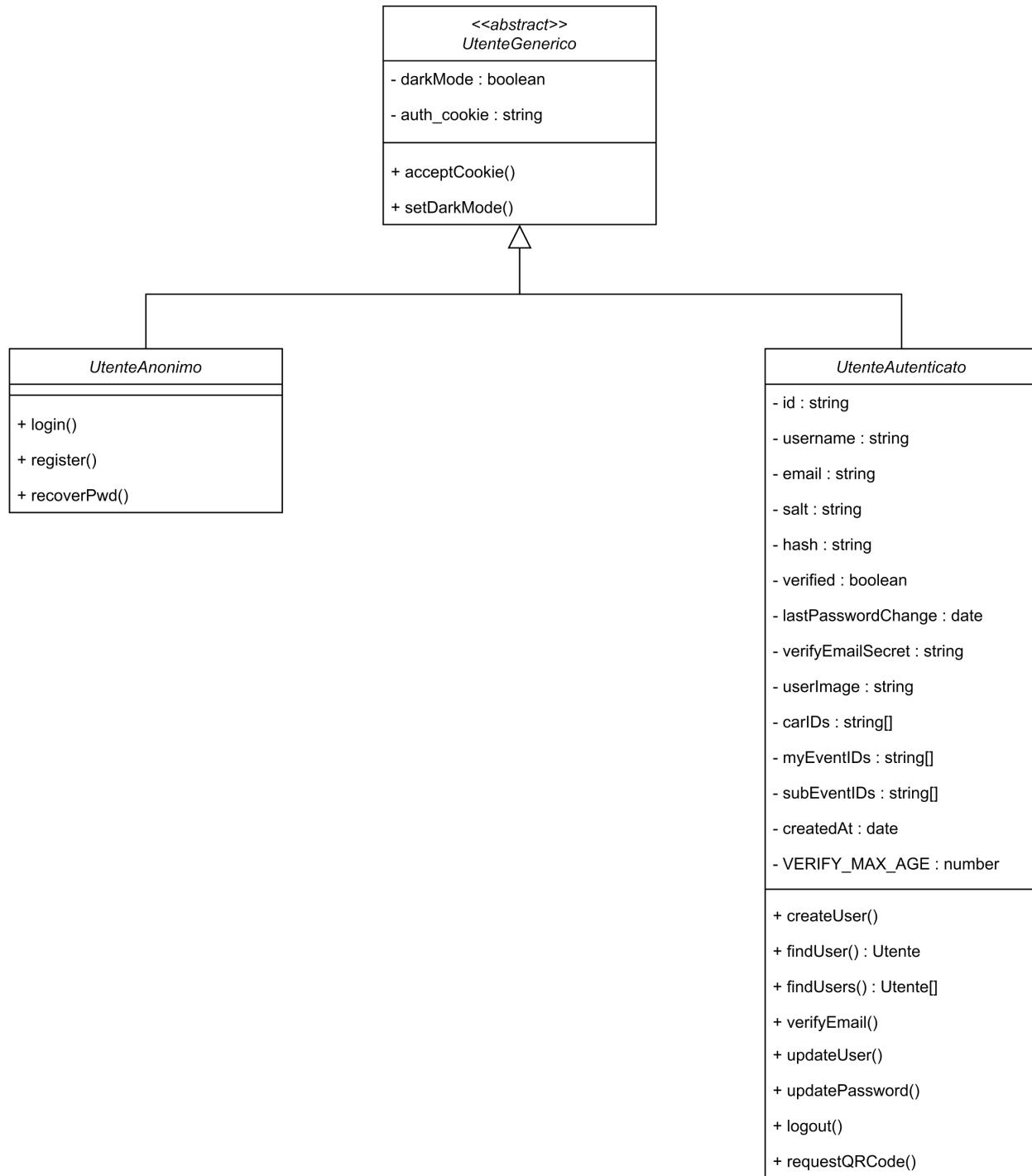
Questa è una classe astratta e indica un qualsiasi utente.

- **1.1.2 UtenteAnonimo**

Questa classe è una specializzazione di **UtenteGenerico** e rappresenta un utente non autenticato che utilizza il sito.

- **1.1.3 UtenteAutenticato**

Questa classe è una specializzazione di **UtenteGenerico** e rappresenta un utente che si è autenticato sul sito tramite la procedura di login.

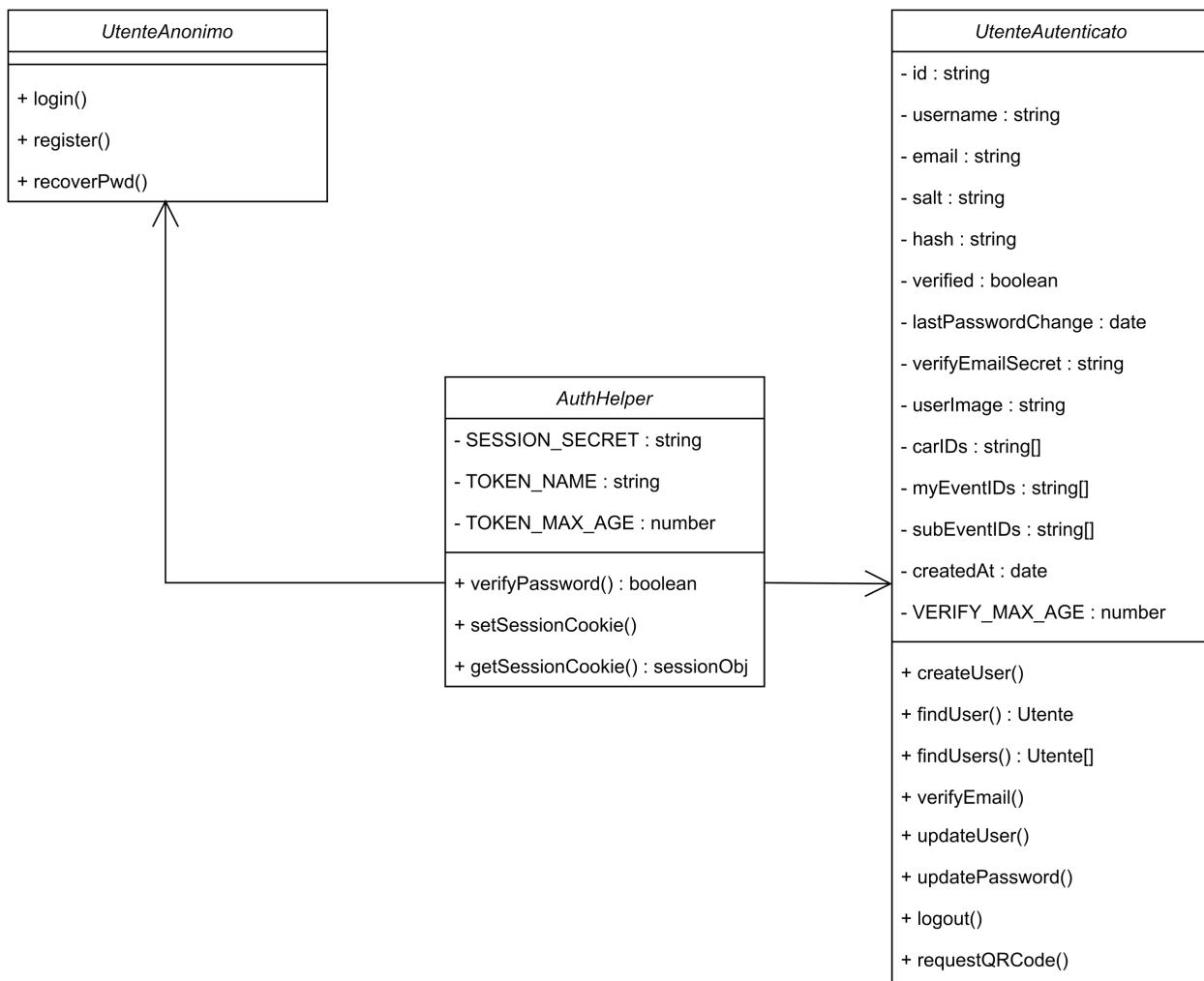


Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AA](#).

1.2 GestoreAccount

- **1.2.1 AuthHelper**

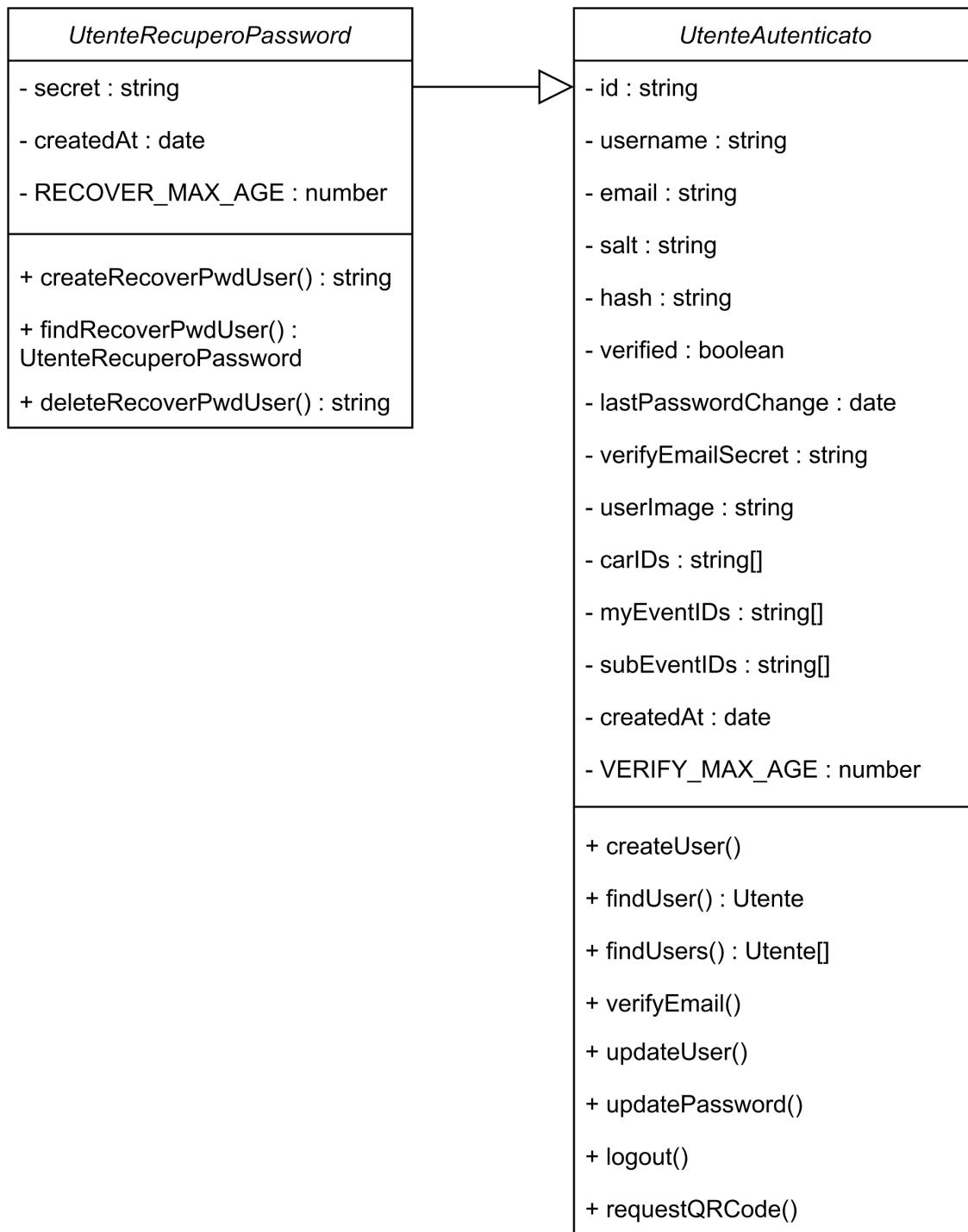
Questa classe ha il compito di fornire le autorizzazioni necessarie a un **UtenteAnonimo** per diventare un **UtenteAutenticato**. Nello specifico ha il compito di validare o meno i dati che gli verranno passati dall'applicazione CarMeeTiN e di notificare l'esito di tale operazione.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AB](#).

- **1.2.2 UtenteRecuperoPassword**

Questa classe ha il compito di permettere a un **UtenteAnonimo** di recuperare la password in caso l'abbia dimenticata.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AC](#).

- **1.2.3 Generatore QR**

Questa è la classe che permette ad un **UtenteAutenticato** di generare il codice QR del proprio profilo.

<i>UtenteAutenticato</i>	
- id : string	
- username : string	
- email : string	
- salt : string	
- hash : string	
- verified : boolean	
- lastPasswordChange : date	
- verifyEmailSecret : string	
- userImage : string	
- carIDs : string[]	
- myEventIDs : string[]	
- subEventIDs : string[]	
- createdAt : date	
- VERIFY_MAX_AGE : number	
+ createUser()	
+ findUser() : Utente	
+ findUsers() : Utente[]	
+ verifyEmail()	
+ updateUser()	
+ updatePassword()	
+ logout()	
+ requestQRCode()	

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AD](#).

1.3 Mapbox API

- **1.3.1 Posizione**

Questa classe contiene le informazioni necessarie a identificare un certo luogo geografico, e i metodi necessari per impostare queste informazioni e richiedere il luogo in linguaggio naturale (ad esempio dalle coordinate a “Romagnano, Trento, TN”).

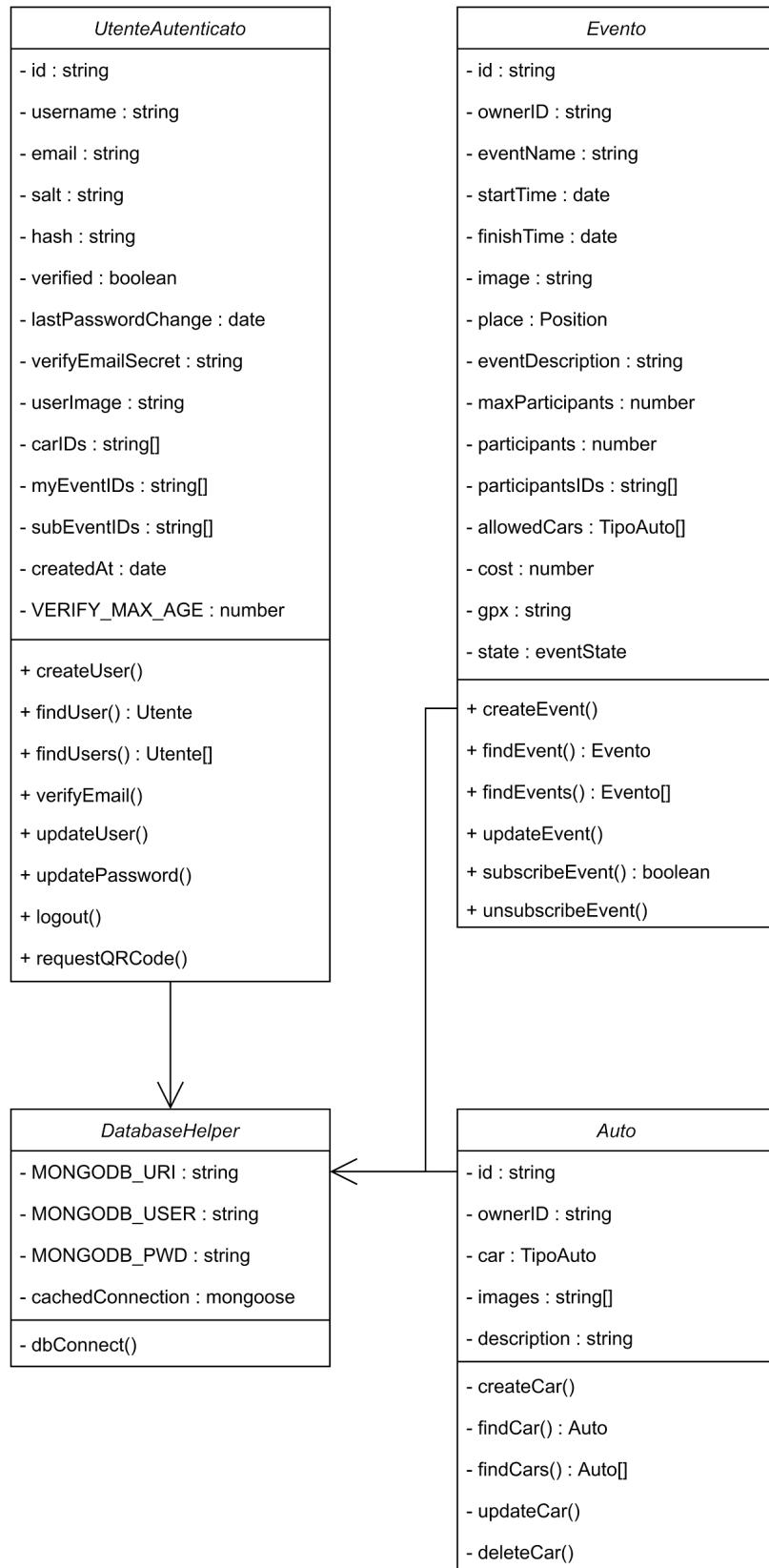
<i>Posizione</i>
- latitude : real
- longitude : real
+ getNameOfPosition() : string
+ setPositionFromName()

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AE](#).

1.4 Interfaccia database

- **1.4.1 DatabaseHelper**

Questa classe contiene i metodi necessari per effettuare le operazioni di lettura e scrittura sul database.

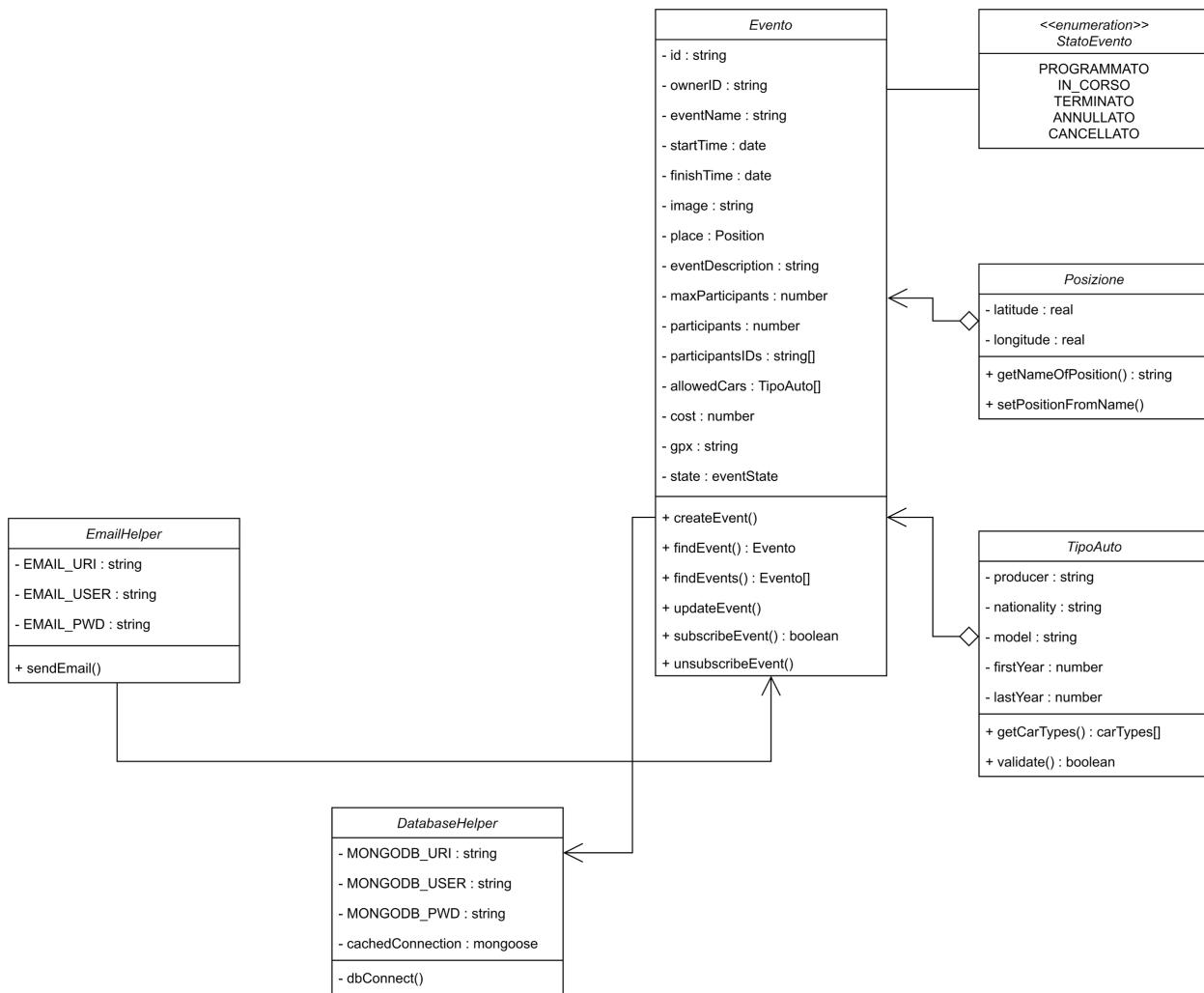


Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AF](#).

1.5 Gestore eventi

- **1.5.1 Evento**

Questa classe contiene le informazioni necessarie a identificare un certo evento.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AG](#).

- **1.5.2 StatoEvento**

Questa è una *enumeration* che contiene i possibili stati di un evento.

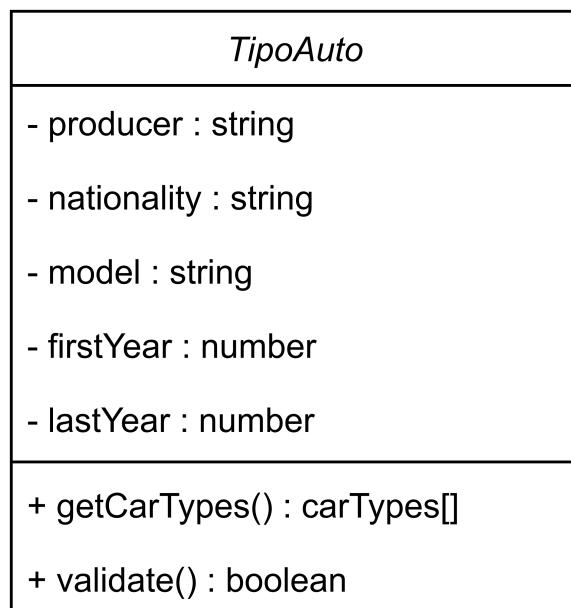


Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AH](#).

1.6 Gestore garage

- **1.6.1 TipoAuto**

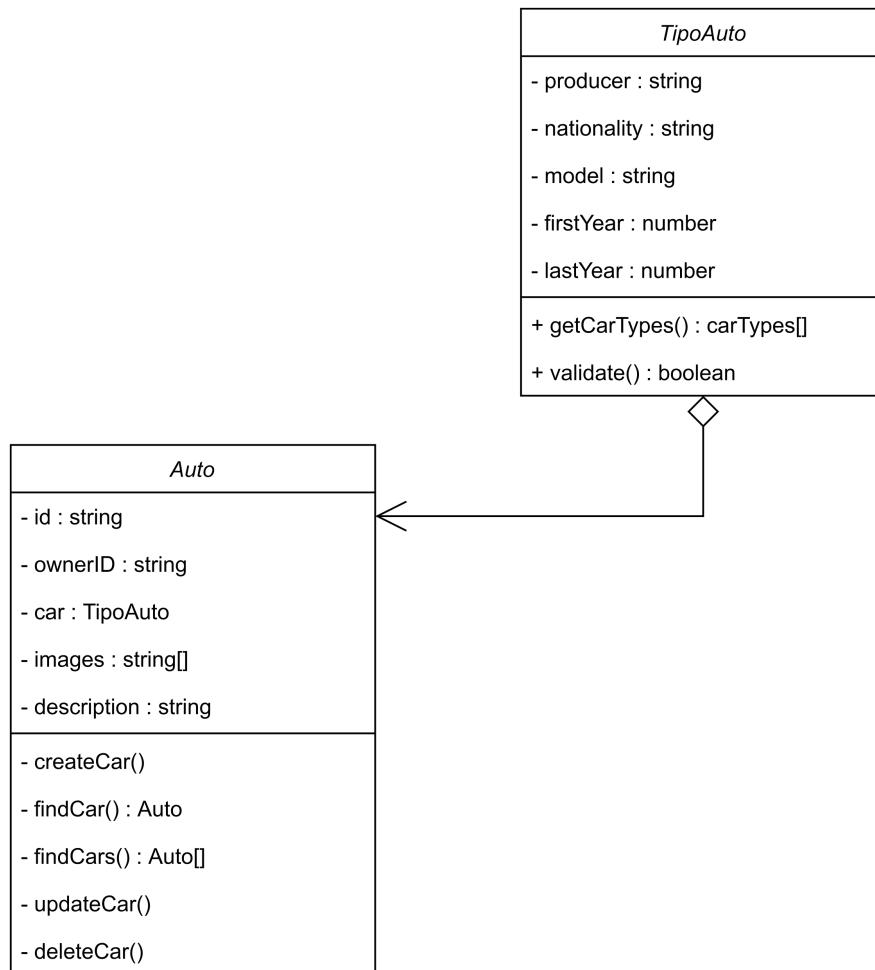
Questa classe contiene le informazioni necessarie a identificare un generico modello di auto di un certo produttore.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AI](#).

- **1.6.2 Auto**

Queste classi contengono le informazioni necessarie a identificare una precisa auto dell'utente che la possiede.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AJ](#).

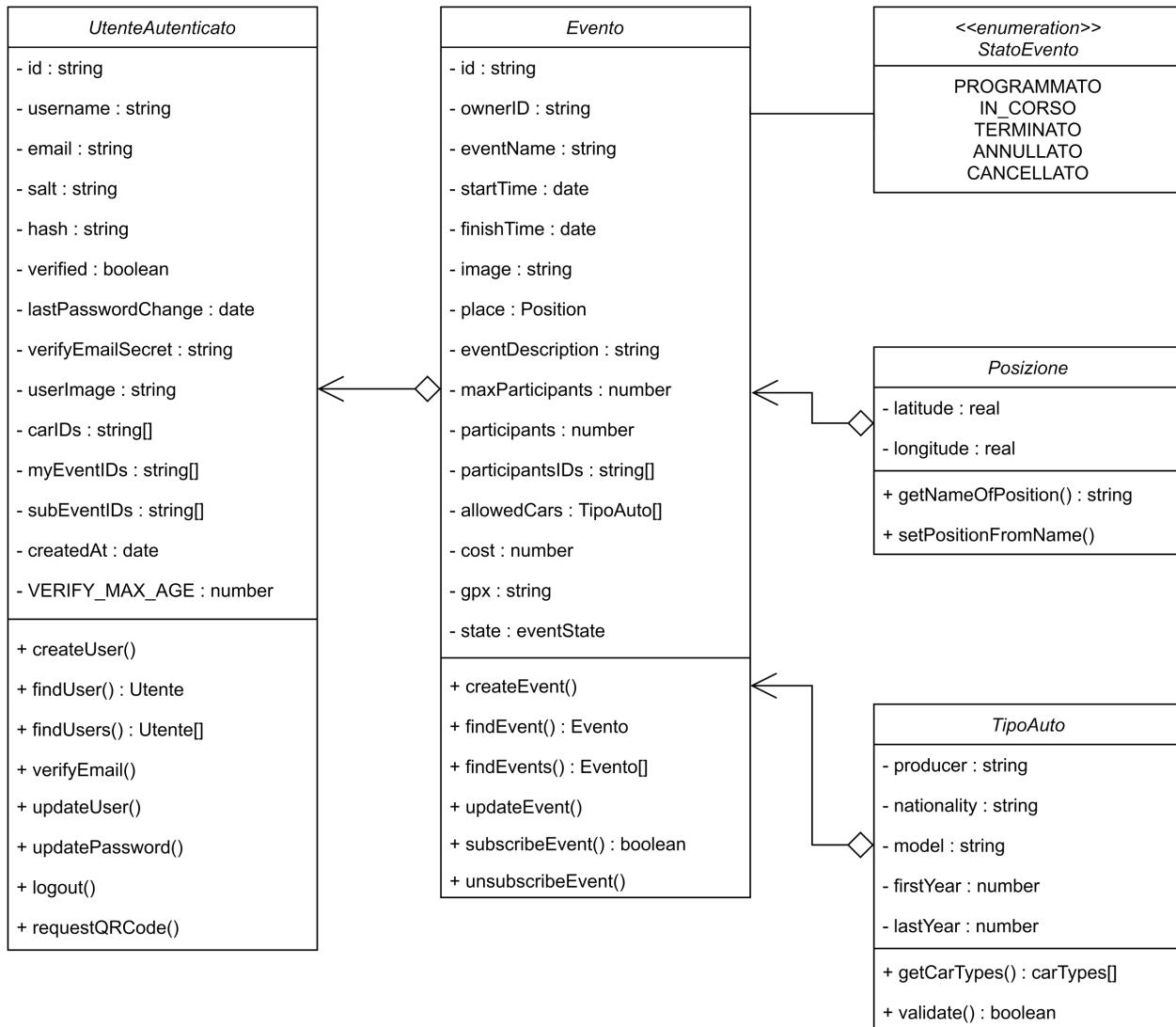
1.7 Pagine Principali del Sito Web

- **1.7.1 Home**

Queste classi contengono le informazioni e i metodi necessari per la generazione della pagina “Home”.

- **1.7.2 Evento**

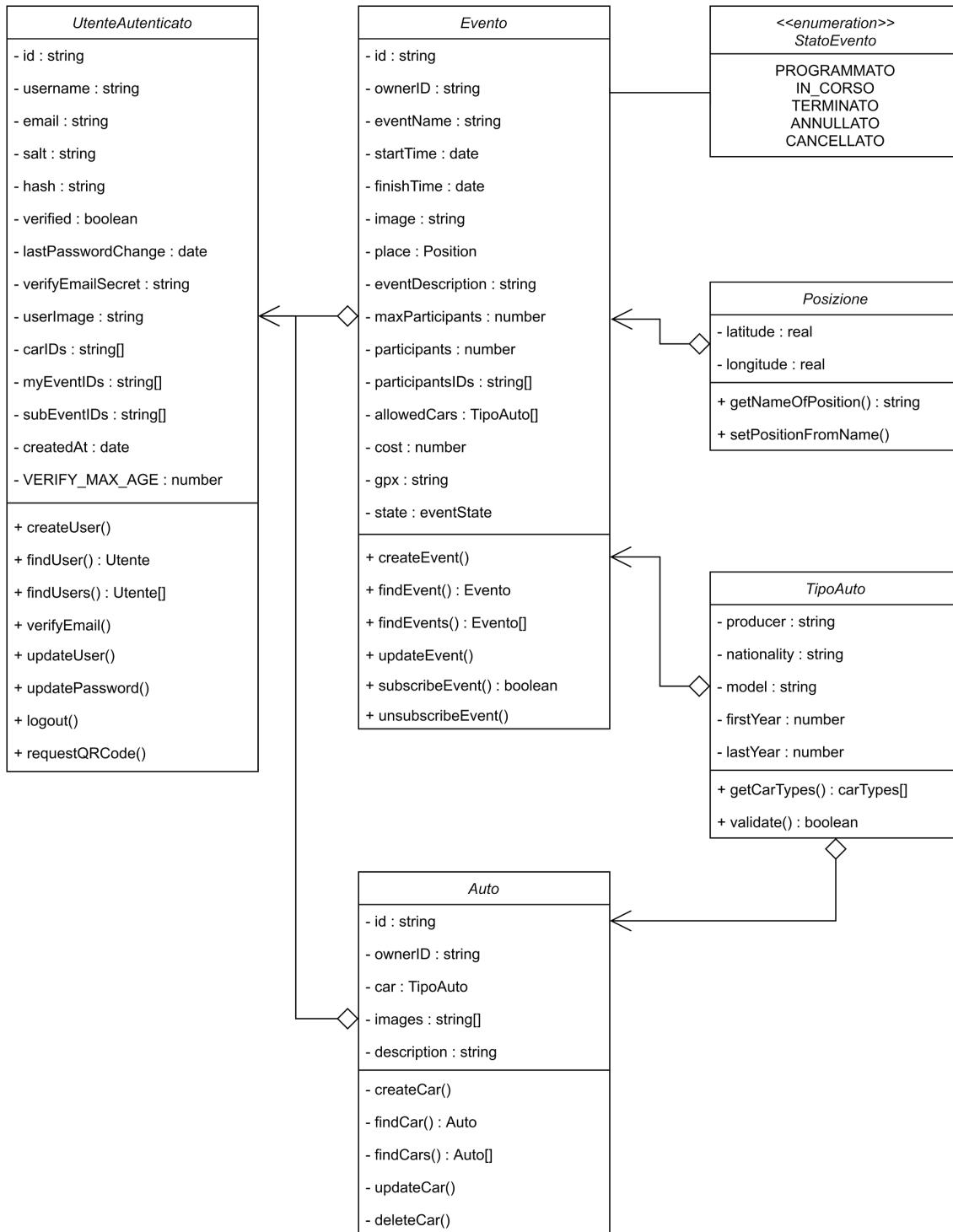
Queste classi contengono le informazioni e i metodi necessari per la generazione della pagina web di un evento.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AK](#).

1.7.3 Profilo Utente

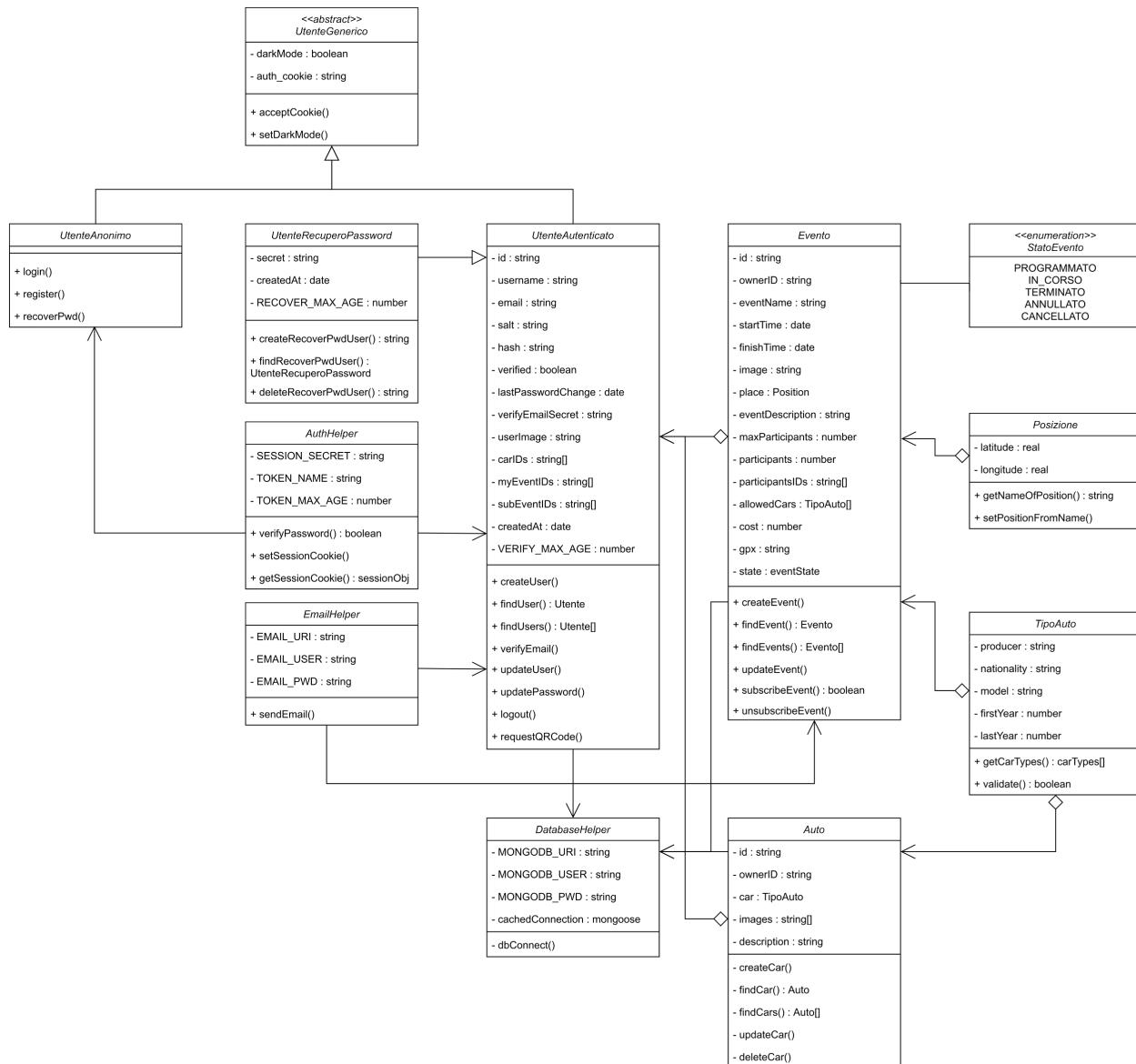
Queste classi contengono le informazioni e i metodi necessari per la generazione della pagina web di un utente.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato AL](#).

1.8 Diagramma delle classi completo

Viene di seguito mostrato il diagramma delle classi completo.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato B](#).

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.

2.1 Stato Evento

Ogni evento della classe **Evento** deve essere sempre in uno stato valido contenuto nella enumerazione **StatoEvento**:

```
context Evento inv:  
  
    (self.startTime > Date.now() and  
     self.state = StatoEvento.PROGRAMMATO)  
    or  
    (self.startTime < Date.now() and self.finishTime > Date.now() and  
     self.state = StatoEvento.IN_CORSO)  
    or  
    (self.finishTime < Date.now() and  
     self.state = StatoEvento.TERMINATO)  
    or  
    self.state = StatoEvento.ANNULLATO  
    or  
    self.state = StatoEvento.CANCELLATO
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CA](#).

2.2 Partecipanti Evento

Ogni evento della classe **Evento** deve aver sempre il numero di partecipanti minore o uguale al numero massimo di partecipanti e maggiore di zero:

```
context Evento inv:  
  
    self.participants <= self.maxParticipants  
    and  
    self.participants > 0
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CB](#).

2.3 Crea Evento

Per poter creare un evento, l'utente che lo richiede deve essere verificato:

```
context Evento::createEvent()
```

pre:

```
req.user.verified
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CC](#).

2.4 Iscrizione Evento

Per potersi iscrivere ad un evento, il numero di partecipanti deve essere minore rispetto al numero massimo di partecipanti dell'evento, l'utente che richiede l'iscrizione deve essere verificato, e l'evento deve essere nello stato di PROGRAMMATO:

```
context Evento::subscribeEvent()
```

pre:

```
self.participants < self.maxParticipants  
and  
req.user.verified  
and  
self.state = StatoEvento.PROGRAMMATO  
and not  
self.participantsIDs.find(req.user)
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CD](#).

2.5 Disiscrizione Evento

Per poter disiscriversi da un evento, esso deve essere nello stato di PROGRAMMATO:

```
context Evento::unsubscribeEvent()
```

pre:

```
self.state = StatoEvento.PROGRAMMATO  
and  
self.participantsIDs.find(req.user)
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CE](#).

2.6 Aggiorna Evento

Per poter modificare un evento, l'utente che lo richiede deve essere il proprietario del suddetto evento:

```
context Evento::updateEvent()
```

pre:

```
req.user = self.ownerID
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CF](#).

2.7 Posizione

Un oggetto **Posizione** è considerato valido, se le coordinate sono diverse dalla coppia “0.0, 0.0” (si noti che, seppur esista il punto “0.0, 0.0”, esso si trova nel mare, posto poco probabile per accogliere un evento di carmeeting):

```
context Posizione inv:
```

```
self.latitude <> 0 and self.longitude <> 0
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CG](#).

2.8 TipoAuto

Un oggetto **TipoAuto**, per essere considerato valido, deve rispettare il seguente vincolo:

```
context TipoAuto inv:
```

```
self.firstYear <= self.lastYear
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CH](#).

2.9 Auto

Un oggetto **Auto** esiste solo se è una configurazione valida specificata nella classe **TipoAuto**:

```
context Auto inv:  
    TipoAuto::validate(self.car)
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CI](#).

2.10 Crea Auto

Ad un utente è concesso creare un'auto solamente se il modello è selezionabile dalla lista predefinita:

```
context Auto inv:  
    TipoAuto::validate(self.car)
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CJ](#).

2.11 Aggiorna Auto

Ad un utente è concesso modificare un'auto solamente se è il vero proprietario dell'auto:

```
context Auto::updateCar()  
  
pre:  
    req.user = self.ownerID
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CK](#).

2.12 Rimuovi Auto

Ad un utente è concesso eliminare un'auto solamente se è il vero proprietario dell'auto:

```
context Auto::deleteCar()  
  
pre:  
    req.user = self.ownerID
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CL](#).

2.13 DatabaseHelper

La configurazione della classe **DatabaseHelper** è considerata valida se e solo se le informazioni riguardanti l'accesso al sistema sono state impostate:

```
context DatabaseHelper inv:  
  
    self.MONGODB_URI <> NULL  
        and  
    self.MONGODB_USER <> NULL  
        and  
    self.MONGO_PWD <> NULL
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CM](#).

2.14 EmailHelper

La configurazione della classe **EmailHelper** è considerata valida se e solo se le informazioni riguardanti l'accesso al sistema sono state impostate:

```
context EmailHelper inv:  
  
    self.EMAIL_URI <> NULL  
        and  
    self.EMAIL_USER <> NULL  
        and  
    self.EMAIL_PWD <> NULL
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CN](#).

2.15 AuthHelper

La configurazione della classe **AuthHelper** è considerata valida se e solo se le informazioni riguardanti il corretto funzionamento del sistema sono state impostate:

```
context AuthHelper inv:  
  
    self.SESSION_SECRET <>> NULL  
        and  
    self.TOKEN_NAME <>> NULL
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CO](#).

2.16 Utente Recupera Password

L'oggetto **UtenteRecuperoPassword** deve solamente esistere per un tempo prestabilito dalla sua creazione (questo sarà gestito automaticamente con gli indici di MongoDB):

```
context  
UtenteRecuperoPassword inv:  
  
    self.createdAt +  
    self.RECOVER_MAX_AGE > Date.now()
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CP](#).

2.17 Utente Anonimo

Un utente è considerato **UtenteAnonimo** se e solo se il cookie di sessione è pari a NULL:

```
context UtenteAnonimo inv:  
  
    self.auth_cookie = NULL
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CQ](#).

2.18 Login

Dopo aver effettuato l'accesso, deve essere impostato il cookie di sessione:

```
context UtenteAnonimo::login():
```

```
post:
```

```
    self.auth_cookie <> NULL
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CR](#).

2.19 Utente Autenticato

L'oggetto **UtenteAutenticato** deve solamente esistere per un tempo prestabilito dalla sua creazione se non viene verificato (questo sarà gestito automaticamente con gli indici di MongoDB). Inoltre l'utente è considerato **UtenteAutenticato** se e solo se il cookie di sessione è stato impostato:

```
context UtenteAutenticato inv:
```

```
    (self.verified or self.createdAt +  
     self.VERIFY_MAX_AGE > Date.now())
```

```
    and
```

```
    self.auth_cookie <> NULL
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CS](#).

2.20 Aggiorna Utente

Ad un **UtenteAutenticato** è concesso aggiornare l'account, solamente se è il proprietario dell'account che cerca di aggiornare, ed ha portato a termine il processo di verifica dell'indirizzo di posta elettronica:

```
context
UtenteAutenticato::updateUser()

pre:
    req.user = self.id
        and
    req.user.verified
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CT](#).

2.21 Aggiorna Password

Ad un **UtenteAutenticato** è concesso aggiornare la propria password, solamente se è il proprietario dell'account che cerca di aggiornare, ed ha portato a termine il processo di verifica dell'indirizzo di posta elettronica:

```
context
UtenteAutenticato::updatePassword()

pre:
    req.user = self.id
        and
    self.verified
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CU](#).

2.22 Logout

Dopo aver effettuato il logout, il cookie di sessione deve essere rimosso:

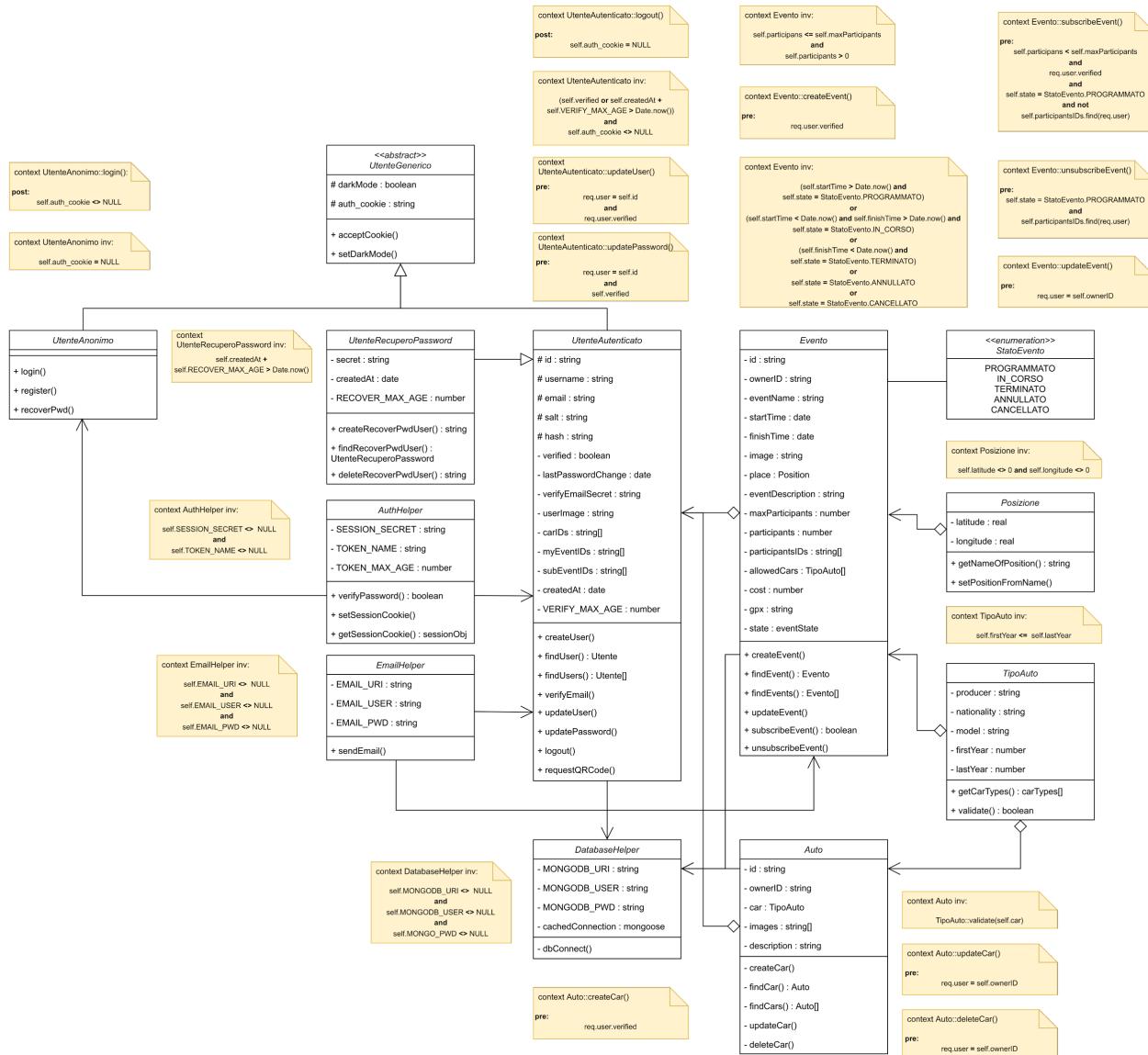
```
context UtenteAutenticato::logout()

post:
    self.auth_cookie = NULL
```

Per la visione di questa immagine ad alta risoluzione si veda l'[allegato CV](#).

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.



Per la visione di questa immagine ad alta risoluzione si veda l'[allegato D](#).