DELIVERABLE D2

Gruppo 5:

➔ Nicola Panozzo 235517

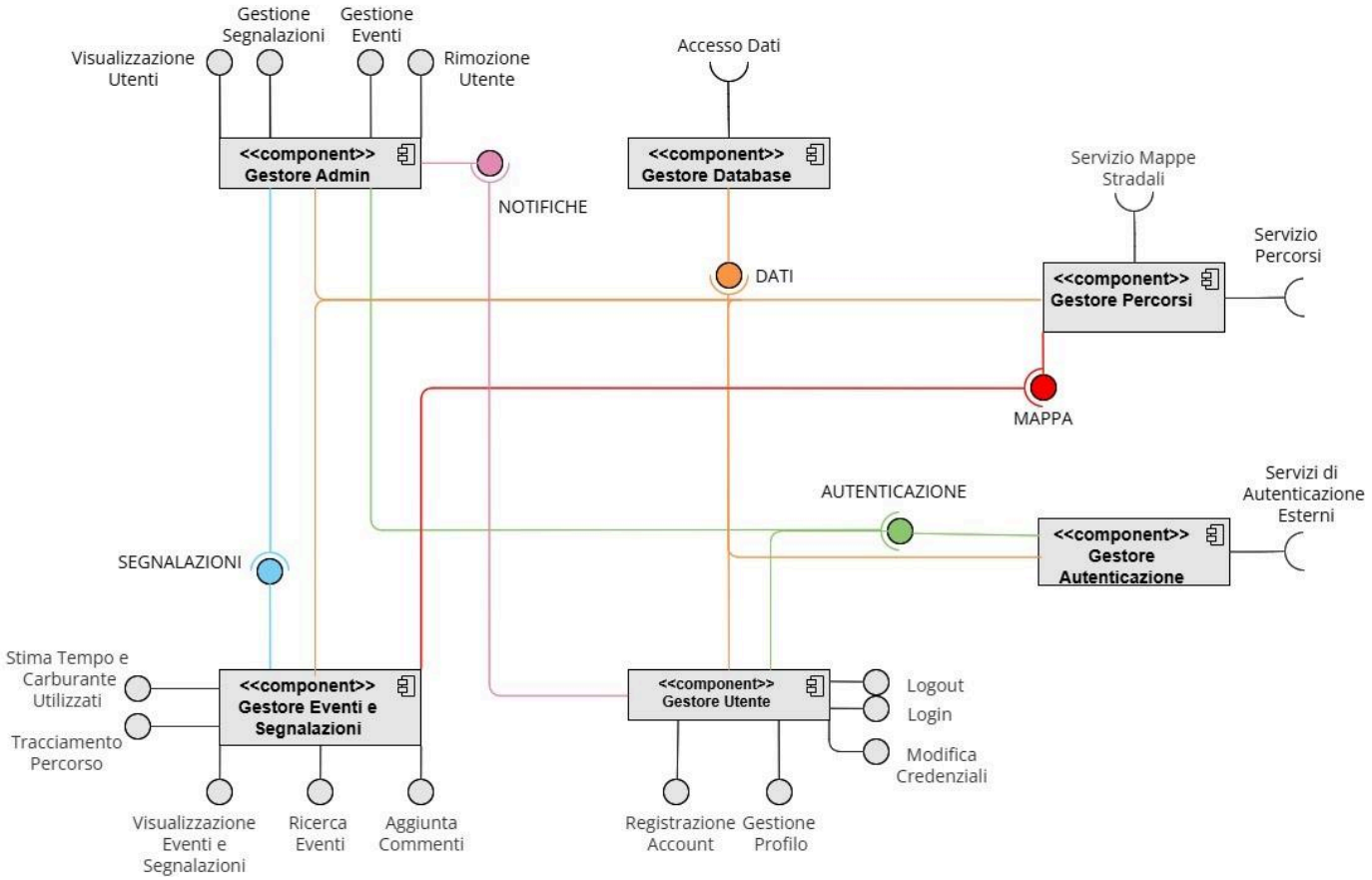
➔ Tommaso Agosti 235053

➔ Valerio Cassone 236359

PAVED WAY

1 Diagramma componenti

1.1 Diagramma



1.2 Descrizione dei Componenti

1.2.1 Gestore Database

Questo componente permette l'accesso ai dati da parte degli altri componenti.

Tipologia	Nome	Descrizione
Fornita	Dati	Il componente fornisce un'interfaccia che permette l'accesso ai dati salvati nel database.
Richiesta	Accesso Dati	Il componente necessita di un'interfaccia fornita dal database per accedere ai dati in esso salvati.

1.2.2 Gestore Percorsi

Questo componente gestisce la mappatura del territorio e il calcolo dei percorsi.

Tipologia	Nome	Descrizione
Fornita	Mappa	Il componente fornisce un'interfaccia che permette la visualizzazione di eventi e segnalazioni sulla mappa del territorio.
Richiesta	Servizio Mappe Stradali	Il componente necessita di un'interfaccia per la mappatura del territorio.
Richiesta	Servizio Percorsi	Il componente necessita di un'interfaccia che permetta il calcolo di percorsi sul territorio.
Richiesta	Dati	Il componente utilizza l'interfaccia "Dati" fornita dal Gestore Database per ottenere dati relativi a eventi e segnalazioni.

1.2.3 Gestore Autenticazione

Questo componente gestisce l'autenticazione degli utenti tramite servizi di autenticazione esterni e/o credenziali salvate nel database.

Tipologia	Nome	Descrizione
Fornita	Autenticazione	Il componente fornisce un'interfaccia che permette la autenticazione di utenti e amministratori.
Richiesta	Servizi di Autenticazione Esterni	Il componente necessita di un'interfaccia fornita da servizi esterni per l'autenticazione degli utenti.
Richiesta	Dati	Il componente utilizza l'interfaccia "Dati" fornita dal Gestore Database per accedere alle credenziali di utenti e amministratori salvate in esso.

1.2.4 Gestore Utente

Questo componente permette agli utenti la gestione del proprio account e dei propri dati.

Tipologia	Nome	Descrizione
Fornita	Registrazione Account	Il componente fornisce un'interfaccia per la registrazione degli utenti. **(RF-UNA1)**
Fornita	Login	Il componente fornisce un'interfaccia per effettuare il login degli utenti. **(RF-UNA2 | RF-A12)**
Fornita	Logout	Il componente fornisce un'interfaccia per effettuare il logout degli utenti. **(RF-UA6 | RF-A13)**
Fornita	Modifica Credenziali	Il componente fornisce un'interfaccia per la modifica delle credenziali d'accesso. **(RF-UA7)**

Fornita	Gestione Profilo	Il componente fornisce un'interfaccia per la modifica delle informazioni presenti nel profilo utente. **(RF-UA7 | RF-UA8)**
Richiesta	Dati	Il componente utilizza l'interfaccia "Dati" fornita dal Gestore Database per accedere ai dati salvati relativi all'utente.
Richiesta	Autenticazione	Il componente utilizza l'interfaccia "Autenticazione" fornita dal Gestore Autenticazione per autenticarsi nel sistema.
Richiesta	Notifiche	Il componente utilizza l'interfaccia "Notifiche" fornita dal Gestore Admin per notificare informazioni di rilievo agli utenti. **(RF-A7)**

1.2.5 Gestore Admin

Questo componente permette agli amministratori la gestione degli eventi, delle segnalazioni e dei profili utente.

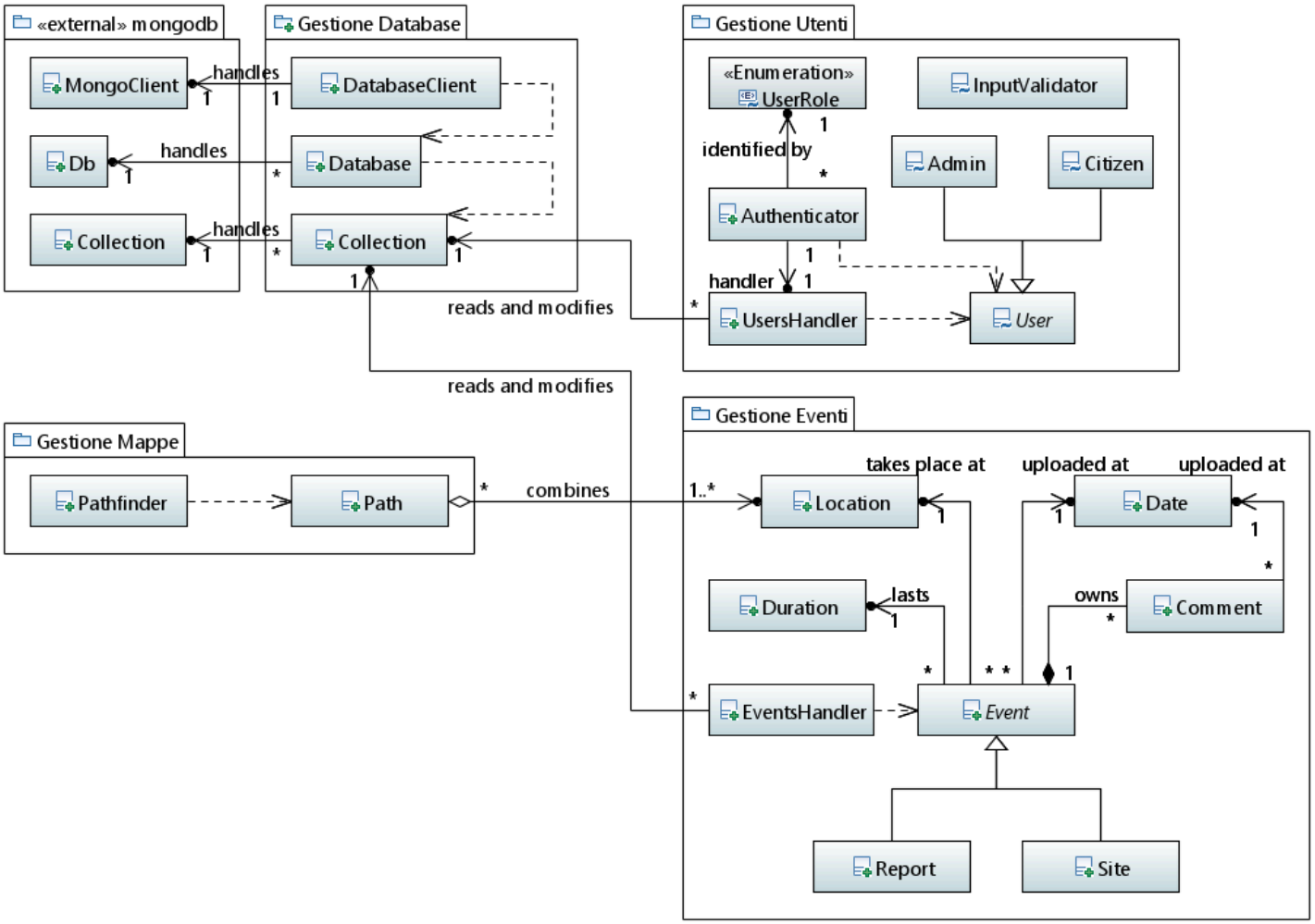
Tipologia	Nome	Descrizione
Fornita	Gestione Eventi	Il componente fornisce un'interfaccia per la gestione degli eventi da parte dell'admin (aggiunta, modifica, annullamento). **(RF-A4 | RF-A5 | RF-A6 | RF-A7)**
Fornita	Gestione Segnalazioni	Il componente fornisce un'interfaccia che permette l'eliminazione di segnalazioni o la loro approvazione. **(RF-A14)**
Fornita	Visualizzazione Utenti	Il componente fornisce un'interfaccia per la visualizzazione degli utenti registrati nel sistema. **(RF-A9 | RF-A10)**
Fornita	Rimozione Utente	Il componente fornisce un'interfaccia per l'eliminazione dei profili utente. **(RF-A11)**
Fornita	Notifiche	Il componente fornisce un'interfaccia per l'invio di notifiche agli utenti riguardo eventi, segnalazioni o modifiche al sistema. **(RF-A7)**
Richiesta	Dati	Il componente utilizza l'interfaccia "Dati" fornita dal Gestore Database per accedere ai dati relativi all'amministratore, agli utenti e agli eventi in esso salvati.
Richiesta	Autenticazione	Il componente utilizza l'interfaccia "Autenticazione" fornita dal Gestore Autenticazione per autenticarsi nel sistema.
Richiesta	Segnalazioni	Il componente utilizza l'interfaccia "Segnalazioni" fornita dal Gestore Eventi per ottenere segnalazioni da gestire. **(RF-UA1)**

1.2.6 Gestore Eventi e Segnalazioni

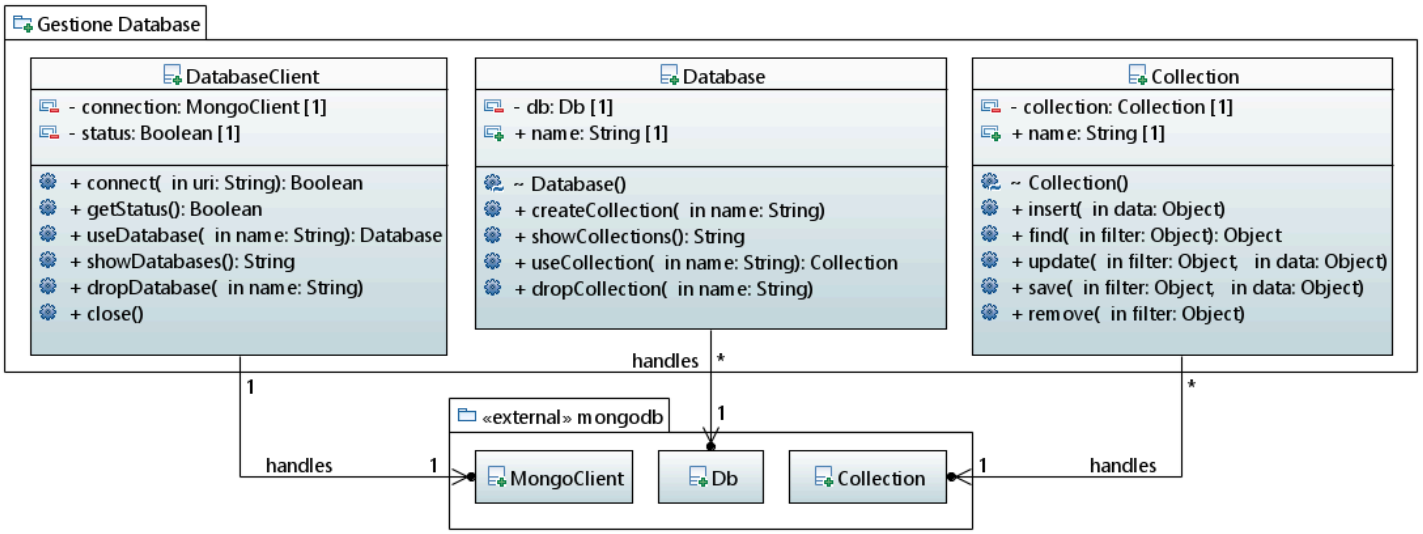
Questo componente fornisce le funzionalità relative alla mobilità tenendo conto di eventi e segnalazioni presenti sul territorio.

Tipologia	Nome	Descrizione
Fornita	Segnalazioni	Il componente fornisce un'interfaccia per l'invio di segnalazioni al Gestore Admin. **(RF-UA2)**
Fornita	Visualizza Eventi e Segnalazioni	Il componente fornisce un'interfaccia che permette di visualizzare eventi e segnalazioni presenti attualmente sulla mappa del territorio. **(RF-U1 | RF-U2 | RF-U3 | RF-U4 | RF-A1 | RF-A2 | RF-A3)**
Fornita	Ricerca Eventi	Il componente fornisce un'interfaccia per la ricerca tramite parametri di eventi. **(RF-U6 | RF-U7)**
Fornita	Tracciamento Percorso	Il componente fornisce un'interfaccia che permette di tracciare il percorso migliore considerando gli eventi presenti sul territorio e le richieste dell'utente. **(RF-U8 | RF-U9)**
Fornita	Stima Tempo e Carburante Utilizzati	Il componente fornisce un'interfaccia per la stima del tempo necessario a percorrere il percorso scelto e per la stima del carburante risparmiato rispetto a percorsi alternativi. **(RF-U10)**
Fornita	Aggiunta Commenti	Il componente fornisce un'interfaccia che permette l'aggiunta di commenti a eventi e segnalazioni presenti sulla mappa. **(RF-UA4)**
Richiesta	Dati	Il componente utilizza l'interfaccia "Dati" fornita dal Gestore Database per accedere ai dati relativi a eventi, segnalazioni e commenti.
Richiesta	Mappa	Il componente utilizza l'interfaccia "Mappa" fornita dal Gestore Percorsi per visualizzare eventi e segnalazioni presenti sul territorio.

2 Diagramma delle classi e descrizione delle classi



2.1 Gestione Database



2.1.1 DatabaseClient

Questa classe si occupa di gestire la connessione con il server MongoDB contenente i dati persistenti del sistema. I suoi attributi permettono di memorizzare la connessione attraverso la relativa libreria esterna e verificarne lo stato.

Operazione	Descrizione
connect(String uri) : Boolean	Effettua un tentativo di connessione al server MongoDB attraverso l'URI specificato e ritorna *True* se avviene con successo, *False* altrimenti (impostando anche il valore di *status*).
getStatus()	Ritorna lo stato della connessione (*True* se attualmente connesso, *False* altrimenti).
useDatabase(String name) : Database	Costruisce e ritorna un oggetto di tipo *Database* connesso al database (nuovo o esistente) su MongoDB chiamato *name*.
showDatabases() : String [*]	Mostra i nomi di tutti i database attualmente esistenti nel server MongoDB selezionato.
dropDatabase(String name)	Elimina il database presente sul server MongoDB chiamato *name*.
close()	Chiude la connessione al server MongoDB e imposta il valore di *status* al nuovo stato.

2.1.2 Database

Questa classe si occupa di gestire la connessione con un database sul server MongoDB del sistema. I suoi attributi forniscono alcune informazioni sul database attraverso la relativa libreria esterna e le sue operazioni permettono di leggerlo e modificarlo.

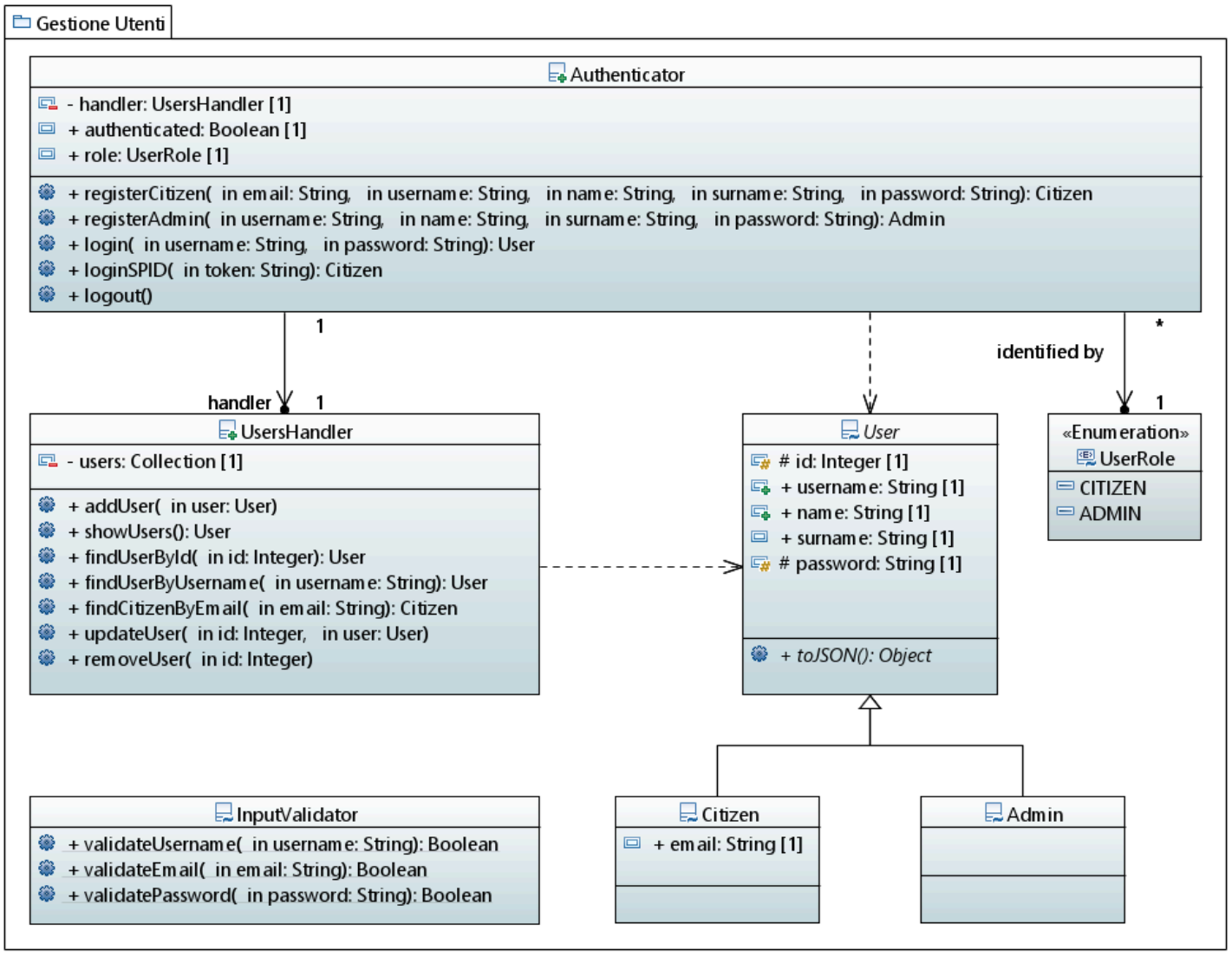
Operazione	Descrizione
Database()	Il costruttore ha visibilità *package* perché dall'esterno le istanze devono essere ottenute esclusivamente attraverso il metodo *useDatabase()* di *DatabaseClient*.
createCollection(String name)	Crea una nuova Collection sul database chiamata *name*.
useCollection(String name) : Collection	Costruisce e ritorna un oggetto di tipo *Collection* connesso alla collection esistente nel database MongoDB selezionato chiamata *name*.
showCollections() : String [*]	Mostra i nomi di tutti le collection attualmente esistenti nel database MongoDB selezionato.
dropCollection(String name)	Elimina la collection esistente nel database MongoDB selezionato chiamata *name*.

2.1.3 Collection

Questa classe si occupa di gestire la connessione con una collection di un database sul server MongoDB del sistema. I suoi attributi forniscono alcune informazioni sulla collection attraverso la relativa libreria esterna e le sue operazioni permettono di leggerla e modificarla.

Operazione	Descrizione
Collection()	Il costruttore ha visibilità *package* perché dall'esterno le istanze devono essere ottenute esclusivamente attraverso il metodo *useCollection()* di *Database*.
insert(Object data)	Inserisce un nuovo document nella collection selezionata.
find(Object filter) : Object	Cerca e restituisce, se trovato, un document nella collection che rispetta le caratteristiche di *filter*.
update(Object filter, Object data)	Modifica tutti i document che rispettano le caratteristiche di *filter* con i valori di *data*.
save(Object filter, Object data)	Salva al posto di tutti i document che rispettano le caratteristiche di *filter* il document *data*.
remove(Object filter)	Elimina dalla collection tutti i document che rispettano le caratteristiche di *filter*.

2.2 Gestione Utenti



2.2.1 User, Citizen, Admin

Queste classi rappresentano gli utenti del sistema. La classe *User* è definita astratta perché ciascun utente del sistema può essere esclusivamente un cittadino oppure un amministratore, rappresentati dalle rispettive classi figlie *Citizen* e *Admin*.

Operazione	Descrizione
toJSON() : Object	Restituisce un oggetto in formato JSON con i valori degli attributi dell'istanza per poterla salvare nel database. Il metodo è astratto perché verrà correttamente implementato nelle classi figlie *Citizen* e *Admin*.

2.2.2 UserRole

Questa è una enumerazione che permette di distinguere due ruoli dell'utente, *CITIZEN* per il ruolo del cittadino e *ADMIN* per il ruolo dell'admin.

2.2.3 InputValidator

Questa classe fornisce metodi statici per verificare se indirizzi email, nomi utente e password rispettano correttamente il formato desiderato.

Operazione	Descrizione
validateUsername(String username) : Boolean	Verifica il formato del nome utente *username*.
validateEmail(String email) : Boolean	Verifica il formato dell'indirizzo mail *email*.
validatePassword(String password) : Boolean	Verifica il formato della password *password*.

2.2.4 UsersHandler

Questa classe si occupa di gestire la collection degli utenti del sistema. I suoi attributi forniscono informazioni sulla collection e le sue operazioni permettono di leggerla e modificarla.

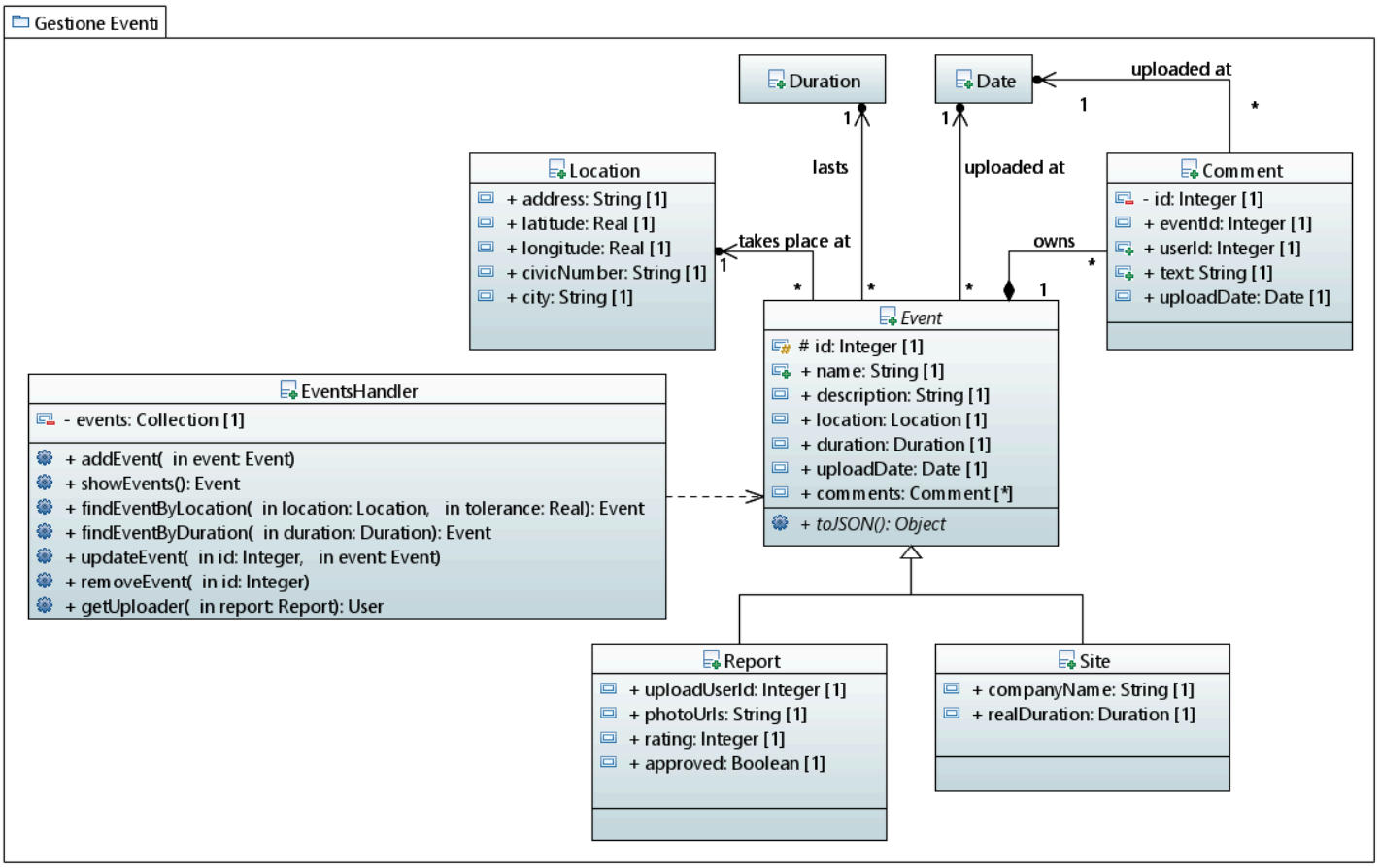
Operazione	Descrizione
addUser(User user)	Inserisce l'utente *user* nella collection del sistema.
showUsers() : User [*]	Restituisce tutti gli utenti salvati nel sistema.
findUserById(Integer id) : User	Cerca e restituisce, se trovato, l'utente del sistema con l'identificativo *id*.
findUserByUsername(String username) : User	Cerca e restituisce, se trovato, l'utente del sistema con il nome utente *username*.
findCitizenByEmail(String email) : Citizen	Cerca e restituisce, se trovato, il cittadino del sistema con l'indirizzo mail *email*.
updateUser(Integer id, User user)	Modifica l'utente del sistema con l'identificativo *id* con le informazioni contenute in *user*.
removeUser(Integer id)	Rimuove dal sistema, se presente, l'utente con l'identificativo *id*.

2.2.5 Authenticator

Questa classe si occupa di gestire il processo di autenticazione al sistema di tutti gli utenti. I suoi attributi permettono di interrogare il database per avere informazioni e aggiungere o rimuovere utenti.

Operazione	Descrizione
registerCitizen(String email, String username, String name, String surname, String password) : Citizen	Registra un nuovo cittadino all'interno del sistema attraverso i campi richiesti e validati di *email*, *username*, *name*, *surname*, *password*. Ritorna un oggetto di tipo *Citizen* rappresentante il cittadino appena registrato.
registerAdmin(String username, String name, String surname, String password) : Admin	Registra un nuovo amministratore all'interno del sistema attraverso i campi richiesti e validati di *username*, *name*, *surname*, *password*. Ritorna un oggetto di tipo *Admin* rappresentante l'amministratore appena registrato.
login(String username, String password) : User	Verifica che le credenziali inserite *username* e *password* corrispondano ad un utente registrato nel sistema e, in caso affermativo, lo autentica e gli permette di accedere impostando a *True* l'attributo *authenticated*.
loginSPID(String token) : Citizen	Verifica che l'autenticazione tramite SPID sia avvenuta correttamente e, in caso affermativo, autentica il cittadino permettendogli di accedere al sistema e impostando a *True* l'attributo *authenticated*.
logout()	Effettua il logout dal sistema dell'utente impostando a *False* l'attributo *authenticated*.

2.3 Gestione Eventi



2.3.1 Event, Report, Site

Queste classi rappresentano gli eventi registrati nel sistema. La classe *Event* è definita astratta perché ciascun evento del sistema può essere esclusivamente una segnalazione oppure un cantiere, rappresentati dalle rispettive classi figlie *Report* e *Site*.

Operazione	Descrizione
toJSON() : Object	Restituisce un oggetto in formato JSON con i valori degli attributi dell'istanza per poterla salvare nel database. Il metodo è astratto perché verrà correttamente implementato nelle classi figlie *Report* e *Site*.

2.3.2 Location

Questa classe rappresenta un indirizzo e le coordinate geografiche di quella posizione. La classe *Event* possiede un attributo di questo tipo per rappresentare il luogo in cui avviene l'evento registrato.

2.3.3 Duration

Questa classe rappresenta una data e una durata nel tempo. La classe *Event* possiede un attributo di questo tipo per rappresentare la durata in cui si svolge l'evento registrato.

2.3.4 Date

Questa classe rappresenta una data, più o meno precisa. La classe *Event* possiede un attributo di questo tipo per rappresentare il momento in cui l'evento viene registrato nel sistema.

2.3.5 Comment

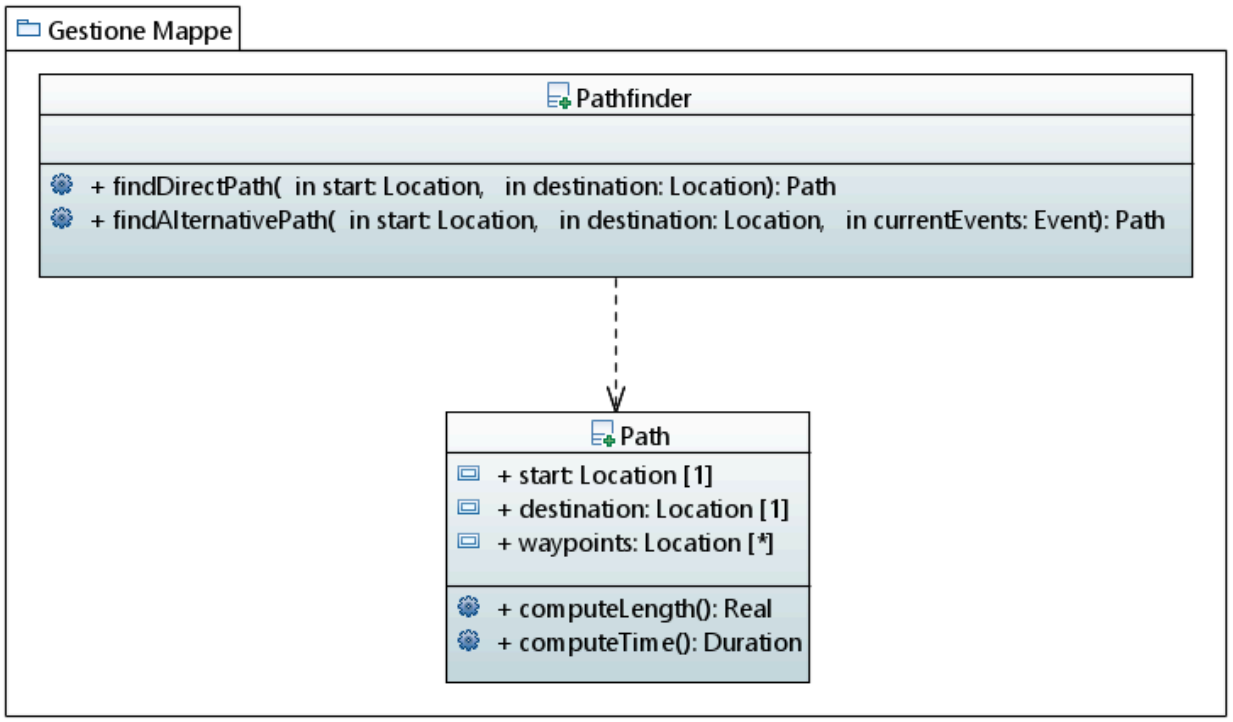
Questa classe rappresenta un commento. La classe *Event* possiede un attributo di questo tipo per rappresentare i commenti relativi a un evento registrato.

2.3.6 EventsHandler

Questa classe si occupa di gestire la collection degli eventi del sistema. I suoi attributi forniscono informazioni sulla collection e le sue operazioni permettono di leggerla e modificarla.

Operazione	Descrizione
addEvent(Event event)	Inserisce l'evento *event* nella collection del sistema.
showEvents() : Event [*]	Restituisce tutti gli eventi salvati nel sistema.
findEventByLocation(Location location, Real tolerance) : Event [*]	Cerca e restituisce, se trovati, gli eventi del sistema che avvengono nei dintorni di *location* con una distanza massima di *tolerance*.
findEventByDuration(Duration duration) : Event [*]	Cerca e restituisce, se trovati, gli eventi del sistema che avvengono nel periodo di tempo *duration*.
removeUser(Integer id)	Rimuove dal sistema, se presente, l'evento con l'identificativo *id*.
getUploader(Report report) : User	Restituisce l'oggetto di tipo *User* rappresentante l'utente che ha caricato la segnalazione *report*.

2.4 Gestione Mappe



2.3.1 Path

Questa classe rappresenta un percorso che unisce un punto di partenza e un punto di arrivo, attraversando una serie di posizioni intermedie. I suoi attributi sono le posizioni che compongono il percorso e le sue operazioni permettono di calcolare alcune informazioni sullo stesso.

Operazione	Descrizione
computeLength() : Real	Calcola e restituisce la lunghezza del percorso descritto.
computeTime() : Duration	Calcola e restituisce il tempo impiegato per seguire il percorso descritto.

2.3.2 Pathfinder

Questa classe si occupa di gestire le interazioni con il sistema esterno di mappe stradali, permettendo di richiedere e fornire i percorsi stradali.

Operazione	Descrizione
findDirectPath(Location start, Location destination) : Path	Attraverso i sistemi esterni di mappe stradali traccia il percorso diretto (più breve) che unisce i due punti di partenza *start* e arrivo *destination*.
findAlternativePath(Location start, Location destination, Event currentEvents [*]) : Path	Attraverso i sistemi esterni di mappe stradali traccia un percorso alternativo che unisce i due punti di partenza *start* e arrivo *destination* senza attraversare le strade in cui ci sono degli eventi in corso.

3 Constraint OCL

I singoli vincoli verranno definiti con una tabella (come nell'esempio):

Descrizione del Vincolo
Constraint Expression

3.1 Gestione Database

3.1.1 Database Client

Impossibile connettersi a più server diversi
context DatabaseClient::connect() pre: self.connection -> Empty() AND self.status == False post: self.connection = MongoClient.URI

Il DatabaseClient deve esistere ed essere online
context DatabaseClient pre: self.connection -> notEmpty() inv: DatabaseClient.connect(uri) == True post: self.status = True

Vengono eliminati solo database esistenti

context DatabaseClient::dropDatabase(name)
pre: self.connection -> notEmpty() AND self.getStatus() == True
inv: showDatabase(name) -> exists(name) == True

I nomi dei database devono essere interpretabili

context DatabaseClient::useDatabase(String name)
pre: name.size > 0 AND name.size < 10

3.1.2 Database

I nomi delle collections devono essere interpretabili

context Database::createCollection(String name)
pre: name.size > 0 AND name.size < 10

Devono essere utilizzate solo collection esistenti

context Database::useCollection(String name)
pre: showCollection() -> Exists(name) == True

Possono essere eliminate solo collections esistenti

context Database::dropCollection(String name)
pre: showCollection() -> Exists(name) == True

3.2 Gestione Utenti

3.2.1 Authenticator

Dopo aver eseguito il login l'attributo autenticato viene impostato a True e gli altri attributi popolati

context Authenticator::login(username, password)
post: self.autenticato = true AND self.role -> notEmpty()

Dopo aver eseguito il login con SPID l'attributo autenticato viene settato a true e gli altri attributi popolati

context Authenticator::loginSPID(token)
pre: loginSPID() -> Exists(token)
inv: registerCitizen(information given by the SPID service)
post: self.autenticato = true AND self.role -> notEmpty()

Dopo aver eseguito il logout l'attributo autenticato viene settato a false

context Authenticator::logout()
post: self.autenticato = false

La password di un utente registrato deve rispettare il formato corretto

context Authenticator::registerCitizen(...)
pre: password.size() >=8 and password.size() <=10

Il ruolo deve essere settato a CITIZEN quando viene registrato attraverso registerCitizen()

context Authenticator::registerCitizen(...)
post: self.role = CITIZEN

Il ruolo deve essere settato a ADMIN quando viene registrato attraverso registerAdmin()

context Authenticator::registerAdmin(...)
post: User.role = ADMIN

3.2.2 Handler

Possono essere eliminati solo utenti esistenti

context Database::removeUser(id)
pre: findUserById() -> Exists(id) == True

3.3 Gestioni Eventi

3.3.1 Event

La durata dell'evento deve essere valida

context Event
pre: duration > 0 (hours/minutes/days) AND < 10 (years)

La data di upload non puó essere posteriore al giorno presente

context Event
pre: uploadDate <= TodayDate

3.3.2 EventHandler

Possono essere eliminati solo eventi esistenti

context EventHandler::removeEvent(id)
pre: showEvent() -> Exists(id) == True

3.3.3 Location

La location degli eventi deve essere valida

context Location
pre: latitude > -90 AND latitude < 90 AND longitude > -90 AND longitude < 90

3.4 Gestione Mappe

3.4.1 Path

La location di inizio e fine deve esistere

context Path
pre: self.start.latitude > -90 AND self.start.latitude < 90 AND self.destination.longitude > -90 AND self.destination.longitude < 90

La distanza non puó essere negativa

context Path::computeLength(): Real
inv: distance > 0

Il tempo di percorrenza non puó essere negativo

context Path::computeTime(): Duration
inv: Duration > 0 (minutes/hours)