UNIVERSITÀ DI TRENTO

Logoipsum

D3

Documento di design architetturale

Gruppo T31

- Luca Dematté
- Diego Oniarti
- Matteo Frizzera

Ingegneria del software

Anno Accademico 2022/23

Versione del documento

Versione	Data	Modifiche
0.1	30/11/2022	Creazione documento.
0.2	2/12/2022	Strutturazione documento e aggiunta immagini classi.
0.5	4/12/2022	Scrittura della sezione su OCL.
0.6	5/12/2022	Aggiunta descrizione classi tipi di dato.
1.0	**7/12/2022**	**Aggiunte descrizioni classi logica e classi pagine.**

Sommario

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.

Diagramma delle classi

Nel presente capitolo vengono presentate le classi previste per il sistema. 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.

Per necessità di layout e chiarezza espositiva, il diagramma delle classi complessivo è stato diviso in tre parti, ognuna delle quali si focalizza su un determinato aspetto da rappresentare: i tipi di dato utilizzati nel sistema, le classi di interazione con il database/business logic e infine le classi che si occupano di interagire con l'esterno del sistema.

1. Classi tipi di dato

Durante il passaggio dal diagramma dei componenti al diagramma delle classi, è apparsa subito evidente la necessità di inserire delle classi per rappresentare le varie entità trattate dal sistema:

1.1. Utente e token

La classe Utente rappresenta il normale utilizzatore del sistema, come è stato descritto nei precedenti diagrammi. Gli attributi memorizzano i dati personali e le informazioni di accesso al sistema. Per la verifica dell'accesso prima dell'esecuzione di una qualsiasi funzionalità che richieda un utente loggato, è stata ideata la classe Token. In pratica, ogni classe che svolge una funzione

riservata a una certa categoria di utenti richiede il passaggio dell'oggetto Token che ricopre due ruoli:

- fornire informazioni sui permessi concessi all'utente (tramite l'attributo privilegio),
- fornire l'identificativo dell'utente che svolge l'azione qualora la funzionalità abbia bisogno di recuperare dati relativi all'utente.

1.2. Metodo di pagamento

La classe Metodo di pagamento contiene le informazioni inserite dall'utente sul suo account di pagamento digitale. Nel caso l'utente non abbia ancora inserito un metodo di pagamento digitale, l'attributo nella classe Utente rimarrà nullo.

1.3. Spazi e servizi

Le classi Spazio e Servizio sono relativamente simili tra loro e contengono gli attributi necessari per poter descrivere ogni oggetto sul sito nella sezione dedicata. L'unica differenza è l'attributo servizi, che memorizza quali servizi sono collegati all'oggetto spazio per poter essere utilizzati durante una prenotazione.

1.4. Eventi

La classe Evento, similmente alle classi Spazio e Servizio, contiene le informazioni inserite dall'utente da mostrare sulla pagina dedicata sul sito, come ad esempio titolo, sottotitolo, una descrizione testuale, il nome dell'organizzatore dell'evento e un banner come immagine a scopo decorativo.

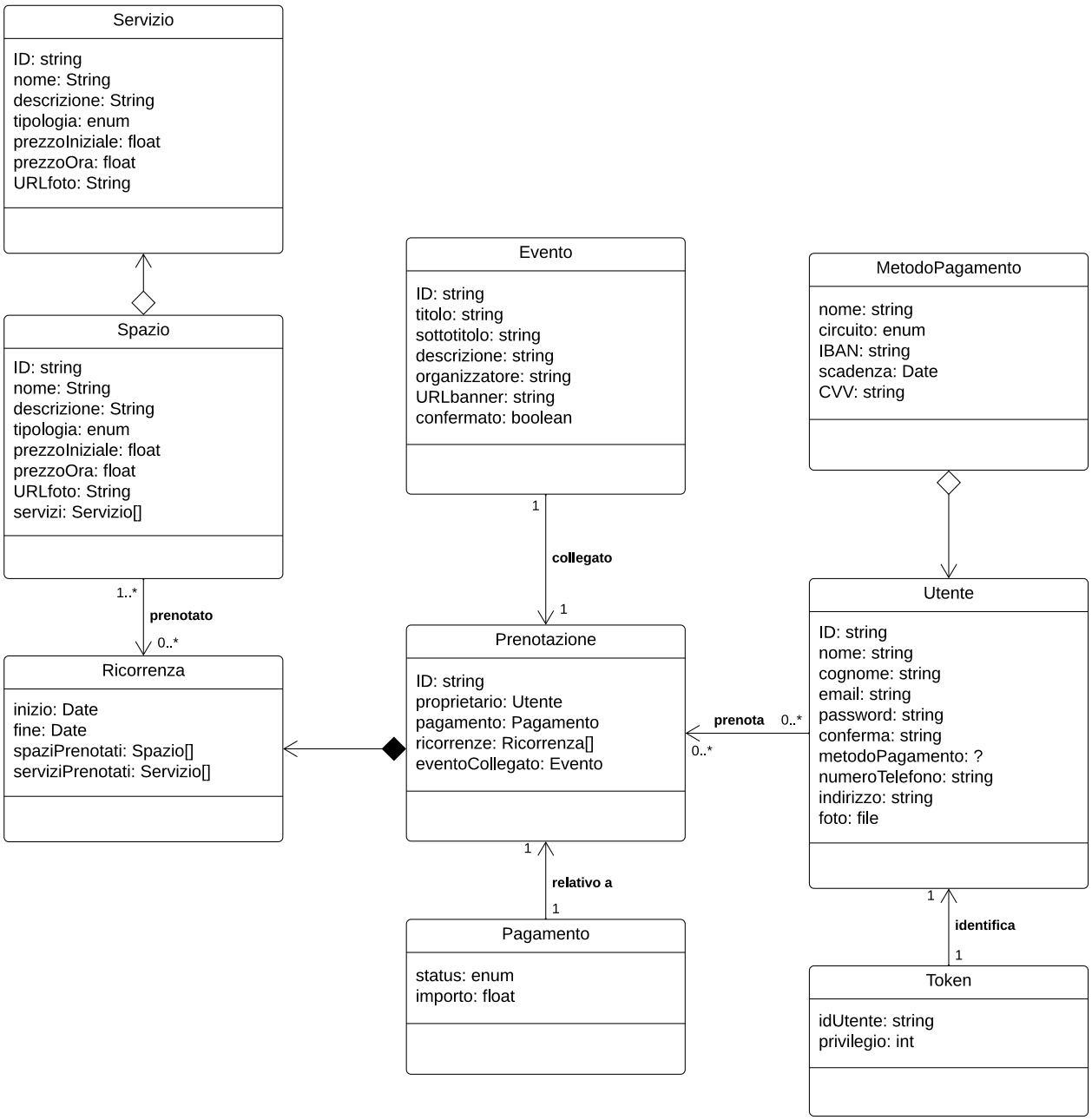
1.5. Prenotazioni

La struttura per la parte dedicata a rappresentare le prenotazioni è stata pensata per rendere possibile la prenotazione con ricorrenze. La classe Prenotazione contiene le informazioni generali e i riferimenti alle altre classi relazionate con la prenotazione. La classe Ricorrenza contiene invece l'inizio e fine della singola ricorrenza e gli spazi e servizi inclusi nella prenotazione.

Questo design permette la massima flessibilità nella gestione di una prenotazione con ricorrenze; infatti, l'utente creerà la prenotazione inserendo le stesse impostazioni per tutte le ricorrenze (che verranno create automaticamente dal sistema). Una volta memorizzate, le ricorrenze sono indipendenti l'una dall'altra e potranno quindi essere modificate singolarmente in caso di variazioni (ad esempio un cambio di orario o un cambio di spazio/servizio utilizzato).

1.6. Pagamento

Ogni prenotazione è collegata a un oggetto di tipo Pagamento che memorizza il totale calcolato in base agli spazi, i servizi e il tempo di utilizzo.



2. Classi di business logic e interazione con il database

Di seguito sono state raggruppate le classi che si occupano di implementare la logica delle operazioni e la comunicazione da e verso il database:

2.1. InterfacciaDB

Questa classe si occupa di stabilire e poi gestire la connessione con il database per permettere alle altre classi di effettuare operazioni di lettura e scrittura di dati.

2.2. EventiDB, AccountDB, SpaziServiziDB, PrenotazioniDB

Queste quattro classi specializzano, tramite ereditarietà, le funzionalità della classe InterfacciaDB, fornendo dei metodi più strutturati per comunicare con il database. È importante notare che queste classi si limitano a recuperare e inviare dati al database, senza svolgere alcuna operazione di carattere logico, aspetto riservato alle classi descritte di seguito.

2.3. GestoreEventi

Appoggiandosi alla classe EventiDB per comunicare con il database, questa classe mette a disposizione tutte le operazioni che possono essere svolte sugli eventi registrati. Ad esempio, sono presenti i metodi `aggiungiNuovoEvento()` per inserire un nuovo evento oppure `convalidaEvento()` per permettere alla segreteria di rendere gli eventi visibili sul sito.

2.4. GestoreUtente e SistemaRegistrazione

GestoreUtente si occupa di effettuare tutte le operazioni sui dati degli account utenti memorizzati nel database. Questa classe viene utilizzata da SistemaRegistrazione, una classe dedicata alla procedura di aggiunta di un nuovo utente.

2.5. InterfacciaServizioEmail

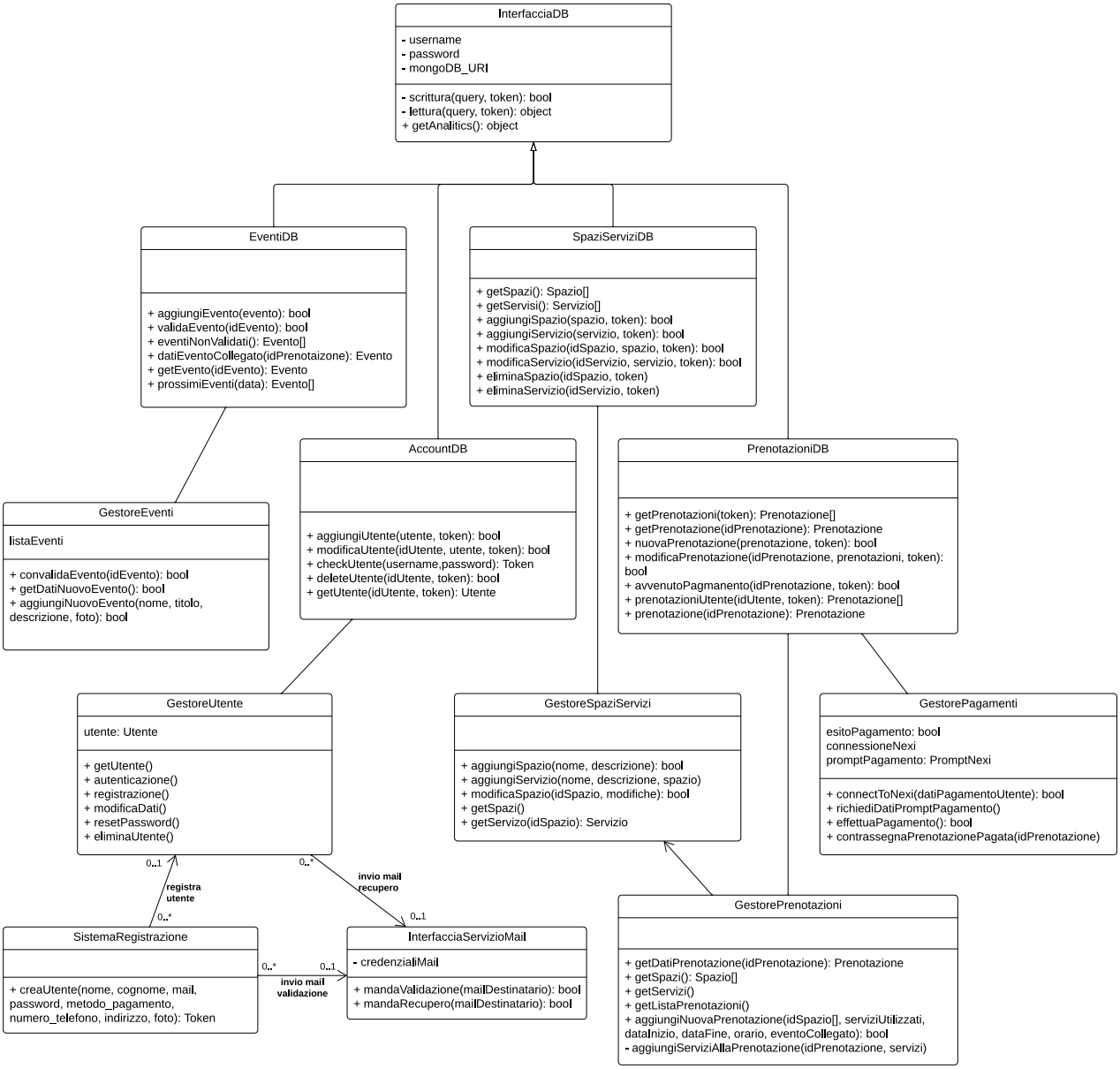
L'interazione con il sistema esterno per l'invio di e-mail agli utenti è assegnata a questa classe, che mette a disposizione dei metodi per inviare e-mail predefinite (ad esempio per il reset della password o la conferma dell'e-mail di un nuovo account), oltre al meccanismo di connessione al sistema.

2.6. GestoreSpaziServizi e GestorePrenotazioni

Queste due classi rappresentano il "cuore" del meccanismo di prenotazione fornito agli utenti. Infatti, queste classi rendono disponibili funzionalità per aggiungere/modificare/ottenere gli spazi e per aggiungere nuove prenotazioni. La classe GestorePrenotazioni, che ha anche il compito di effettuare tutti i controlli sui dati prima del loro invio al database, deve comunicare con GestoreSpaziServizi per ottenere le informazioni relative agli spazi.

2.7. GestorePagamenti

La classe GestorePagamenti ha il compito di trattare tutte le funzioni relative ai pagamenti, quindi il collegamento al sistema di pagamento online Nexi, la procedura di registrazione del pagamento e infine l'aggiornamento dello stato della prenotazione tramite la classe PrenotazioniDB.



3. Classi di interazione con l'utente

Queste classi forniscono un modo semplice per l'utente di interagire con il sistema:

3.1. Pagina

Una classe astratta che rappresenta una pagina web del sito che viene specializzata tramite ereditarietà dalle altre classi pagine.

Le pagine hanno un livello di riservatezza che indica quali utenti possono vederne il contenuto (ad esempio, alcune pagine possono essere visualizzate unicamente dall'utente di segreteria).

Le pagine possono anche utilizzare i Token salvati dal browser per ottenere informazioni riguardanti l'utente e modificare il loro contenuto di conseguenza.

3.2. Navbar

Tutte le pagine contengono un oggetto di classe navbar. Questo oggetto utilizza il Token salvato dal browser per presentare all'utente:

- I tasti "login" e "registrati" qualora l'utente non sia loggato.
- Il nome dell'utente, la foto dell'utente, e il tasto "logout" in caso sia loggato.

La navbar permette inoltre all'utente non loggato di effettuare il login tramite un pop-up.

3.3. SistemaAutenticazione

Questa classe permette alle pagine e alla navbar del sito di utilizzare le funzioni di jwt.js per verificare l'identità dell'utente corrente, ovvero:

- Il metodo `login()`, che esegue l'accesso al database per confrontare le credenziali indicate con quelle salvate. Questo metodo ritorna un Token che verrà salvato nel browser per ricordare l'utente.
- Il metodo `verificaToken()`, che prende in input un Token e, utilizzando funzionalità specifiche di jwt.js, restituisce un oggetto di classe Utente identificato da quel Token.

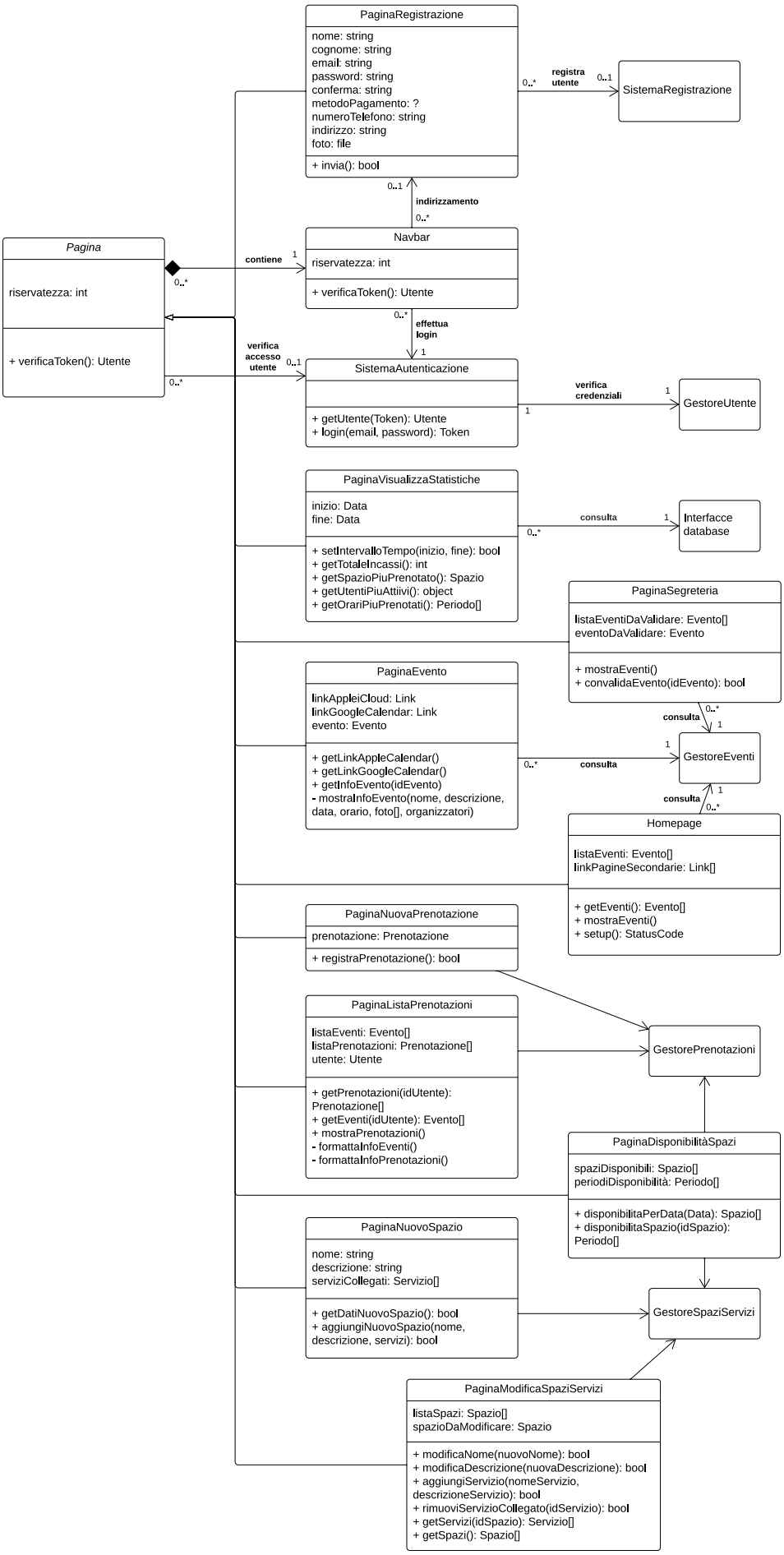
3.4. PaginaRegistrazione

La pagina di registrazione è accessibile in ogni momento dall'utente non loggato tramite la navbar. Questa pagina contiene un form che permette all'utente di creare un account fornendo le informazioni necessarie.

3.5. Altre pagine

Tutte le pagine presenti nel sito sono rappresentate come classi che estendono la classe generale "Pagina" e hanno le seguenti caratteristiche:

- Le diverse pagine possono accedere ai vari gestori e alle interfacce database per ottenere le informazioni da mostrare all'utente e per inviare al sistema le informazioni fornite dall'utente.
- Le pagine possono implementare comportamenti differenti in base al livello di autorità dell'utente e modificare il contenuto presentato in base allo specifico utente.
- Alcune pagine del sito non sono associate ad alcuna classe. Questo perché sono pagine statiche che non presentano alcuna logica e sono sempre accessibili da ogni utente.


PaginaRegistrazione
nome: string
cognome: string
email: string
password: string
conferma: string
metodoPagamento: ?
numeroTelefono: string
indirizzo: string
foto: file
+ invia(): bool
registra utente
SistemaRegistrazione
indirizzamento
Pagina
riservatezza: int
+ verificaToken(): Utente
contiene
Navbar
riservatezza: int
+ verificaToken(): Utente
effettua login
verifica accesso utente
SistemaAutenticazione
+ getUtente(Token): Utente
+ login(email, password): Token
verifica credenziali
GestoreUtente
PaginaVisualizzaStatistiche
inizio: Data
fine: Data
+ setIntervalloTempo(inizio, fine): bool
+ getTotaleIncassi(): int
+ getSpazioPiuPrenotato(): Spazio
+ getUtentiPiuAttiivi(): object
+ getOrariPiuPrenotati(): Periodo[]
consulta
Interfacce database
PaginaSegreteria
listaEventiDaValidare: Evento[]
eventoDaValidare: Evento
+ mostraEventi()
+ convalidaEvento(idEvento): bool
PaginaEvento
linkAppleiCloud: Link
linkGoogleCalendar: Link
evento: Evento
+ getLinkAppleCalendar()
+ getLinkGoogleCalendar()
+ getInfoEvento(idEvento)
- mostraInfoEvento(nome, descrizione, data, orario, foto[], organizzatori)
GestoreEventi
Homepage
listaEventi: Evento[]
linkPagineSecondarie: Link[]
+ getEventi(): Evento[]
+ mostraEventi()
+ setup(): StatusCode
PaginaNuovaPrenotazione
prenotazione: Prenotazione
+ registraPrenotazione(): bool
PaginaListaPrenotazioni
listaEventi: Evento[]
listaPrenotazioni: Prenotazione[]
utente: Utente
+ getPrenotazioni(idUtente): Prenotazione[]
+ getEventi(idUtente): Evento[]
+ mostraPrenotazioni()
- formattaInfoEventi()
- formattaInfoPrenotazioni()
GestorePrenotazioni
PaginaDisponibilitàSpazi
spaziDisponibili: Spazio[]
periodiDisponibilità: Periodo[]
+ disponibilitaPerData(Data): Spazio[]
+ disponibilitaSpazio(idSpazio): Periodo[]
PaginaNuovoSpazio
nome: string
descrizione: string
serviziCollegati: Servizio[]
+ getDatiNuovoSpazio(): bool
+ aggiungiNuovoSpazio(nome, descrizione, servizi): bool
GestoreSpaziServizi
PaginaModificaSpaziServizi
listaSpazi: Spazio[]
spazioDaModificare: Spazio
+ modificaNome(nuovoNome): bool
+ modificaDescrizione(nuovaDescrizione): bool
+ aggiungiServizio(nomeServizio, descrizioneServizio): bool
+ rimuoviServizioCollegato(idServizio): bool
+ getServizi(idSpazio): Servizio[]
+ getSpazi(): Spazio[]

Object Constraint Language (OCL)

In questo capitolo è descritta in modo formale la logica prevista nell'ambito di alcune operazioni delle 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.

Viene seguita la stessa separazione utilizzata nel capitolo precedente.

OCL Classi tipi di dato

- Nella classe Servizio e Spazio i prezzi devono essere numeri positivi e la tipologia deve essere tra quelle specificate:

```
context Servizio inv: prezzoIniziale >= 0
context Servizio inv: prezzoOra >= 0
context Servizio inv: tipologia.values()->any( t | tipologia = t )

context Spazio inv: prezzoIniziale >= 0
context Spazio inv: prezzoOra >= 0
context Servizio inv: tipologia.values()->any( t | tipologia = t )
```

- Nella classe Prenotazione la data di inizio deve essere antecedente a quella di fine:

```
context Prenotazione inv: fine - inizio >= 0
```

- Nella classe MetodoPagamento il circuito di pagamento deve essere tra quelli accettati, l'IBAN deve essere composto da 27 caratteri e il CVV deve essere di 3 o 4 cifre:

```
context MetodoPagamento inv: IBAN.size() = 27
context MetodoPagamento inv: CVV.size() = 3 OR CVV.size() = 4
context MetodoPagamento inv: circuito IN circuitiValidi
context MetodoPagamento inv: circuito.values()->any(c | circuito=c)
```

- Nella classe Utente l'e-mail deve contenere il carattere '@' ed avere username (parte prima il carattere '@') e un dominio (parte dopo il carattere '@'). Il numero di telefono deve essere composto da massimo 15 cifre, la password deve rispettare il RNF2.2 (avere almeno 8 caratteri, tra cui almeno un carattere maiuscolo, un

numero e un carattere speciale). Inoltre la foto profilo dell'utente ha una dimensione massima pari a 5 MB.

```
context Utente inv: email->any(c | c = '@') AND
email.username.size() > 0 AND email.dominio.size() > 0

context Utente inv: numeroTelefono.size() <= 15

context Utente inv: password.size() >= 8 AND password->any(
caratterePassword | "abcdefghijklmnopqrstuvwxyz"-> any(
caratttereMinuscolo | caratterePassword = caratttereMinuscolo  AND
"0123456789"->any( numero | caratterePassword = numero ) AND
"!"£$%&/()=?'^[]ç@°#§-,._;:|\><*`~+€"->any ( carattereSpeciale |
caratterePassword = carattereSpeciale ) )

context Utente inv: foto.size <= 5 MB
```

- Nella classe Pagamento l'importo pagato deve essere un numero positivo:

```
context Pagamento inv: importo >= 0
```

OCL Classi di business logic e interazione con il database

- Nella classe EventiDB è presente un metodo `prossimiEventi(data)` che ritorna tutti gli eventi che si svolgeranno dalla data corrente fino alla data indicata nel parametro. La data corrente deve sempre essere antecedente alla data indicata a parametro.

```
context EventiDB::prossimiEventi(data)

pre: dataCorrente - data >= 0
```

- Nella classe GestorePagamenti per effettuare un pagamento e richiedere il prompt è necessario prima aver effettuato la connessione con Nexi:

```
context GestorePagamenti::richiediDatiPromptPagamento()

pre: connessioneNexi.alive = true


context GestorePagamenti::effettuaPagamento()

pre: connessioneNexi.alive = true
```

- Nella classe GestorePrenotazioni è possibile aggiungere un servizio alla prenotazione:

```
context GestorePrenotazioni::aggiungiServiziAllaPrenotazione(
idPrenotazione, servizi)

post: PrenotazioniDB.getPrenotazione(idPrenotazione)
.serviziPrenotati -> includesAll( servizi )
```

OCL Classi di interazione con l'utente

- Nella pagina PaginaVisualizzaStatistiche dopo aver impostato l'intervallo di tempo di cui si vogliono vedere le statistiche, agli attributi inizio e fine deve essere stato attribuito un valore e la data di inizio deve essere antecedente a quella di fine.

  Inoltre per ottenere le statistiche è prima necessario impostare un intervallo di tempo.

```
context PaginaVisualizzaStatistiche inv: fine - inizio > 0

context PaginaVisualizzaStatistiche::setIntervalloTempo( inizio,
fine )

post: inizio != null AND fine != null

context PaginaVisualizzaStatistiche::getTotaleIncassi()

pre: inizio != null AND fine != null

context PaginaVisualizzaStatistiche::getSpazioPiuPrenotato()

pre: inizio != null AND fine != null

context PaginaVisualizzaStatistiche::getUtentiPiuAttivi()

pre: inizio != null AND fine != null

context PaginaVisualizzaStatistiche::getOrariPiuPrenotati()

pre: inizio != null AND fine != null
```

- Nella PaginaRegistrazione gli attributi `email`, `password`, `numeroTelefono` e `foto` rispettano gli stessi invarianti definiti nella classe Utente:

```
context PaginaRegistrazione inv: email->any(c | c = '@') AND
email.username.size() > 0 AND email.dominio.size() > 0

context PaginaRegistrazione inv: numeroTelefono.size() <= 15
```

```
context PaginaRegistrazione inv: password.size() >= 8 AND password->any( caratterePassword | “abcdefghijklmnopqrstuvwxyz”-> any( carattereMinuscolo | caratterePassword = carattereMinuscolo ) AND “0123456789”->any( numero | caratterePassword = numero ) AND “!”£$%&/()=?’^[]ç@°#§-,._;:|\><*`~+€”->any ( carattereSpeciale | caratterePassword = carattereSpeciale ) )

context Utente inv: foto.size <= 5 MB
```

- Nella PaginaSegreteria è necessario selezionare un evento per convalidarlo. Inoltre dopo che è stato chiamato il metodo `convalidaEvento()`, l’evento selezionato deve essere contrassegnato come confermato:

```
context PaginaSegreteria::convalidaEvento( idEvento )

pre: idEvento != null AND eventoDaValidare != null


context PaginaSegreteria::convalidaEvento( idEvento )

post: evento.confermato = true
```

- Nella PaginaListaPrenotazione è necessario specificare un utente per poter richiedere la lista delle sue prenotazioni effettuate ed eventi organizzati:

```
context PaginaListaPrenotazioni::getPrenotazioni( idUtente )

pre: idUtente != null AND utente != null


context PaginaListaPrenotazioni::getEventi( idUtente )

pre:  idUtente != null AND utente != null
```

- Nella PaginaDisponibilitàSpazi dopo aver richiesto gli intervalli di tempo in cui uno spazio è disponibile, questi devono essere salvati nell’attributo `periodiDisponibilità`. Analogamente dopo aver richiesto tutti gli spazi disponibili in una determinata data, questi vengono salvati nell’attributo `spaziDisponibili`:

```
context PaginaDisponibilitàSpazi::disponibilitaPerSpazio(idSpazio)

post: periodiDisponibilità != null


context PaginaDisponibilità::disponibilitaPerData(data)

post: spaziDisponibili != null
```

- Nella PaginaModificaSpazioServizi è sempre necessario selezionare uno spazio da modificare. Dopo aver modificato il nome o la descrizione di uno spazio, queste informazioni devono essere aggiornate. Dopo aver chiamato i metodi `aggiungiServizio()` o `rimuoviServizioCollegato()`, il servizio specificato a parametro deve essere rispettivamente aggiunto o rimosso dallo spazio.

```
context PaginaModificaSpazioServizi::modificaNome( nuovoNome )
pre: spazioDaModificare != null

context PaginaModificaSpazioServizi::modificaDescrizione (
nuovaDescrizione )
pre: spazioDaModificare != null

context PaginaModificaSpazioServizi::aggiungiServizio(
nomeServizio, descrizioneServizio )
pre: spazioDaModificare != null

context PaginaModificaSpazioServizi::rimuoviServizioCollegato(
idServizio )
pre: spazioDaModificare != null

context PaginaModificaSpazioServizi::modificaNome( nuovoNome )
post: self.nome = nuovoNome

context PaginaModificaSpazioServizi::modificaDescrizione(
nuovaDescrizione )
post: self.descrizione = nuovaDescrizione

context PaginaModificaSpazioServizi::aggiungiServizio(
nomeServizio, descrizioneServizio )
post: spazioDaModificare.servizi->any( servizio | servizio.nome =
nomeServizio AND servizio.descrizioneServizio =
descrizioneServizio)

context PaginaModificaSpazioServizi::rimuoviServizioCollegato(
idServizio )
post: spazioDaModificare.servizi->excludes( servizio | servizio.ID
= idServizio)
```