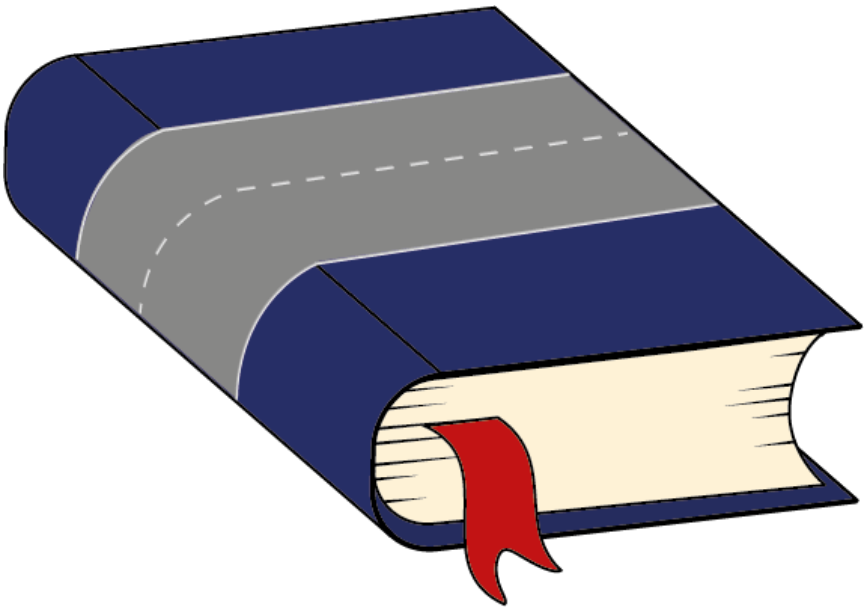
Università degli studi di Trento
Dipartimento di Ingegneria e Scienza dell'Informazione
Anno Accademico 2021-22

Book & Drive



D3: Context Diagrams, Class Diagram, OCL

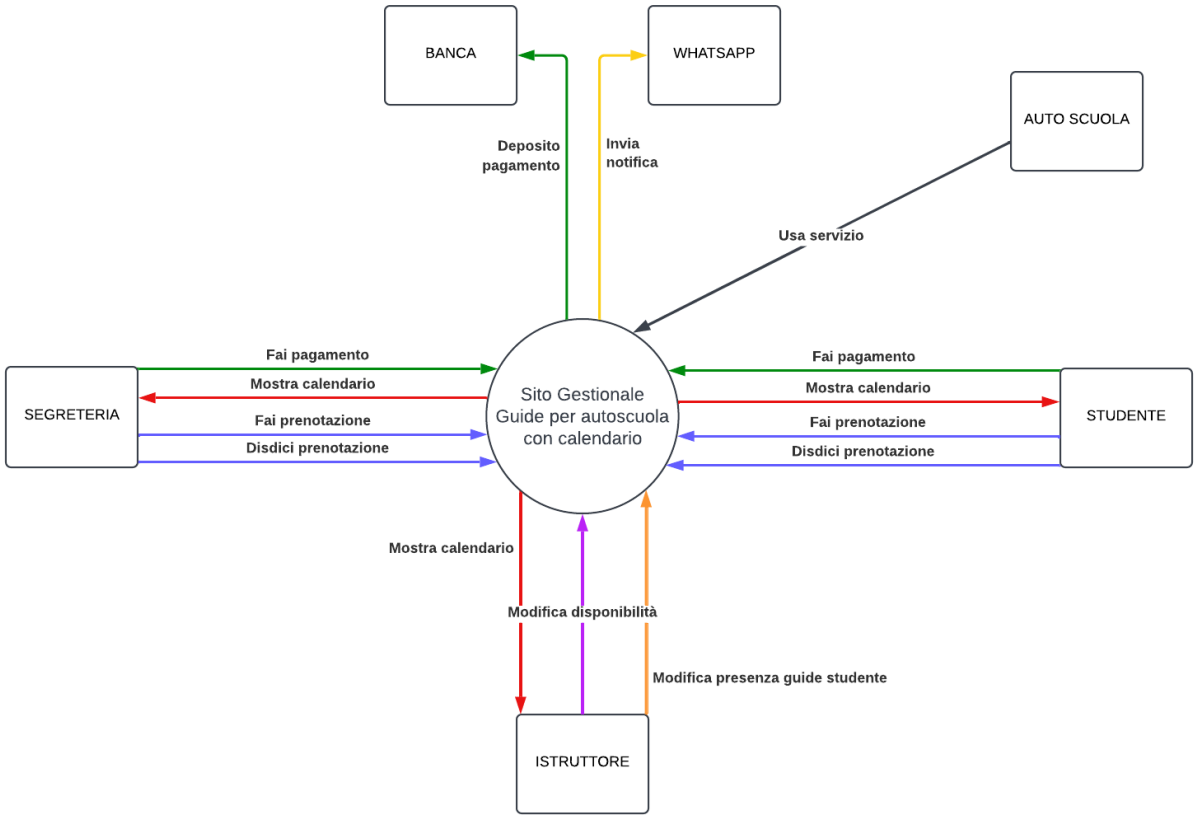
Composta Elisa 209077
Lorenzetti Andrea 209548
Scevaroli Simone 209427
Zeni Luca 210554

SCOPO DEL DOCUMENTO

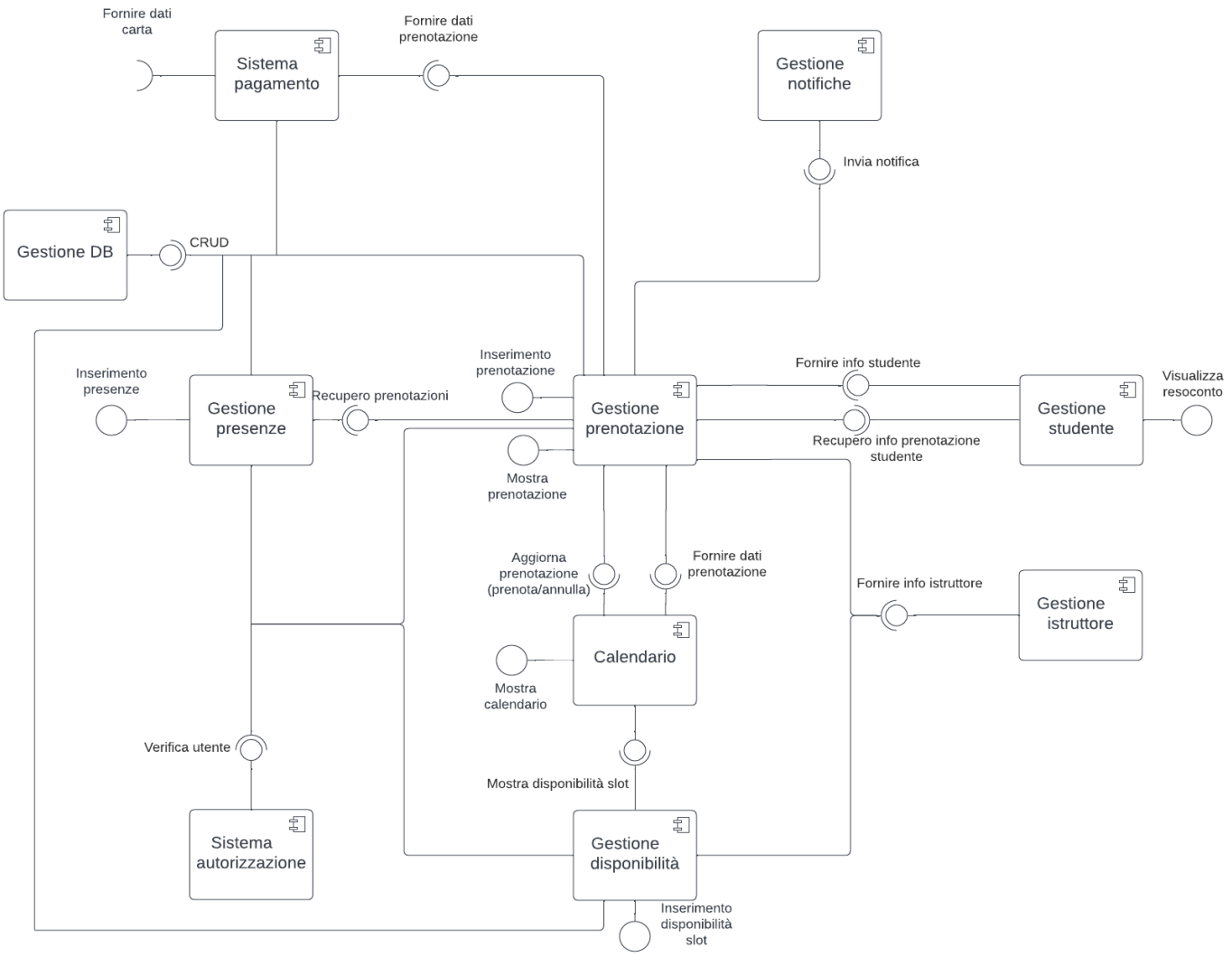
L'obiettivo di questo documento è quello di rappresentare e descrivere il Sequence Diagram relativo all'applicazione "Book & Drive". Inoltre verranno analizzate e definite tutte le componenti facenti parte del seguente diagramma, con i relativi livelli di coesione. L'ultima parte del documento definisce l'architettura del sistema dettagliando le classi che dovranno essere implementate a livello di codice e la logica che regola il comportamento del software in OCL.

Context Diagram

Si riporta nel seguito il Context Diagram, per comodità, già presentato e descritto nel documento precedente.



Component Diagram



Analisi e definizione delle componenti

In questa parte viene analizzata e definita la struttura in termini di componenti interni al sistema definiti sulla base dei requisiti analizzati nei precedenti documenti. La connessione tra le varie componenti viene rappresentata dal Component Diagram, mostrato in precedenza. Si identificano le interfacce delle componenti verso sistemi esterni e infine viene mostrato il livello di accoppiamento tra i componenti.

Sistema di autorizzazione

Motivazione: Determinate azioni, quali la prenotazione, la modifica della presenza ad una guida e la modifica della disponibilità in uno slot, sono operazioni che solo determinate persone sono autorizzate a fare. È questo il motivo per il quale è stato identificato un componente **Sistema autorizzazione**, che verificherà anche dal lato back-end che l'attore che richiede e svolge quella determinata azione sia effettivamente autorizzato a farla (e.g. la modifica della disponibilità è un'operazione che solo l'istruttore è autorizzato a fare).
Coesione: livello **7** - funzionale.
Spiegazione: Il componente in questione ha il compito di gestire tutto ciò che riguarda il controllo dei permessi e delle autorizzazione per eseguire particolari funzioni del sistema. Viene interpellato ogni qualvolta si richiede di effettuare una modifica per verificare se l'utente ha effettivamente il permesso di utilizzare quella particolare funzionalità.

Gestione prenotazione

Motivazione: La componente **Gestione prenotazioni** è al centro del progetto e soddisfa il **RF03** e **RF19**. Lo scopo di questa componente è quello di fornire tutte le informazioni necessarie al fine di modificare, inserire o visitare una o più prenotazioni. Allo stesso tempo, richiede informazioni riguardanti gli attori che partecipano alla registrazione delle prenotazioni e alla componente **Gestione DB** per mostrare la totalità delle informazioni nel calendario.
Coesione: livello **4** - procedurale.
Spiegazione: Il componente gestisce l'intera procedura che l'utente segue per eseguire/annullare la prenotazione, dopo l'avvenuta selezione dello slot su cui si intende fare quest'ultima. Infine potrà mostrare le prenotazioni effettuate.

Gestione presenze

Motivazione: È stato preso in considerazione il **RF15** con relativo mock-up. È stato dunque definito il componente **Gestione presenze** allo scopo di gestire le modifiche da parte dell'istruttore della presenza degli studenti ad una determinata guida o meno (ovvero se uno studente si è presentato alla guida che aveva prenotato oppure no).
Coesione: livello **7** - funzionale.
Spiegazione: Questo componente si occupa di tutto ciò che riguarda le presenze degli studenti, l'unica funzione che mette a disposizione è quella di confermare o meno una presenza, tutti i suoi elementi sono finalizzati allo svolgimento di questa operazione.

Gestione disponibilità

Motivazione: È stato preso in considerazione il **RF13**. È stato dunque definito il componente **Gestione disponibilità** allo scopo di gestire le modifiche da parte degli istruttori della propria disponibilità in un determinato slot nel calendario.
Coesione: livello **7** - funzionale.

Spiegazione: Questo componente permette di gestire la funzionalità che modifica la disponibilità dell'istruttore in un certo slot temporale. Il modulo gestisce la parte di selezione di uno slot non prenotato per permettere all'istruttore di indicare se in quella data e ora è disponibile o meno per una guida.

Calendario

Motivazione: Essendo il calendario l'anima di questo progetto, a cui si appoggiano la quasi totalità delle funzionalità, si definisce un componente **Calendario**, che è necessario per soddisfare i **RF02**, **RF03**, **RF12**, **RF13**, **RF19**, tramite apposite interfacce, differenti a seconda dell'operazione richiesta. Va inteso principalmente come il lato front-end, ovvero serve per visualizzare sullo schermo le prenotazioni e gli slot richiesti dall'utente in modo chiaro e intuitivo, ma gestisce anche le richieste di informazioni dei singoli slot da parte di altre componenti.
Coesione: livello **5** - comunicazionale.
Spiegazione: Il livello di coesione di questo componente è *comunicazionale*, perché segue una sequenza predefinita di operazioni da svolgere per eseguire le sue funzioni e il tutto avviene all'interno della stessa struttura dati, contenente ogni informazione necessaria alla visualizzazione/modifica delle prenotazioni.

Sistema di pagamento

Motivazione: Facendo riferimento a **RF09** e **RF22**, è stato definito il componente **Sistema pagamento** per gestire le richieste di pagamento delle prenotazioni da parte degli studenti o della segreteria sotto richiesta di questi ultimi.
Coesione: livello **7** - funzionale.
Spiegazione: Il modulo in questione ha livello di coesione *funzionale* in quanto il suo scopo è quello di permettere a chi ne fa uso di pagare una guida selezionata dall'utente tramite un portale di pagamenti esterno all'applicazione.

Gestione notifiche

Motivazione: Dati i **RF14** e **RF21**, è stato necessario definire un componente **Gestione notifiche**, che permettesse di gestire le richieste di invio notifica da parte del sistema allo studente o all'istruttore nel caso di prenotazione/annullamento di una guida, fornendo un'interfaccia apposita.
Coesione: livello **7** - funzionale.
Spiegazione: Questo componente svolge in autonomia tutto ciò che riguarda l'invio delle notifiche. Gli elementi sono coesi e finalizzati a portare a termine questa singola funzione senza appoggiarsi ad altre strutture dati.

Gestione istruttore

Motivazione: Il componente **Gestione istruttore** è pensato come un gestore dei dati degli istruttori, ergo è in grado di fornire tali informazioni tramite un'interfaccia apposita qualora richiesti, ad esempio, per una prenotazione.
Coesione: livello **6** - informazionale.
Spiegazione: A questo componente è stato attribuito un livello di coesione *informazionale,* in quanto si occupa di gestire i dati degli istruttori, i quali sono parte della stessa struttura dati. Questo modulo è in grado di fornire dati diversi in base alle richieste ricevute.

Ad esempio, i dati forniti al fine di effettuare una prenotazione saranno diversi da quelli richiesti per gestire la propria disponibilità.

Gestione studente

Motivazione: Il componente **Gestione studente** è pensato come un gestore dei dati degli studenti, ergo è in grado di fornire tali informazioni tramite un'interfaccia apposita nel caso in cui essi siano richiesti, per esempio, per una prenotazione. Inoltre, per realizzare i **RF07** e **RF20**, offre un'interfaccia apposita per la visualizzazione di tali dati in formato leggibile e intuitivo su una specifica pagina dell'applicazione.
Coesione: livello **6** - informazionale.
Spiegazione: A questo componente è stato assegnato il livello di coesione *informazionale*, poiché si occupa di gestire i dati di ogni studente, eseguendo funzioni diverse e tra loro indipendenti sulla stessa struttura dati.
Ad esempio, avrà a disposizione un metodo per fornire le informazioni relative alle prenotazioni e uno per mostrare il resoconto di ogni studente.

Gestione DB

Motivazione: Questa componente gestisce tutte le interazioni con il database, è pensato per creare, leggere, aggiornare ed eliminare (*CRUD*) i record del database interessati dalle componenti **Gestione Prenotazioni** e **Gestione Disponibilità**, su richiesta degli specifici utenti abilitati a svolgere queste operazioni.
Coesione: livello **4** - procedurale.
Spiegazione: Questa componente si occupa di interagire direttamente con il database; essa ha un flusso di operazioni da seguire, nel quale coordina tutti i suoi elementi, per portare a termine con successo le sue operazioni. Per offrire le sue funzionalità ad un altro componente deve prima cercare, prelevare e organizzare le informazioni in suo possesso. Per questo motivo, il livello di coesione è quello *procedurale*.

Tabella riassuntiva dei livelli di coesione

Componenti/ Livello di coesione	Casuale	Logica	Temporale	Procedurale	Comunicazionale	Informazionale	Funzionale
Sistema di autorizzazione							X
Gestione prenotazioni				X			
Gestione presenza							X
Gestione disponibilità							X
Calendario					X		
Sistema di pagamento							X
Gestione notifiche							X
Gestione istruttore						X	
Gestione studente						X	
Gestione database				X			

Livello di accoppiamento delle componenti

Accoppiamento componenti	Sistema di autorizzazione	Gestione prenotazioni	Gestione presenza	Gestione disponibilità	Calendario	Sistema di pagamento	Gestione notifiche	Gestione istruttore	Gestione studente	Gestione database
Sistema di autorizzazione	**X**	Stamp	Stamp	Stamp						
Gestione prenotazioni		**X**	Data		Content	Stamp	Stamp	Stamp	Stamp	Data
Gestione presenza			**X**							Data
Gestione disponibilità				**X**	Data			Stamp		Data
Calendario					**X**					
Sistema di pagamento						**X**				Stamp
Gestione notifiche							**X**			
Gestione istruttore								**X**		
Gestione studente									**X**	
Gestione database										**X**

Di seguito le motivazioni che ci hanno portati a definire i livelli di accoppiamento tra i vari componenti.

Sistema di autorizzazione - Gestione prenotazione – Livello *Stamp*
Spiegazione: I moduli si scambiano dati riguardo l'id dell'utente per verificare l'autorizzazione per accedere a questa funzione del sistema.

Sistema di autorizzazione - Gestione presenze – Livello *Stamp*
Spiegazione: I moduli si scambiano dati riguardo l'id dell'utente per verificare l'autorizzazione per accedere a questa funzione del sistema.

Sistema di autorizzazione - Gestione disponibilità – Livello *Stamp*
Spiegazione: I moduli si scambiano dati riguardo l'id dell'utente per verificare l'autorizzazione per accedere a questa funzione del sistema.

Gestione prenotazione - Gestione presenza– Livello *Data*
Spiegazione: La componente gestione prenotazione passerà alla gestione presenza la struttura dati contenente la prenotazione alla quale modificare la presenza di uno studente.

Gestione prenotazione - Calendario – Livello *Content*
Spiegazione: Il componente calendario fa diretto riferimento al contenuto del componente gestione prenotazione, in quanto quest'ultima contiene tutte le informazioni necessarie a svolgere tutte le funzionalità del calendario.

Gestione prenotazione - Sistema di pagamento – Livello *Stamp*
Spiegazione: I moduli si scambiano solo i dati necessari per definire la guida per la quale è necessario il pagamento.

Gestione prenotazione - Gestione notifiche – Livello *Stamp*
Spiegazione: I moduli si scambiano solo i dati necessari per comporre ed inviare la notifica agli interessati della prenotazione (istruttore e studente).

Gestione prenotazione - Gestione istruttore – Livello *Stamp*
Spiegazione: Il componente per la gestione dell'istruttore passerà alla gestione prenotazione solo alcuni dati dell'istruttore, necessari per la prenotazione.

Gestione prenotazione - Gestione studente – Livello *Stamp*
Spiegazione: Il componente per la gestione dell'istruttore passerà alla gestione prenotazione solo alcuni dati dello studente, necessari per la prenotazione.

Gestione prenotazione - Gestione database – Livello *Data*
Spiegazione: I moduli si passano gli argomenti e le strutture dati necessari per inserire/rimuovere una prenotazione nel database.

Gestione presenze - Gestione database – Livello *Data*

Spiegazione: I moduli si passano gli argomenti e le strutture dati necessari per aggiornare le presenze ad una prenotazione nel database.

Gestione disponibilità - Calendario – Livello *Data*
Spiegazione: Il componente per la gestione della disponibilità riceverà dal calendario lo slot a cui modificare la stessa.

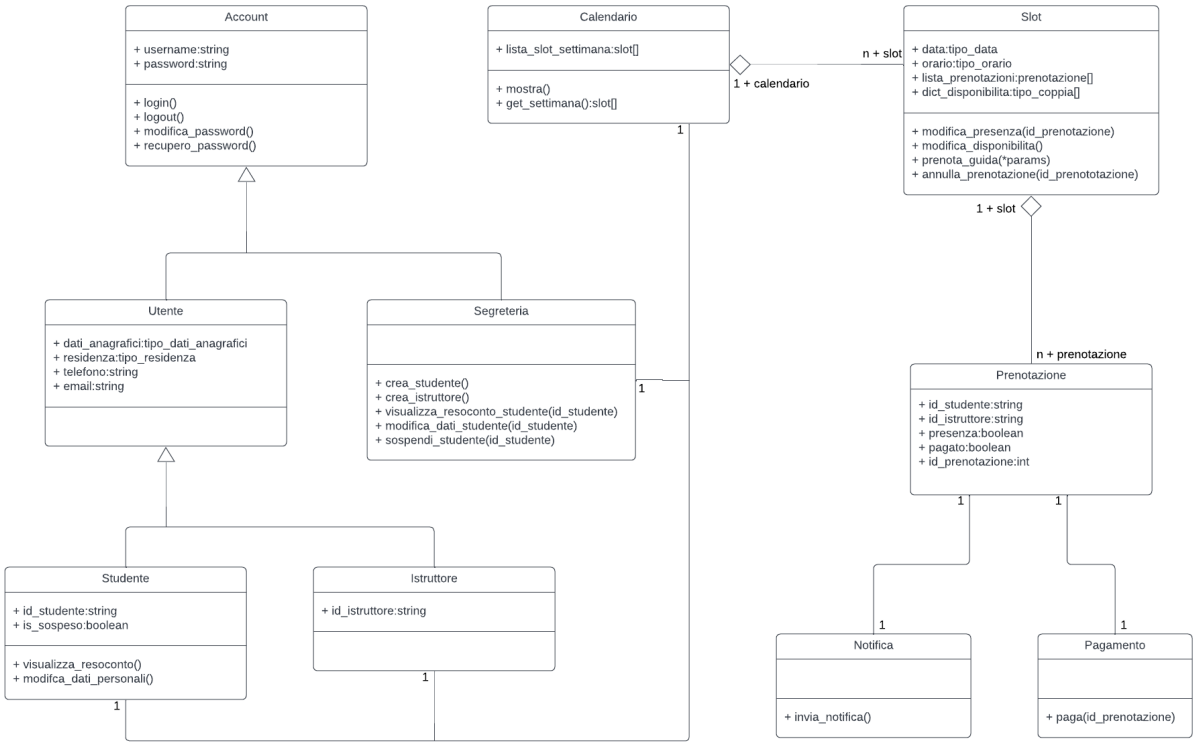
Gestione disponibilità - Gestione istruttore – Livello *Stamp*
Spiegazione: Il componente per la gestione dell'istruttore passerà alla gestione disponibilità solo alcuni dati dell'istruttore, necessari per la modifica.

Gestione disponibilità - Gestione database – Livello *Data*
Spiegazione: I moduli si passano gli argomenti e le strutture dati necessari per aggiornare la disponibilità di un istruttore nel database.

Sistema di pagamento - Gestione database – Livello *Stamp*
Spiegazione: I moduli si passano solo alcuni dei dati che sono stati necessari per effettuare un pagamento, per rendere possibile la modifica degli stessi all'interno del database.

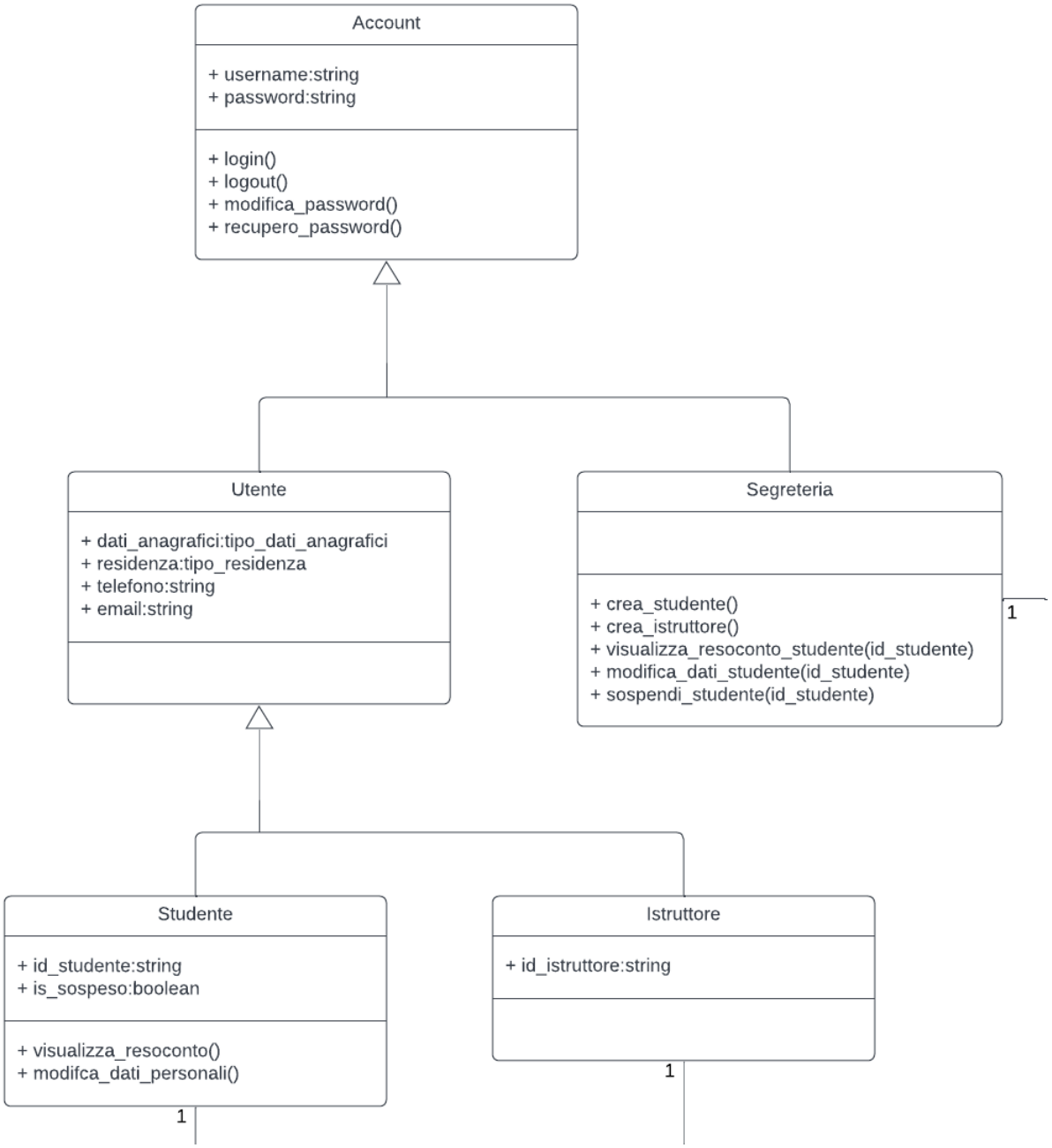
Class Diagram

In questo capitolo vengono presentate le diverse classi che compongono la nostra applicazione "Book & Drive". Le componenti rappresentate nel capitolo precedente vengono ora rappresentate in una o più classi che gestiscono le funzione e le istanze che modellano il progetto.
Le classi individuate sono caratterizzate da un nome, una lista di attributi (se necessari) per descrivere le particolarità di ciascuna classe e una lista di metodi (ove presenti) che definiscono i servizi messi a disposizione da ciascuna classe e/o ne modificano gli attributi. Le relazioni, che collegano le classi, indicano le associazioni tra ognuna e rappresentano la possibilità di fornire e modificare informazioni interclasse.
Riportiamo di seguito le classi individuate a partire dai diagrammi di contesto e da quello dei componenti. In questo processo si è proceduto anche nel massimizzare la coesione e minimizzare l'accoppiamento tra classi.



Account

Gli attori individuati per questa applicazione, ovvero ***Segreteria***, ***Studente*** e ***Istruttore***, sono identificati da classi diverse, poiché hanno alcuni campi e metodi distinti.
Dato che determinati campi e metodi sono comuni a tutti e tre, è stata sfruttata la generalizzazione, come descritta in seguito.
La classe ***Account***, astratta, contiene le informazioni che sono comuni a tutti e tre, legate al processo di autenticazione (*username*, *password*, e metodi relativi).
La classe ***Segreteria***, in aggiunta, ha la possibilità di creare studenti e istruttori, visualizzarne il resoconto, modificarne i dati e lo stato.
Le classi ***Studente*** e ***Istruttore***, che si distinguono per l'*id* e *is_sospeso* e alcuni metodi, sono a loro volta accomunate da altri dati (es: *anagrafica*, *residenza,* ecc), motivo per cui è stata aggiunta un'ulteriore generalizzazione, la classe ***Utente***.

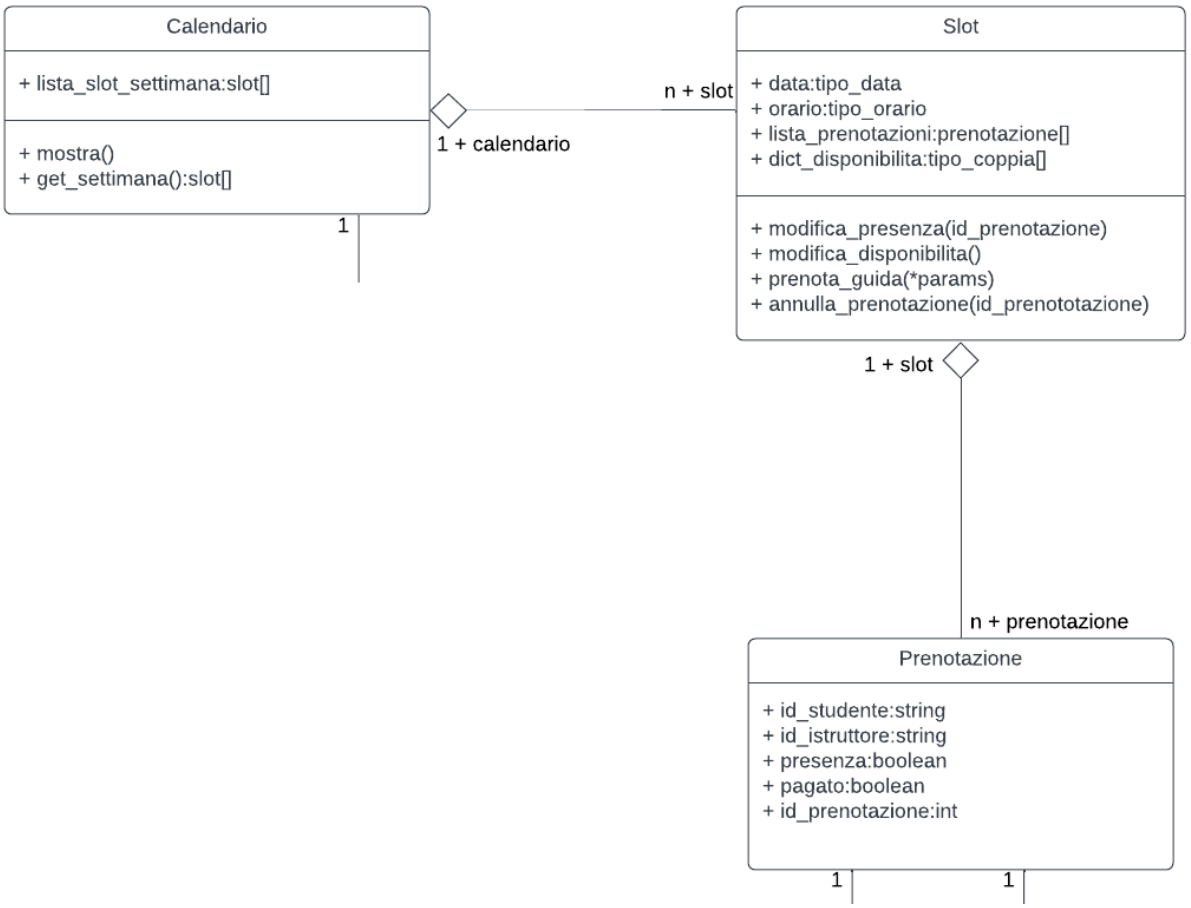


Calendario, Slot, Prenotazione

Ogni attore ha la possibilità di visualizzare e interagire con un'istanza di **Calendario**.
Il ***Calendario*** è un'aggregazione di ***Slot***, classe che permette di gestire le prenotazioni (*modifica_presenza*, *modifica_disponibilita*, *prenota/annulla guida*), come è possibile vedere nell'immagine sottostante.
Lo ***Slot*** è a sua volta un'aggregazione di ***Prenotazione***, classe che contiene i campi relativi alla prenotazione di una guida, e da cui è possibile gestire il pagamento e l'invio di notifiche, come spiegato nel seguito di questo documento.

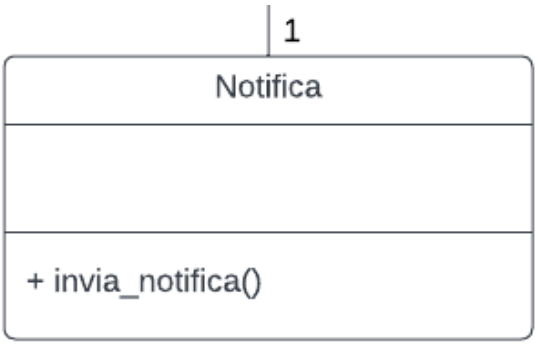
Nota 1: al metodo *prenota_guida* di ***Slot***, è necessario passare dei parametri **params*.
Questi parametri sono quelli necessari per istanziare una ***Prenotazione*** (*id_studente*, *id_istruttore*)

Nota 2: Nella classe ***Slot*** abbiamo istanziato una lista di ***tipo_coppia*** (vedi definizione nel class diagram completo a inizio capitolo), grazie alla quale teniamo traccia delle coppie istruttore-disponibilità per indicare quando un istruttore è disponibile all'interno di quello slot (implementando una sorta di dizionario).



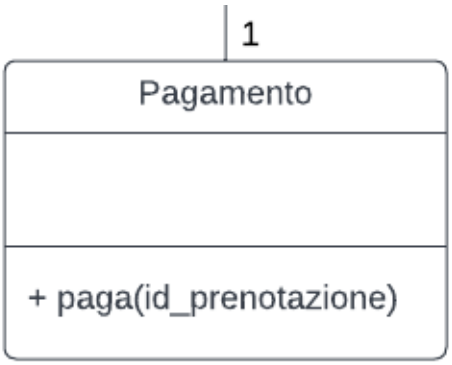
Notifica

E' stata definita la classe ***Notifica*** che, legata in una molteplicità 1 a 1 con una determinata prenotazione, fornisce il metodo *invia_notifica()* per notificare lo studente e l'istruttore dell'avvenuta prenotazione. Nella nostra applicazione, l'invio effettivo della notifica verrà svolto da un sistema esterno, ergo questa classe non implementa realmente la funzionalità (è solo una wrapper).



Pagamento

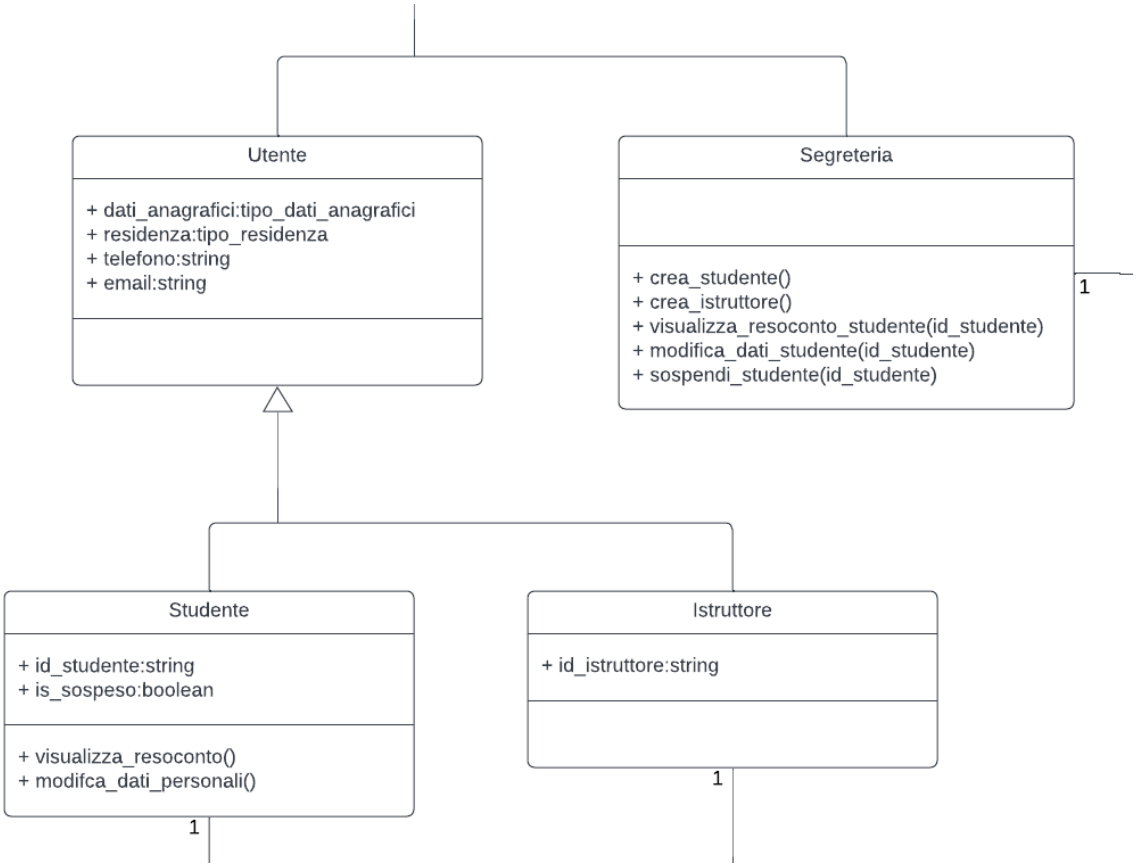
E' stata definita la classe ***Pagamento*** che, legata in una molteplicità 1 a 1 con una determinata prenotazione, fornisce il metodo *paga()* a cui passare l'*id_prenotazione* della guida da pagare. Nella nostra applicazione, il pagamento effettivo verrà svolto da un sistema esterno, ergo questa classe non implementa realmente la funzionalità (è solo una wrapper).



OCL

Sospensione studente

Successivamente all'invocazione del metodo *sospendi_studente*() da parte della classe ***Segreteria*** su uno specifico studente, individuato attraverso il suo *id_studente*, quest'ultimo vedrà il suo attributo *is_sospeso* cambiato a "true". Questo significa che lo studente non può utilizzare alcuna funzione perché il suo account è sospeso.



La condizione sopra descritta è espressa in ocl attraverso la seguente postcondizione:

```
context Segreteria::sospendi_studente(id_studente)
post: studente.is_sospeso = true
```

Annullamento prenotazione solo se prenotazione effettuata

All'interno della classe ***Slot*** abbiamo un vincolo *annulla_prenotazione*(). Infatti, come è possibile vedere nell'immagine sottostante, una prenotazione può essere annullata se e solo se l'id dello studente è tra quelli che hanno una prenotazione in quello slot (da notare che uno slot può avere più studenti, in base al numero di istruttori disponibili).

```
context Slot:: annulla_prenotazione(id_prenotazione)
pre: id_studente IN lista_prenotazioni[ ].id_studente
```

Modifica presenza guida

Utilizzando la funzione *modifica_presenza*() della classe ***Slot***, necessaria per indicare se uno studente si è presentato alla guida o meno, viene modificato l'attributo *presenza* di ***Prenotazione***, che invertirà il suo valore.

```
context Slot::modifica_presenza(id_prenotazione)
post: prenotazione.presenza = NOT(prenotazione.presenza)
```

Modifica disponibilità solo se non c'è nessuna guida nello stesso slot

Con questo vincolo viene indicato che un istruttore può modificare la sua disponibilità in uno specifico slot solo se il suo id non è presente tra gli istruttori impegnati nella guida in quel giorno e in quella fascia oraria.
Questa funzione opera nella classe ***Slot*** e il codice del OCL viene riportato qui sotto.

```
context Slot:: modifica_disponibilita()
pre: id_istruttore NOT IN lista_prenotazioni[ ].id_istruttore
```

Prenotazione disponibile in giornata con un'ora di preavviso

Per prenotare una lezione di guida è necessario che l'invariante riportata sotto risulti soddisfatta, ossia che la *data* e l'*orario* che si vuole prenotare sia maggiore di almeno 1 ora rispetto al momento della prenotazione.

```
context Slot::prenota_guida(*params) inv:
slot.data > data.today OR slot.data == data.today
AND slot.orario >= orario.now + 1h
```

Prenotazione guida disponibile solo se studente non sospeso

Perchè uno slot possa essere prenotato da parte di uno studente, è necessario che tale studente abbia il parametro *is_sospeso*=false per indicare appunto che al suo account è permesso usufruire delle funzioni offerte dal sistema. Questa precondizione fa riferimento alla classe ***Slot*** ed è espressa in OCL attraverso il codice riportato qui sotto.

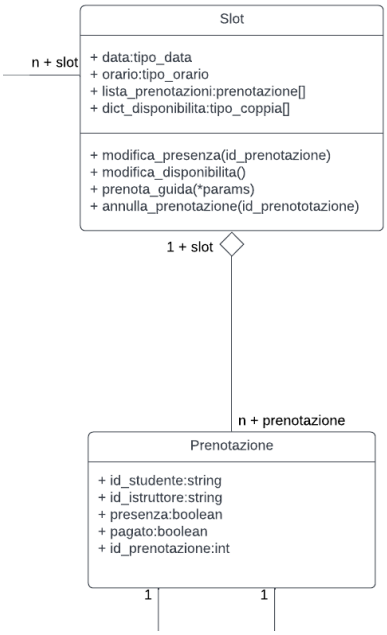
```
context Slot: prenota_guida(*params)
pre: studente.is_sospeso == false
```

Prenotazione guida solo con almeno un istruttore disponibile

L'ultima precondizione della classe ***Slot*** indica che per prenotare un certo slot è necessario che il numero degli istruttori disponibili (la disponibilità è indicata dal secondo valore del dizionario *dict_disponibilità*), sia diverso da 0.
La condizione è riportata qui sotto:

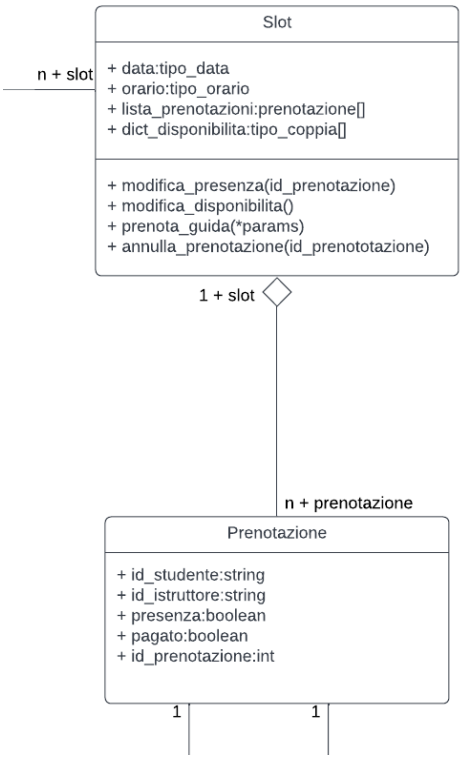
```
context Slot::prenota_guida(*params)
pre: dict_disponibilita.count(disponibile == true) != 0
```

I codici degli OCL qui sopra descritti fanno riferimento alle due classi riportate di seguito:



Numero massimo prenotazioni minore-uguale a numero istruttori disponibili

L'invariante, in OCL, qui riportata indica che qualsiasi sia la dimensione della lista *lista_prenotazioni*, il numero di elementi presenti sarà sempre inferiore al numero di istruttori registrati alla scuola guida.

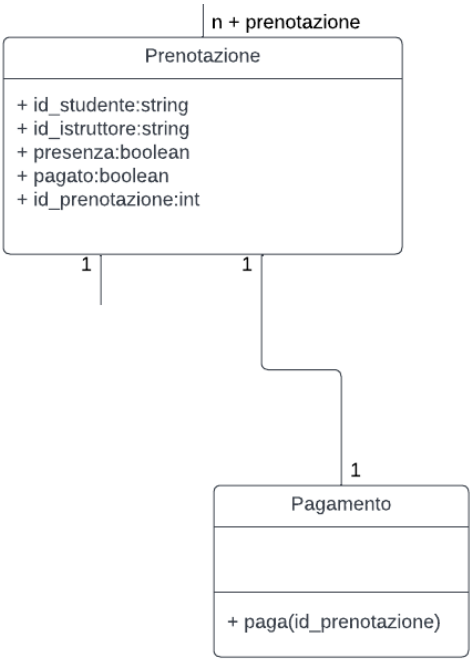



```
context lista_prenotazioni[ ] inv:
lista_prenotazioni[ ].size() <= #istruttori
```

Aggiornamento attributo pagato dopo pagamento

Con questa postcondizione viene indicato che, dopo l'avvenuto pagamento di una guida prenotata, mediante la classe ***Pagamento*** e il suo metodo, l'attributo *pagato* della prenotazione cambia il suo valore da false a true.

Il codice OCL viene riportato qui sotto:




```
context Pagamento:: paga(id_prenotazione)
post: prenotazione.pagato = true
```

Class Diagram completo con OCL