ShuttleLive

Software per autisti/clienti che vogliono offrire o usufruire di servizi taxi e navetta

Studenti:

Antonio Inveninato: 1000055701

Angelo Cocuzza: 1000055700

Sommario

Introduzione 3
Iterazione 1 4
Iterazione 2 13
Iterazione 3 21
Iterazione 4 26
Iterazione 5 30

Introduzione

L' applicazione ShuttleLive offre un servizio di prenotazione viaggi tramite bus/navetta o automobili privati.

Gli autisti si registrano alla piattaforma specificando vari dati anagrafici, tipo di patente, la città in cui opera e i mezzi a disposizione indicando marca, targa e numero posti per ognuno di essi. L'autista può creare un viaggio con una meta prestabilita indicando luogo di partenza, la meta di destinazione e la data di partenza e ritorno e specificando un eventuale evento. Gli autisti in alternativa possono offrire un servizio taxi/navetta mettendosi a disposizione del cliente in un determinato luogo e in un determinato lasso di tempo.

Il cliente si registra con dati anagrafici. Esso può cercare un viaggio organizzato in base all'evento che gli interessa o alla data di partenza e gli verranno forniti tutti i viaggi organizzati filtrati. Può anche cercare autisti per il servizio taxi/navetta inserendo data e ora, luogo di partenza e destinazione. Verranno scremati gli autisti disponibili e presenti nelle vicinanze del luogo di partenza. In questa lista vi può essere un autista che mette a disposizione più mezzi in modo tale che il cliente può scegliere quello più adatto alle sue esigenze.

Nei viaggi con meta prestabilita il prezzo è fisso per la singola persona, mentre per il servizio taxi/navetta il prezzo è proporzionale alla distanza percorsa e al numero di posti del mezzo a disposizione a cui verranno poi imposti degli sconti in base varie condizioni.

Iterazione 1

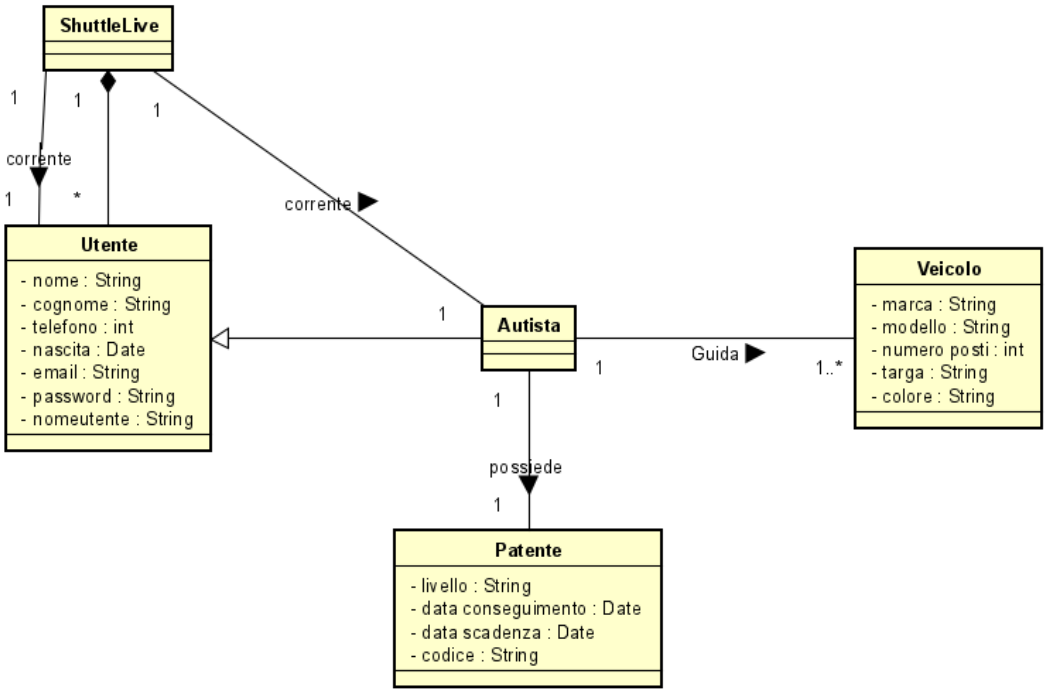
Nella prima iterazione si è scelto di effettuare l'analisi, la progettazione, l'implementazione e il testing dei casi d'uso UC1: Registrazione nuovo Utente/Autista e UC9: login Utente/Autista.

Per l'implementazione abbiamo cercato di utilizzare tutti i pattern GRASP creator, low copling, information expert, controller (in modo tale che ogni chiamata dalla Ui venga gestita da ShuttleLive), high coesion. Abbiamo anche utilizzato il pattern GoF Singleton applicato per impedire la creazione di più istanze della classe ShuttleLive.

MODELLO DI ANALISI

MODELLO DI DOMINIO

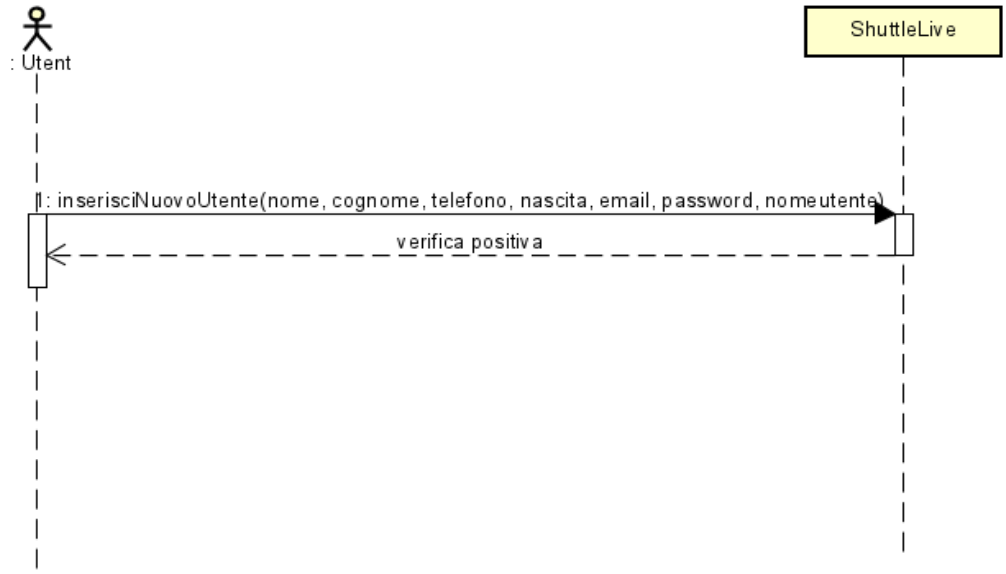
Di seguito è riportato il modello di dominio relativo ai casi d'uso presi in considerazione



SSD

Di seguito vengono illustrati gli ssd dei casi d'uso elaborati in questa iterazione

SSD UC1_1

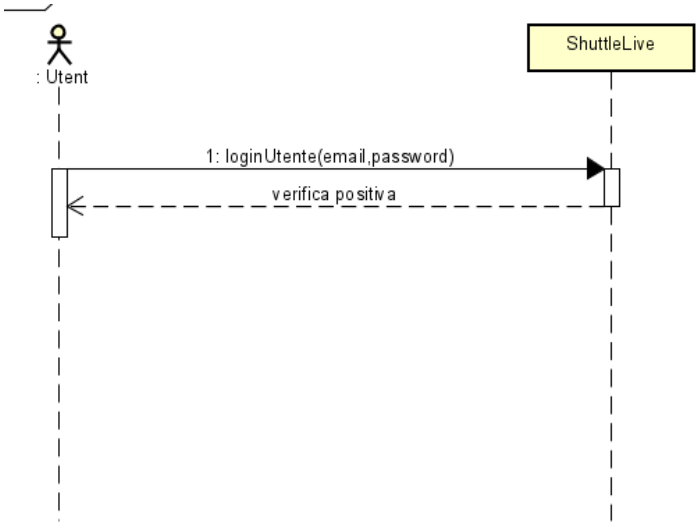


La prima operazione che viene effettuata è quella di **inserisciNuovoUtente**, in cui si passano tutte le informazioni necessarie per poter registrare un nuovo utente nel sistema

SSD UC1_2

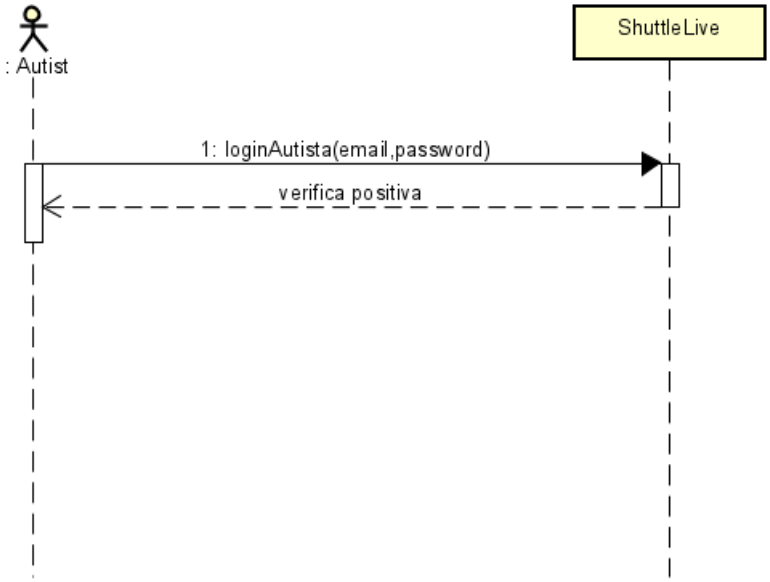
La prima operazione che viene effettuata è quella di **inserisciNuovoaAutista**, in cui si passano tutte le informazioni necessarie per poter registrare un nuovo autista nel sistema, successivamente si effettuano in sequenza l'operazione **inserisciPatente** nella quale passiamo tutte le informazioni per registrare la patente nell'autista che la sta inserendo e un operazione analoga viene fatta con **inserisciVeicolo.**

SSD UC9_1



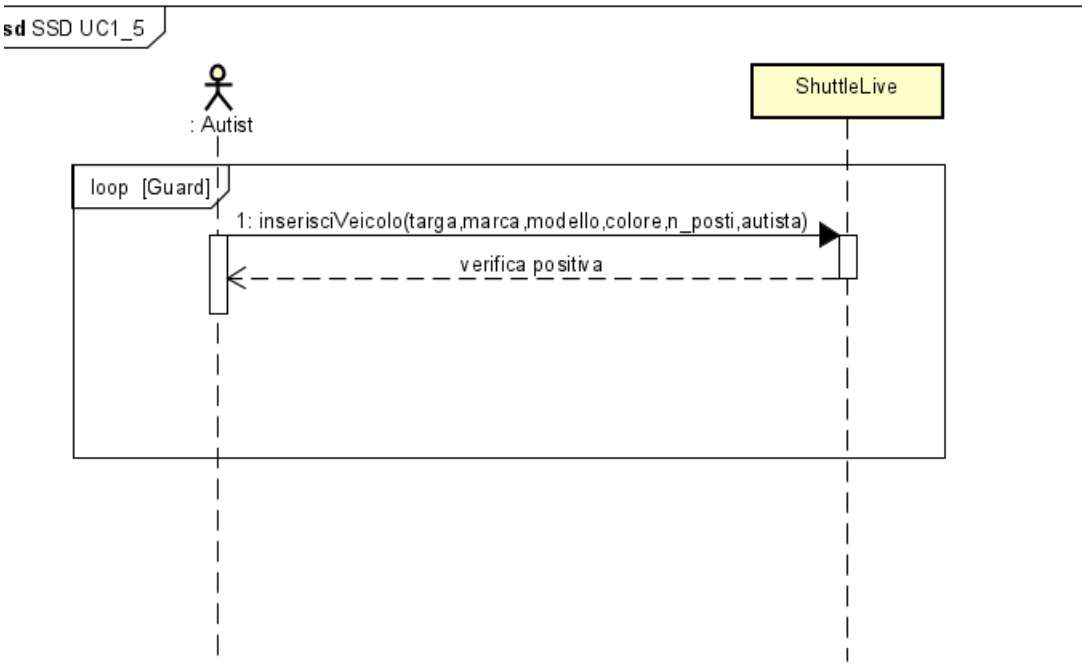
La prima operazione che viene effettuata è **loginUtente** con la quale si autentica un utente sulla base di email e passord passati

SSD UC9_2



La prima operazione che viene effettuata è **loginAutista** con la quale si autentica un autista sulla base di email e password passati .

SSD UC1_3



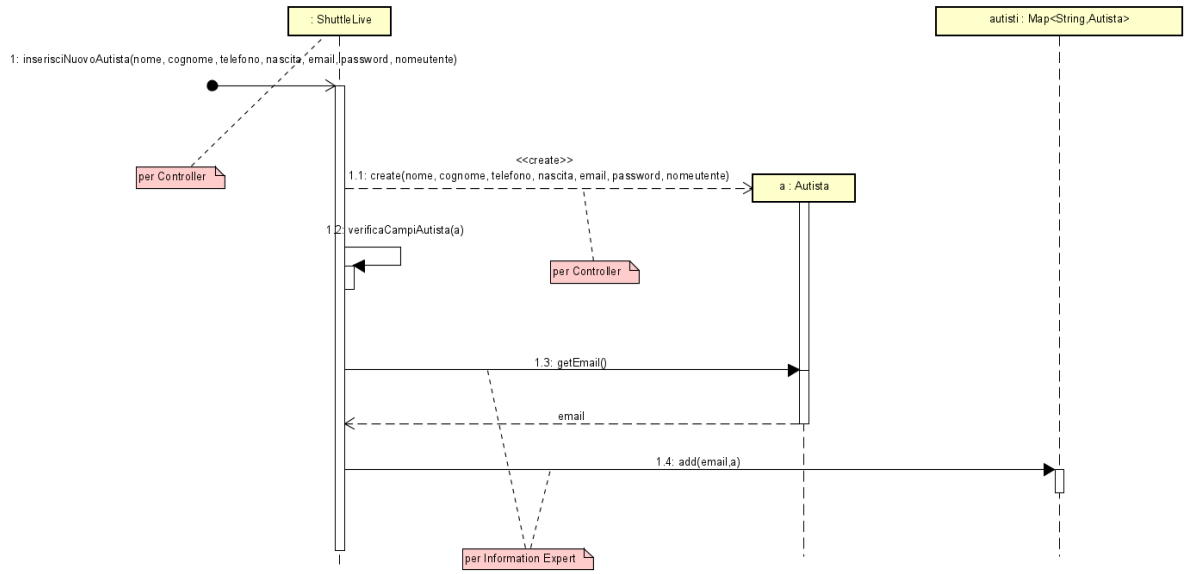
La prima operazione che viene effettuata è quella di **inserisciNuovoVeicolo**, in cui si passano tutte le informazioni necessarie per poter registrare un nuovo veicolo per l'autista che vuole registrare tale

MODELLO DI PROGETTO

Di seguito vengono illustrati gli sd dei casi d'uso elaborati in questa iterazione

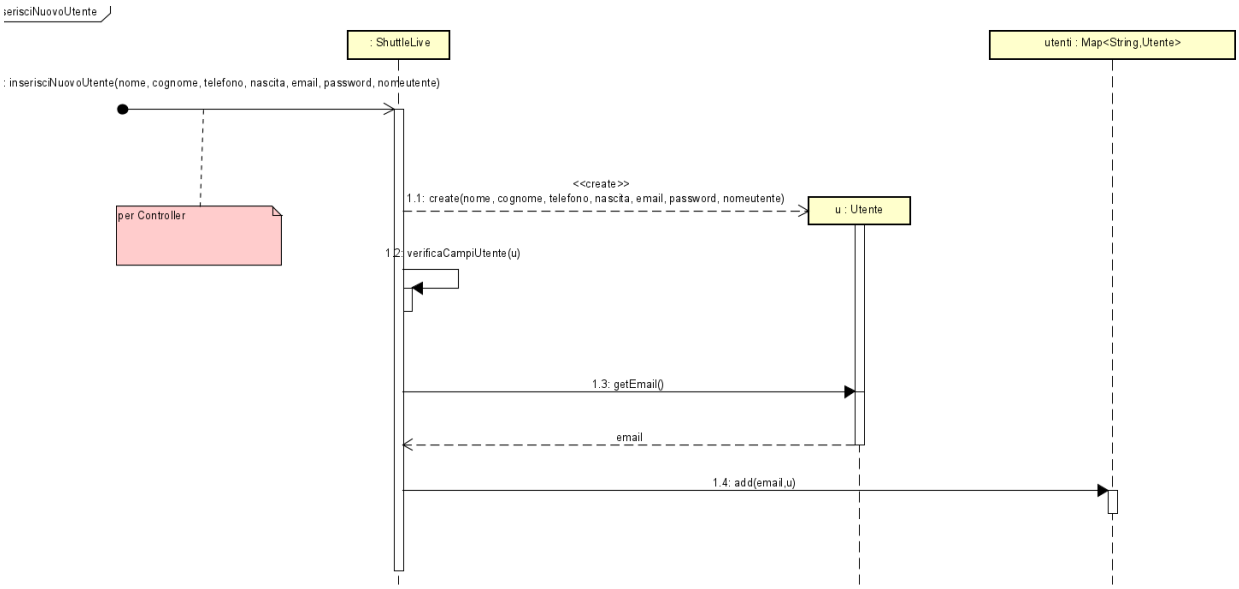
veicolo.

SD_1



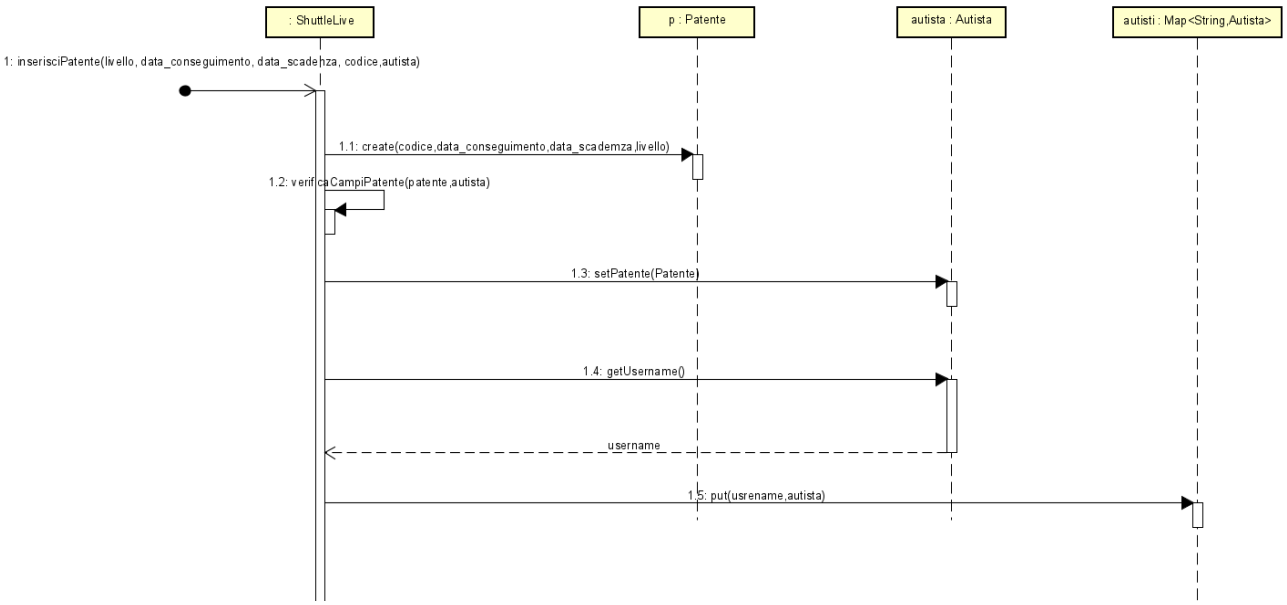
A ShuttleLive vengono passate dallo strato della UI le informazioni necessarie alla creazione di nuovo Autista. Successivamente tramite la funzione verificaCampiAutista verifichiamo che la mail inserita dall'autista non sia già presente nel sistema, tramite questo metodo viene creata una nuova istanza di , a cui ci si riferisce in ShuttleLive tramite l'associazione *"corrente"*. Infine viene inserito nella mappa di autisti che fa da registri di sistema;

SD_2



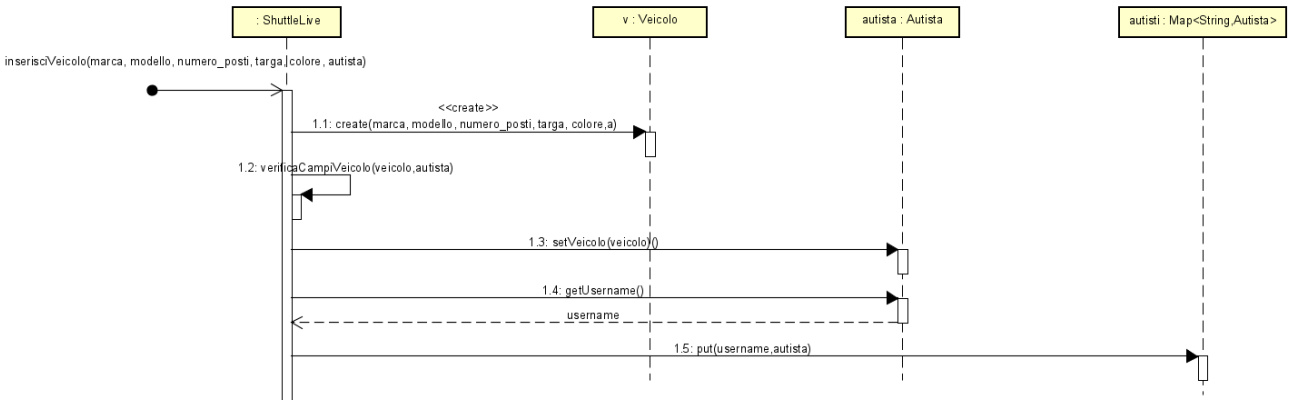
Questo sd è analogo al precedente solo dal punto di vista utente

SD_3



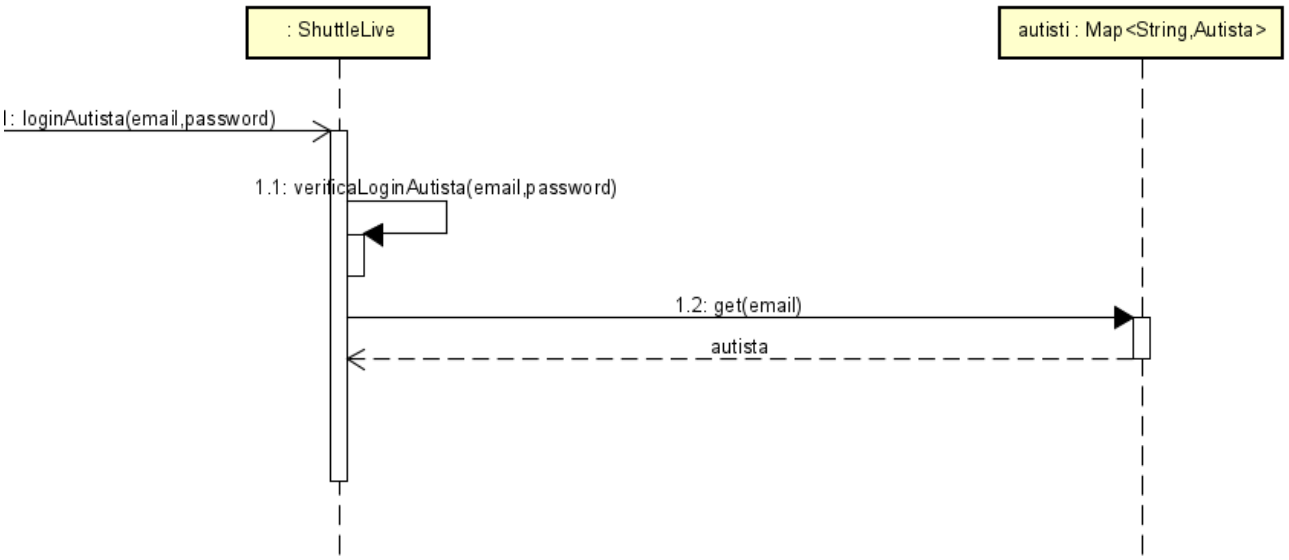
A ShuttleLive vengono passate da parte dell'autista dallo strato della UI le informazioni necessarie alla creazione di una nuova Patente. Successivamente tramite la funzione verificaCampiPatente dopo aver verificato i campi della patente settiamo la patente all'interno dell'autista che vuole registrare la patente che corrisponde con l'autista corrente, infine, l'autista viene aggiornato all'interno del registro di autisti.

SD_4



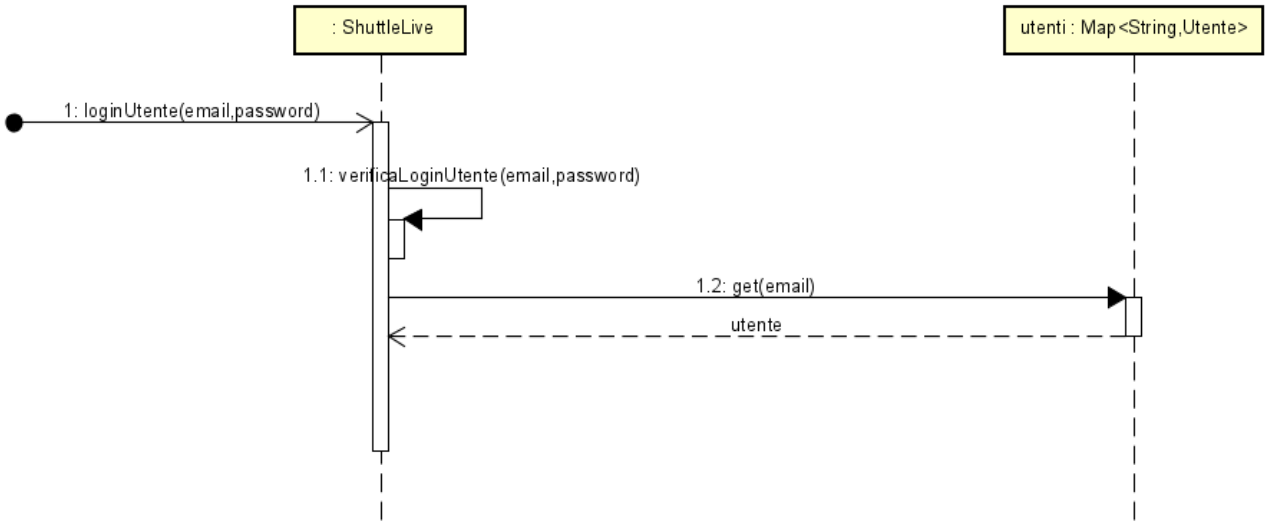
Analogo del precedente con inserimento di un veicolo all'interno di un autista

SD_5



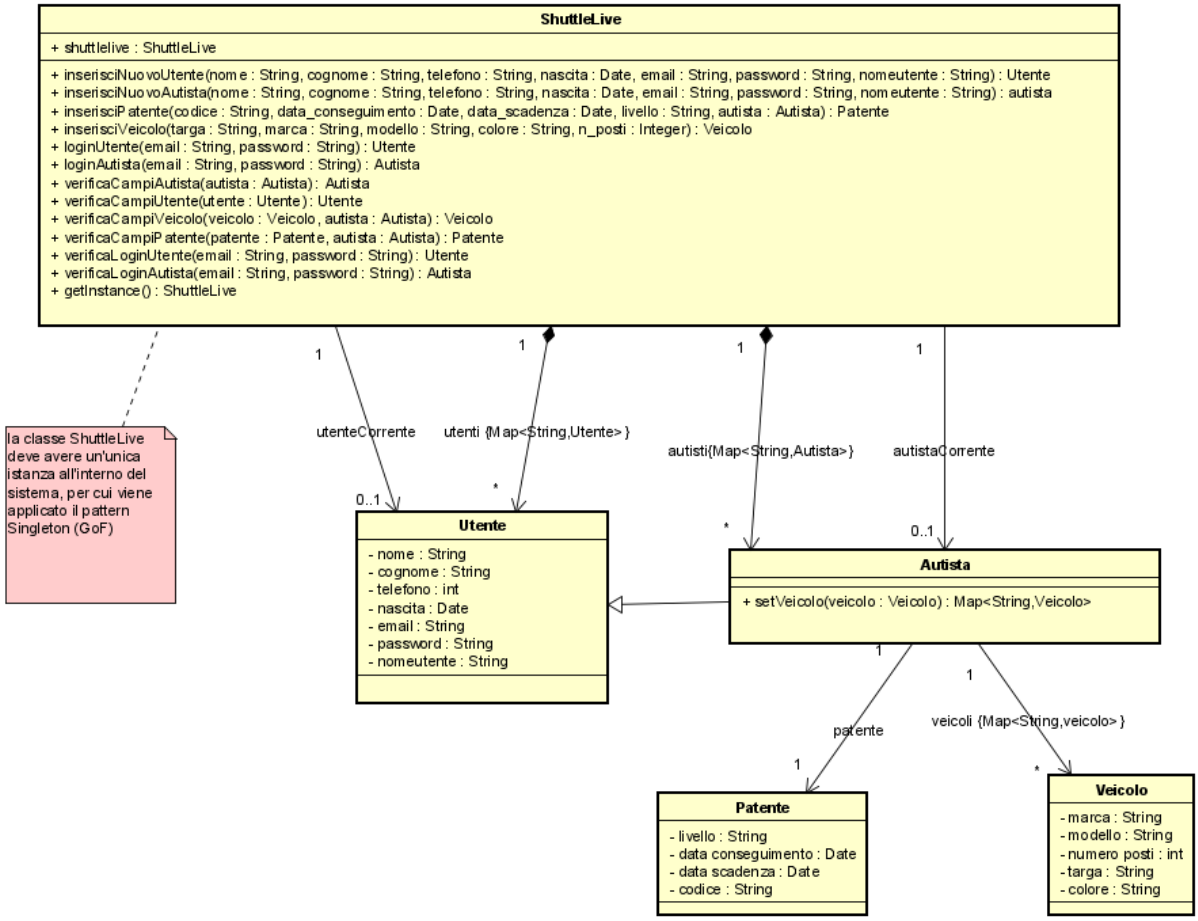
Tramite l'operazione di **loginUtente** diamo accesso all'utente identificato da email e password questa operazione richiama l'operazione **verificaLoginAutista**. Che dopo aver effettuato una verifica dei campi introdotti ricerca all'interno della mappa *autisti* e restituisce l'autista richiesto se trova una corrispondenza impostandolo come autista corrente

SD_6



Questo sd è identico al precedente dal punto di vista utente

Diagramma delle classi



Test

Tutti i test saranno effettuati tramite le funzioni asssertNotNull() che si assicura che il valore passatogli non sia nullo assertNull() che si assicura sia nullo e assertEquals() che confronta i messaggi di errore

Nelle funzioni **testInserisciNuovoUtente e testInserisciNuovoAutista** viene testato che l'inserimento sia andato a buon fine, inoltre vengono effettuati diversi test sui campi passati a tale funzione: viene assicurato che nessun campo venga lasciato vuoto, che la password abbia un numero di caratteri non inferiore a 8 e che l'email o la password non siano già stati inseriti data l'unicità.

Nella funzione **testInserisciVeicolo** viene testato che l'inserimento sia andato a buon fine, inoltre vengono effettuati diversi test sui campi passati a tale funzione: viene assicurato che nessun campo venga lasciato vuoto, che il numero di posti del veicolo sia maggiore di zero, che la targa abbia un numero di caratteri pari a 7 e che la targa non sia già stata inserita.

Nella funzione **testInserisciPatente** viene testato che l'inserimento sia andato a buon fine, inoltre viene assicurato che nessun campo venga lasciato vuoto.

Nelle funzioni **testLoginUtente** e **testLoginAutista** viene testato che inserendo le corrette credenziali il login va in porto tornando un utente,inoltre vengono fatti ei controlli sui campi del

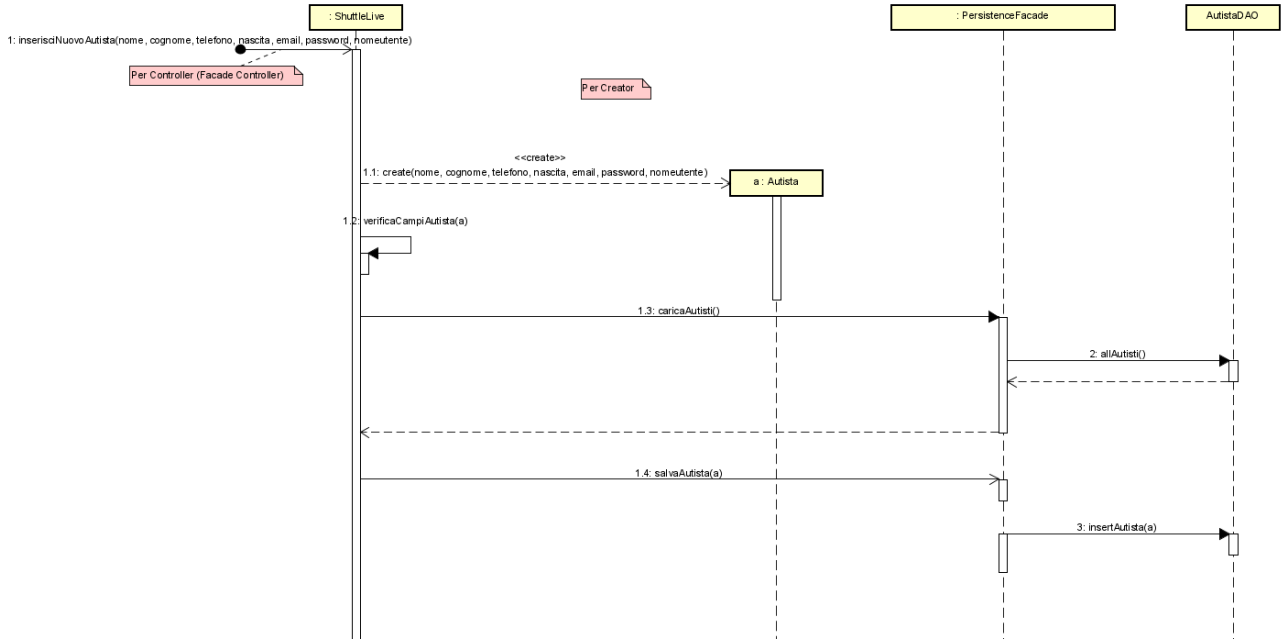
login : viene assicurato che nessun campo sia lasciato vuoto, che la password abbia un numero di caratteri maggiore o uguale a 8 e che fornendo Le credenziali sbagliate l’utente ritornato sia nullo;

Iterazione 2

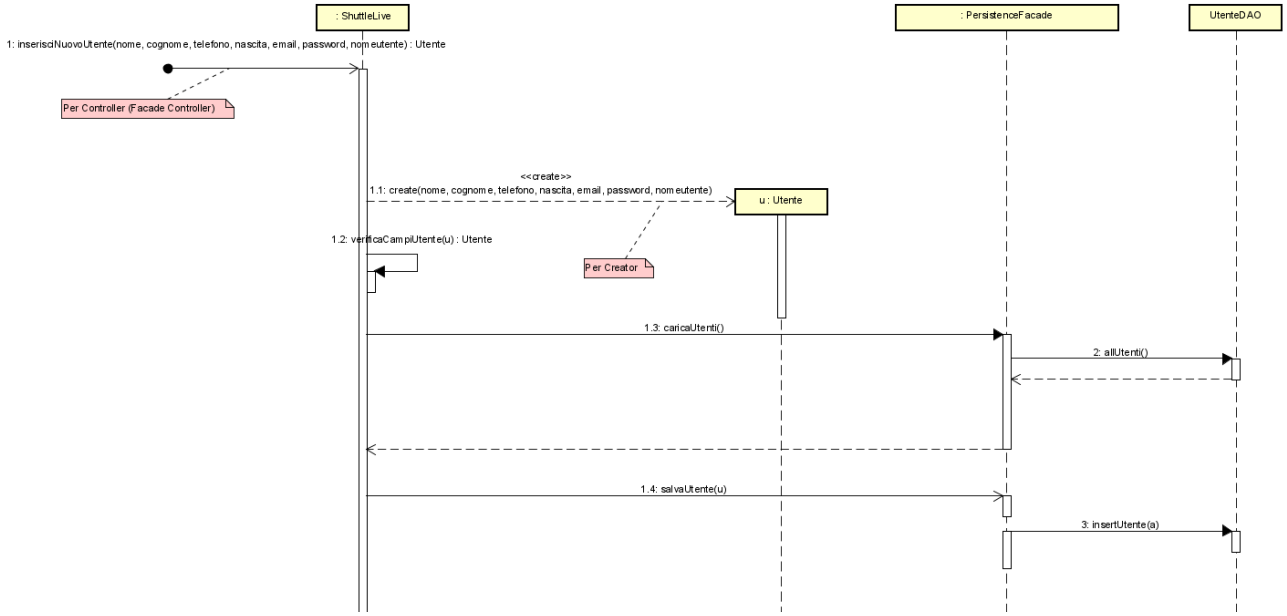
Nella seconda iterazione si è valutato di effettuare l'analisi, la progettazione, l'implementazione e il testing dei casi d'uso UC2: Elabora prenotazione taxi, UC10: Aggiungi disponibilità.

Alla fine della prima iterazione dato il possibile utilizzo concorrente da parte di molti utenti e autisti differenti si è optato per l'utilizzo del database e di conseguenza del pattern DAO che permette di scollegare le classi del dominio da quelle che si occupano di interfacciarsi con il database. Si è inoltre aggiornato il diagramma delle classi e i diagrammi di sequenza relativi alla prima iterazione. Mostrati di seguito:

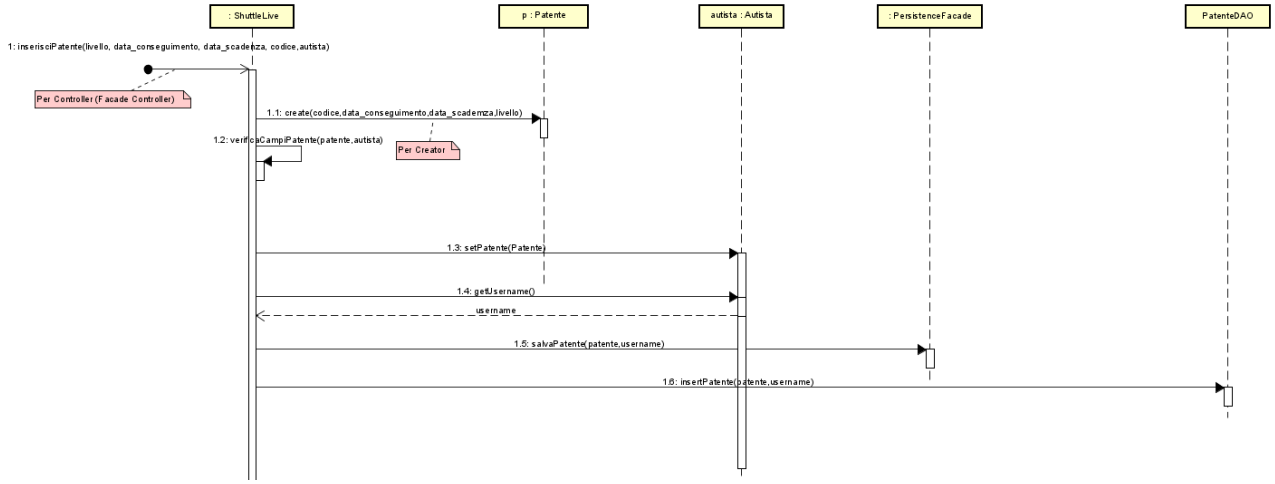
SD_1 inserisciNuovoAutista



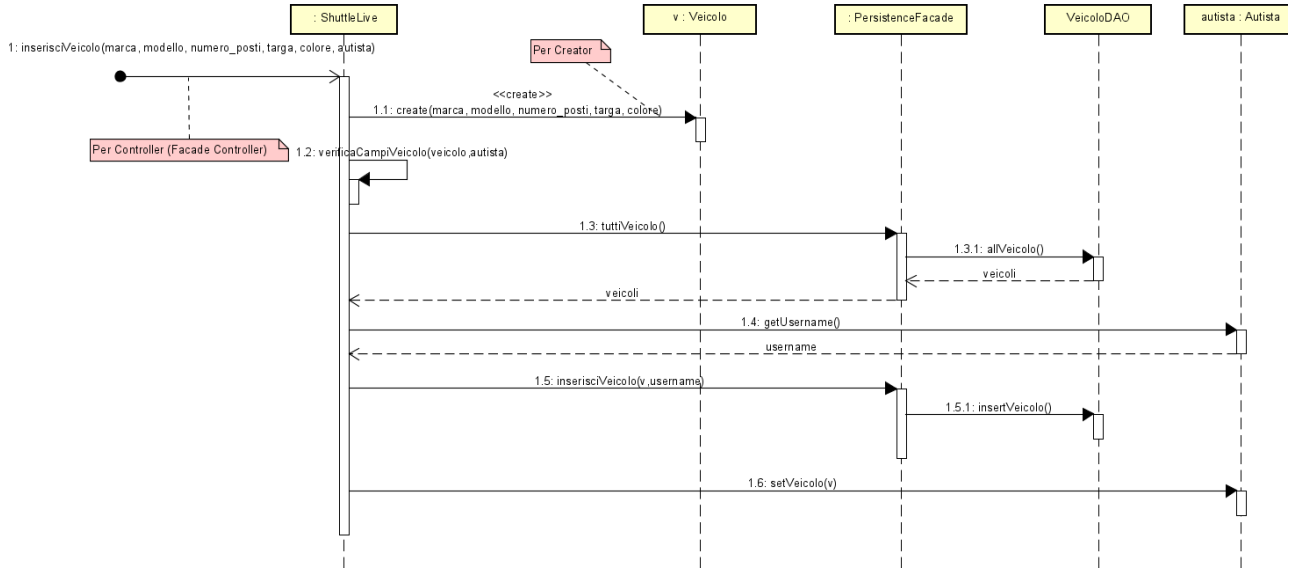
SD_2 inserisciNuovoUtente



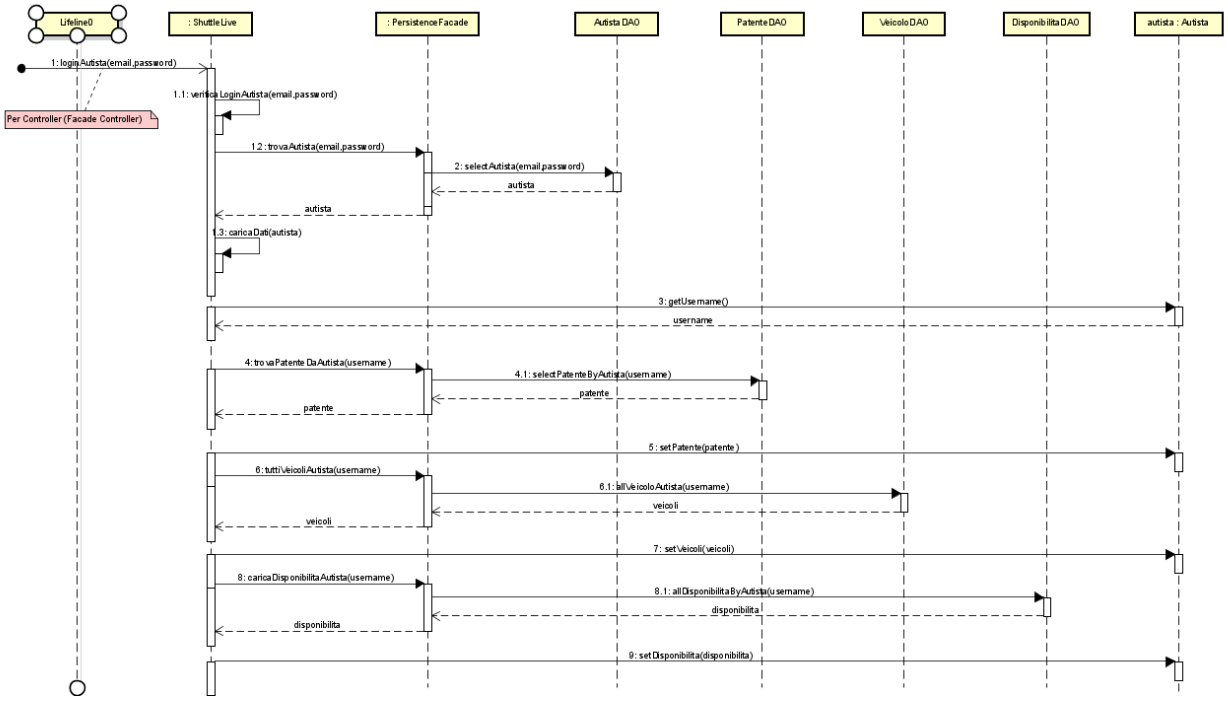
SD_3 InserisciPatente



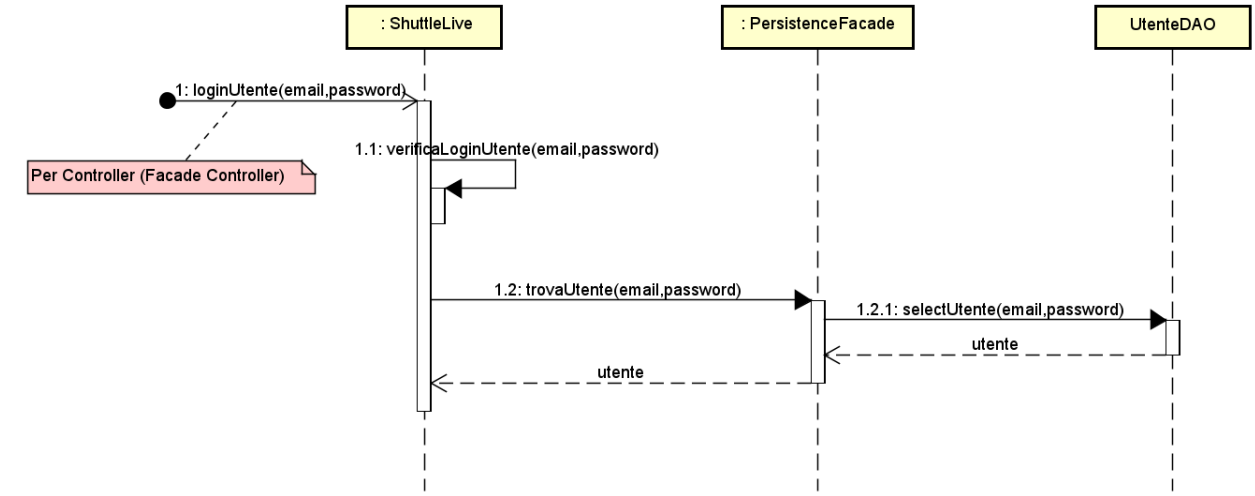
SD_4 InserisciVeicolo



SD_5 loginAutista



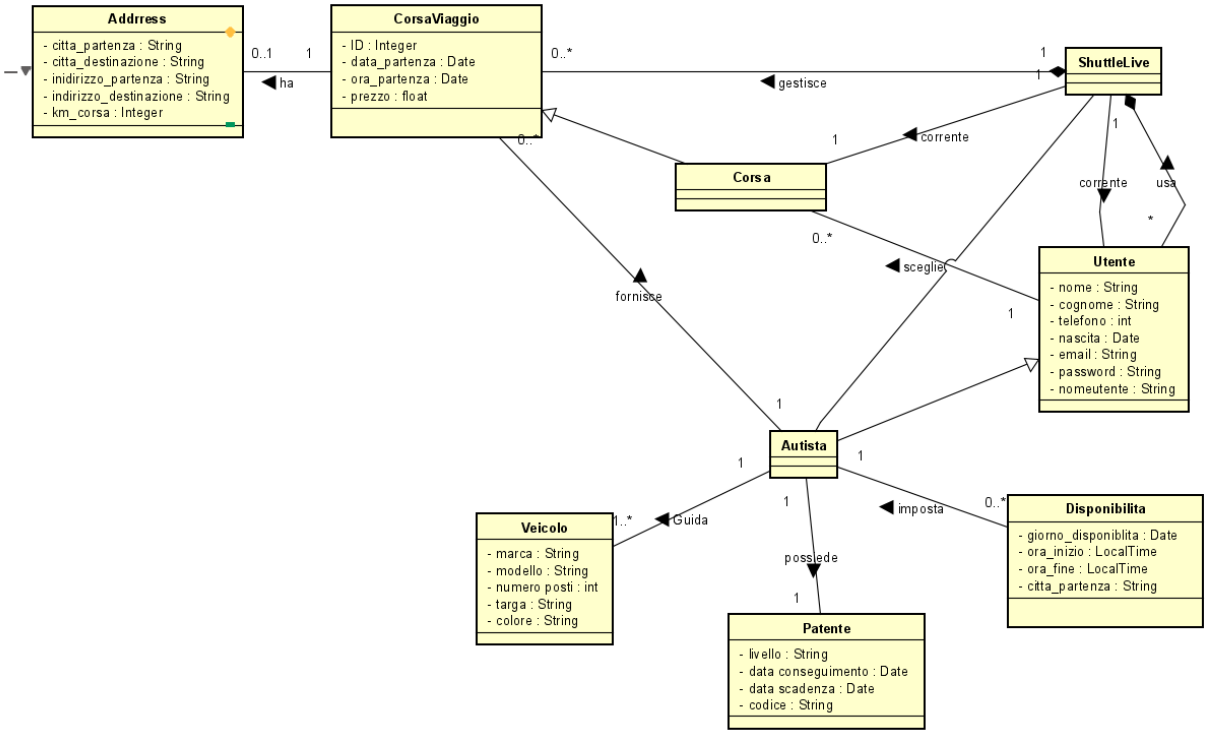
SD_6 loginUtente



MODELLO DI ANALISI

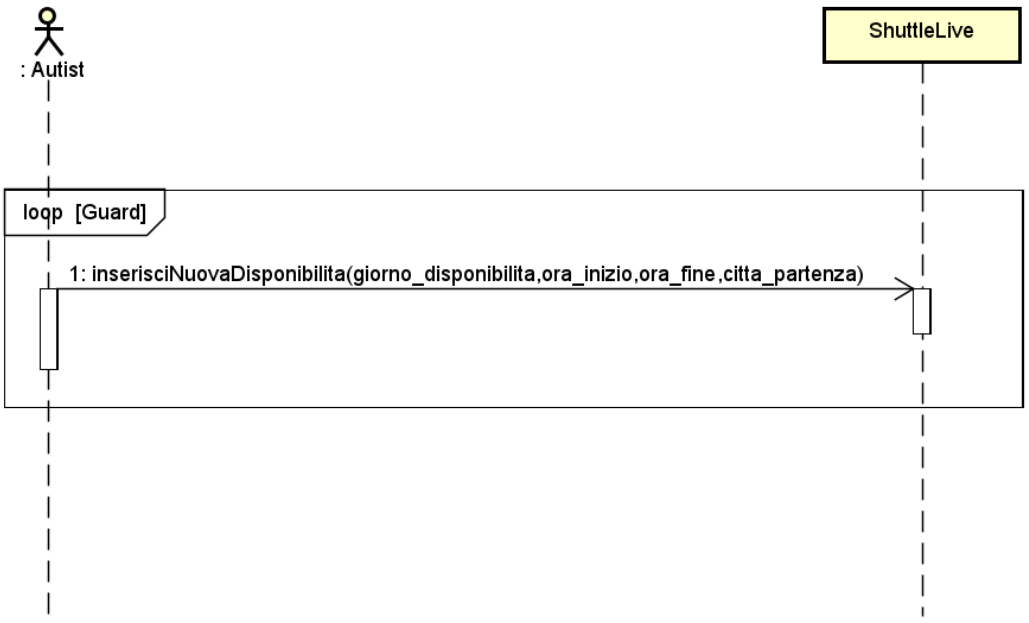
MODELLO DI DOMINIO:

Di seguito è riportato il modello di dominio relativo ai casi d'uso precedentemente citati.



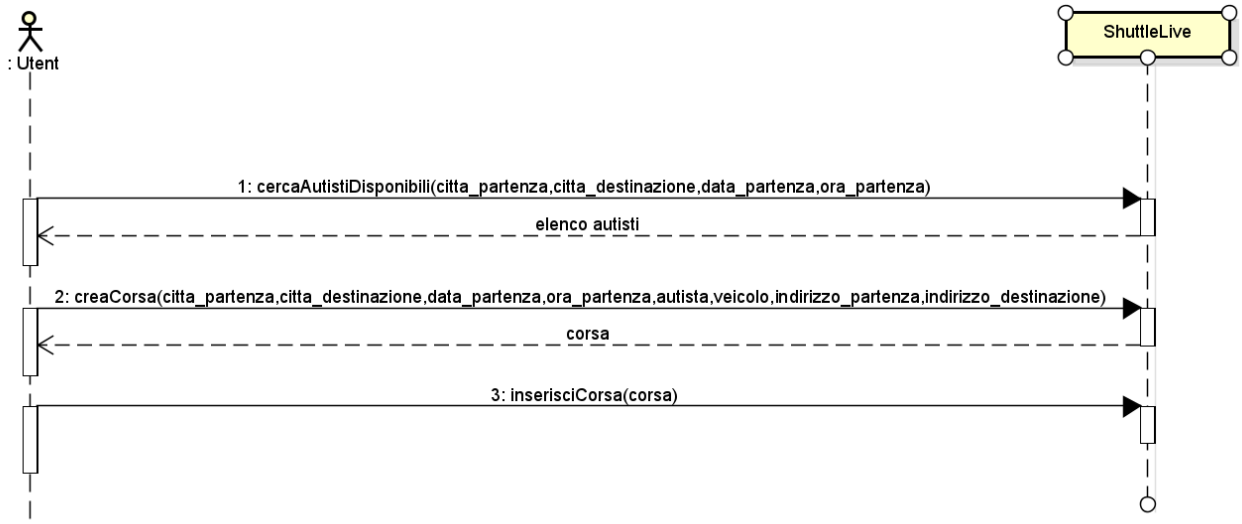
In seguito, vengono riportati i vari SSD dei casi d'uso UC2, UC10 implementati in questa iterazione

SSD UC10:



Viene effettuata la funzione inserisciNuovaDisponibilita che consiste nel creare una nuova istanza disponibilita associarla all'autista corrente e inserirla all'interno del Database.

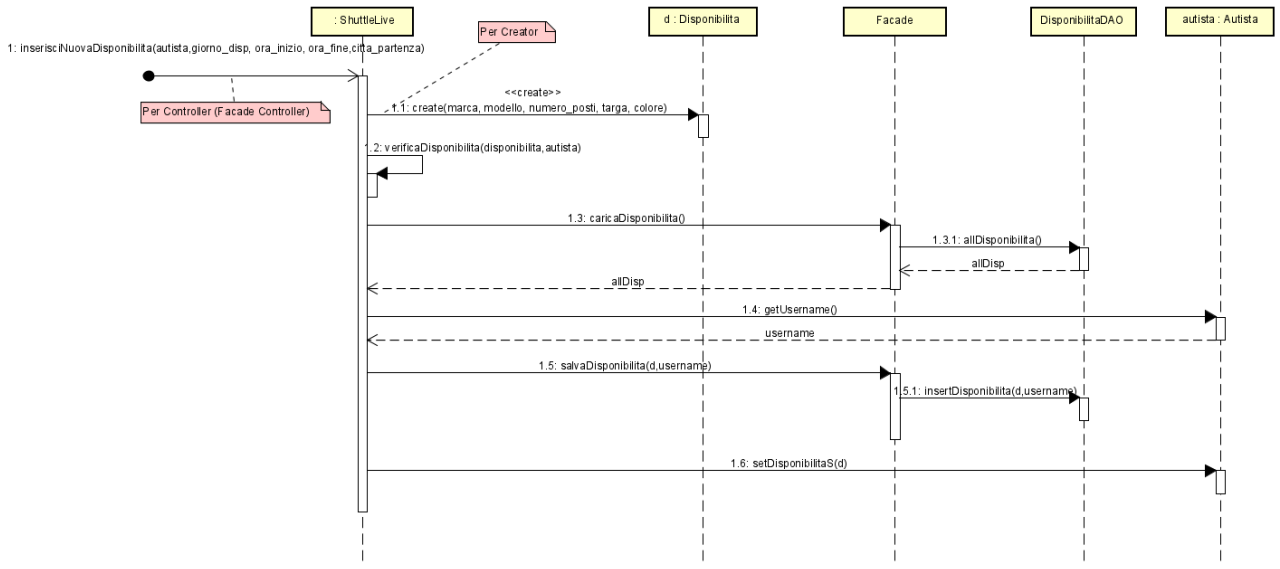
SSD UC2:



Innanzitutto, viene effettuata la funzione cercaAutistiDisponibili con la quale si vanno a cercare tutti gli autisti che si sono resi disponibili per il giorno, l'ora e la città indicata. Successivamente, dopo aver scelto l'autista e il veicolo viene creata mediante ,la funzione creaCorsa, l'istanza corsa con i vari parametri scelti in precedenza. Infine, la inserisciCorsa setta l'utente alla corsa corrente e inserisce quest'ultima all'interno del DataBase mediante l'utilizzo del pattern DAO.

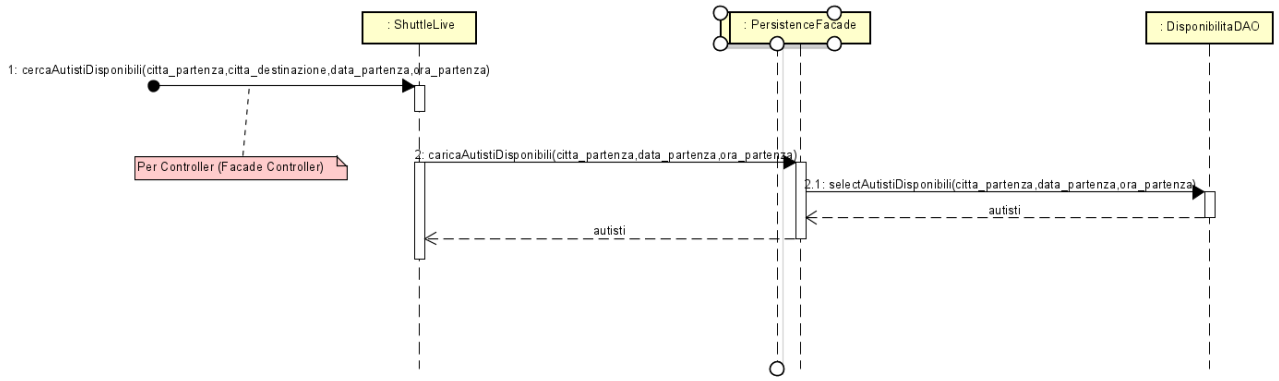
MODELLO DI PROGETTO

SD_1 InserisciNuovaDisponibilita



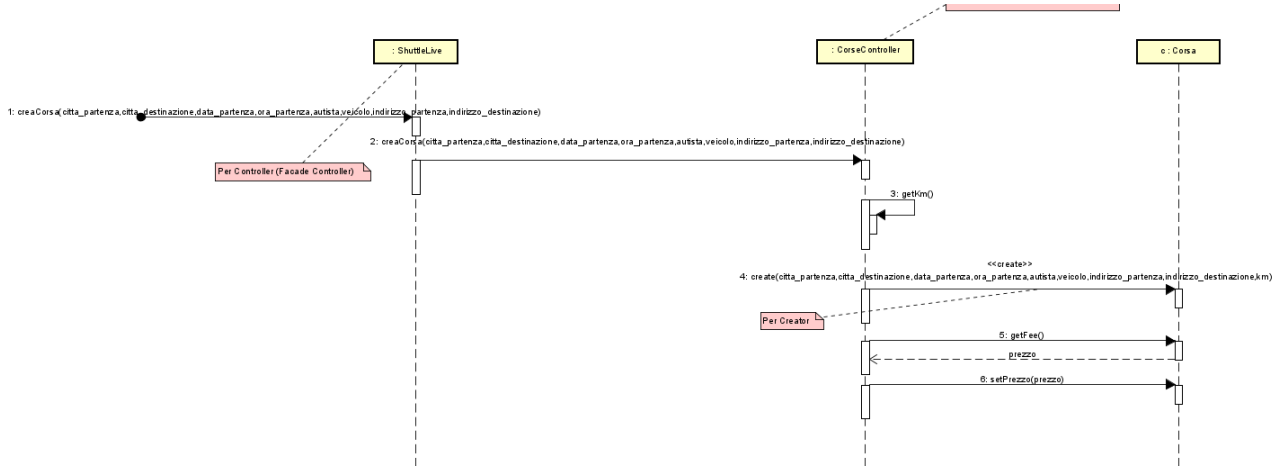
Avviene la creazione di un'istanza di Disponibilita. Vengono verificati i campi inseriti dall'autista con la funzione verifica campi. All'interno di verifica campi viene fatto un controllo che mi evita di creare una nuova istanza se già per quel giorno è già presente una disponibilita. Questo controllo viene effettuato medinate accesso al database interfacciandosi con la classe PersistenceFacade classe introdotta per fare da controller alle varie classi DAO e mantenere High Coesion tra tutte le funzioni per lo scambio di dati con il database. Inoltre questa classe implementa i pattern GOF Singleton e Facade. Viene settata la disponibiulita all'autista corrente. Dopodiché viene ricavato l'username dell'autista e viene inserita la disponibilita all'interno del Database.

SD_2 cercaAutistiDisponibili



Qui viene effettuata la ricerca di autisti disponibili per l'ora, la data e il luogo indicato. Il sistema si interfaccia a PersistenceFacade per ottenere la lista di autisti.

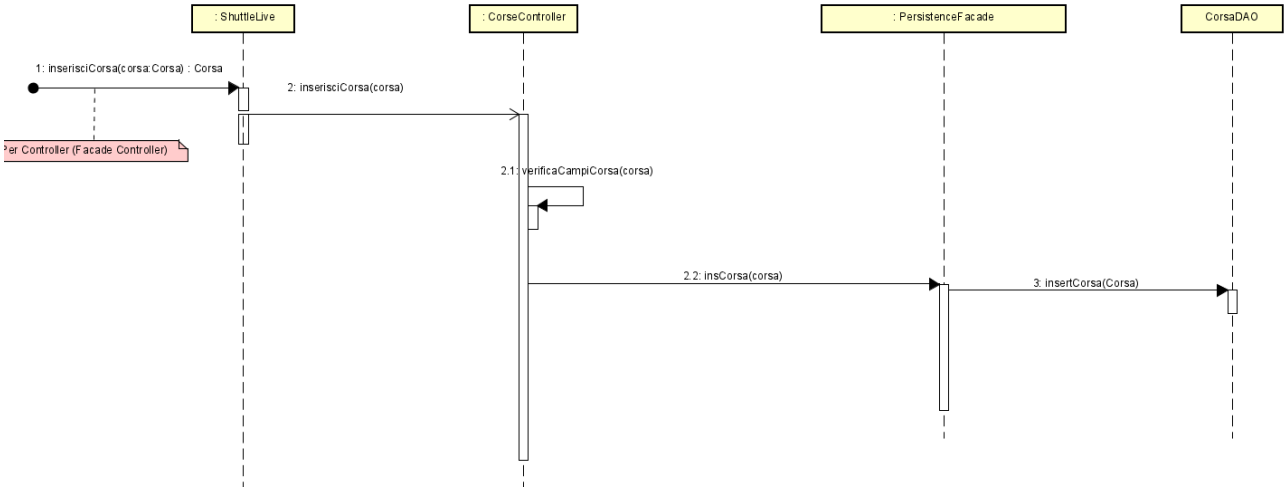
SD_3 creaCorsa



Durante questa operazione viene creata un'istanza corsa di Corsa utilizzando i parametri inseriti precedentemente tra cui autista e veicolo. Tale compito è svolto dal corseController che offre supporto a ShuttleLive nell'eseguire varie funzioni. Della corsa si calcolano i km, il prezzo e li si settano a essa.

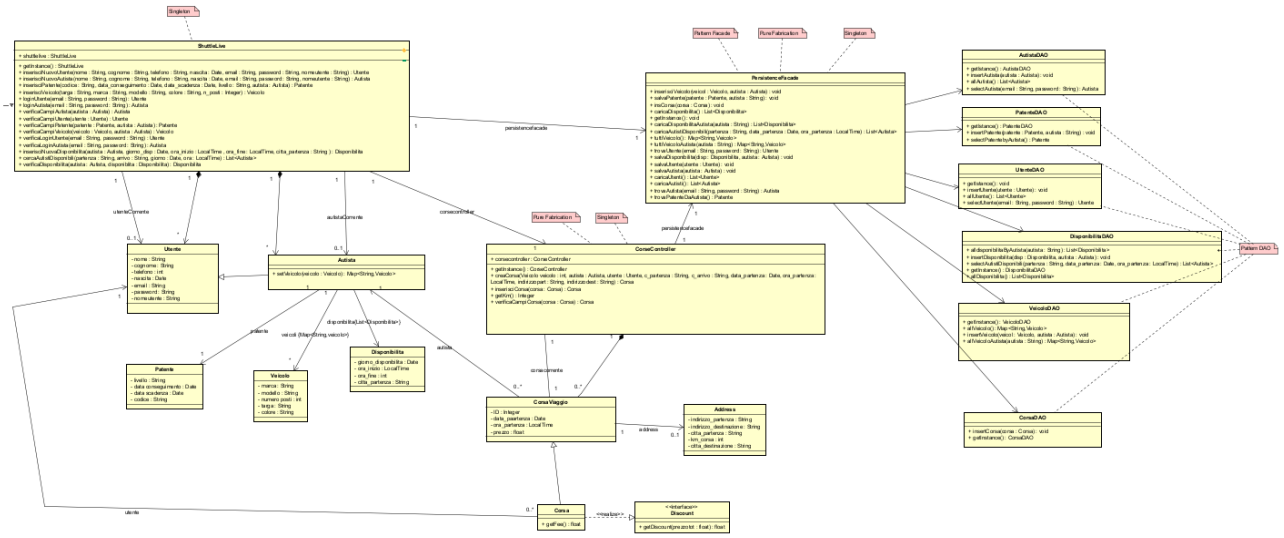
CorseController è una classe introdotta per Pure Fabrication: la sua introduzione viene fatta per nascondere la complessità che vi è nelle operazioni legate alla classe Corsa. In questo modo evito che ShuttleLive debba gestire tale complessità rispettando, quindi, il principio di High Coesion. Inoltre, utilizza i pattern GOF Singleton.

SD_4 inserisciCorsa



Viene effettuato la verifica e successivamente l'inserimento della corsa all'interno del database incaricando sempre la classe PersistenceFacade.

DIAGRAMMA DELLE CLASSI



TEST

In **testInserisciNuovaDisponibilita** è stato testato l'inserimento di una nuova disponibilità. Vengono effettuati vari controlli, tra cui il mancato inserimento di alcuni parametri, una data inserita non valida e antecedente a quella odierna.

In **testCercaAutistiDisponibili** è stato effettuata la ricerca di autisti in base ai parametri in ingresso. Essa fornisce una lista di autisti.

In **testcreaCorsa** è stata testata la creazione di una corsa

In **testInserisciCorsa** viene testato l'inserimento della corsa nel database e vengono controllati i vari campi della corsa.

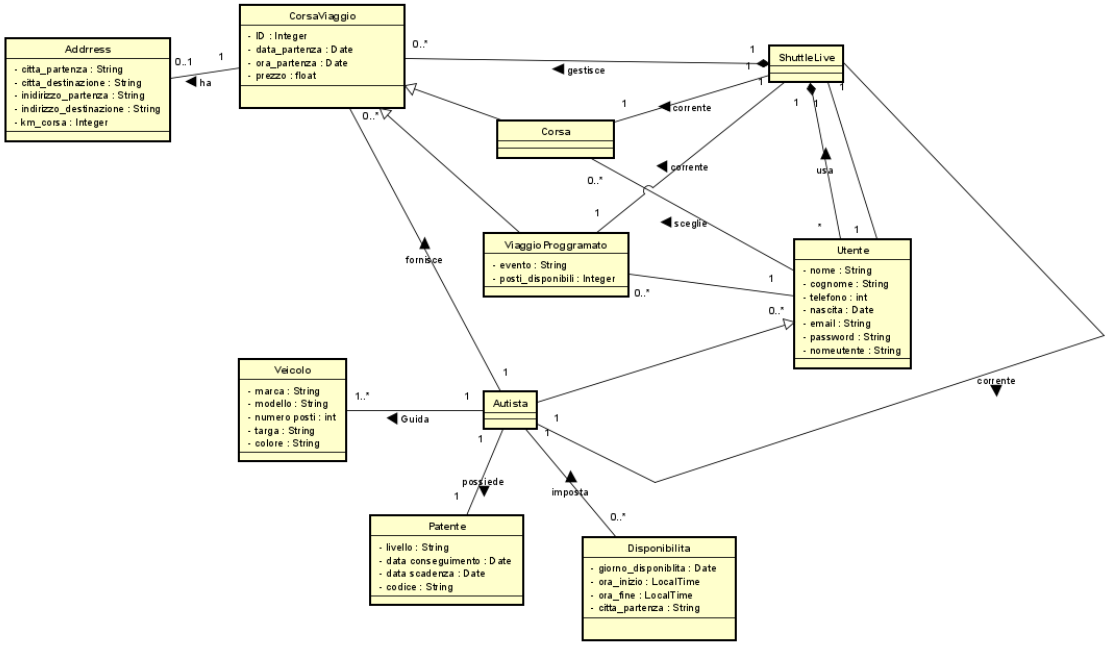
In **testGetDiscount** viene calcolato lo sconto relativo alla corsa e viene settato il prezzo di quest'ultima.

Iterazione 3

Nella seconda iterazione si è effettuata l'analisi, la progettazione, l'implementazione e il testing dei casi d'uso UC3: Inserisci nuovo viaggio programmato, UC4: Elabora prenotazione viaggio programmato.

MODELLO DI ANALISI

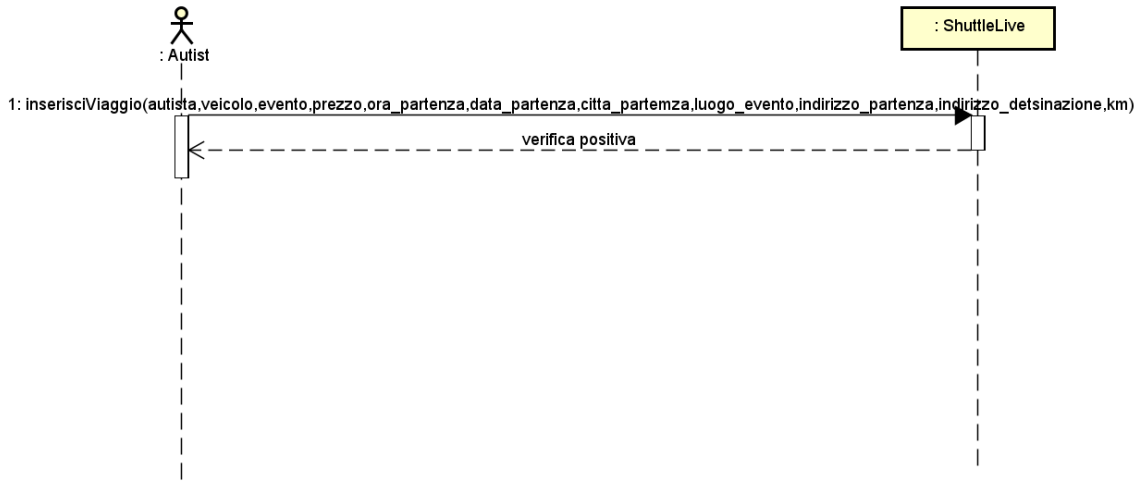
MODELLO DI DOMINIO:



Di seguito è riportato il modello di dominio relativo ai casi d'uso precedentemente citati.

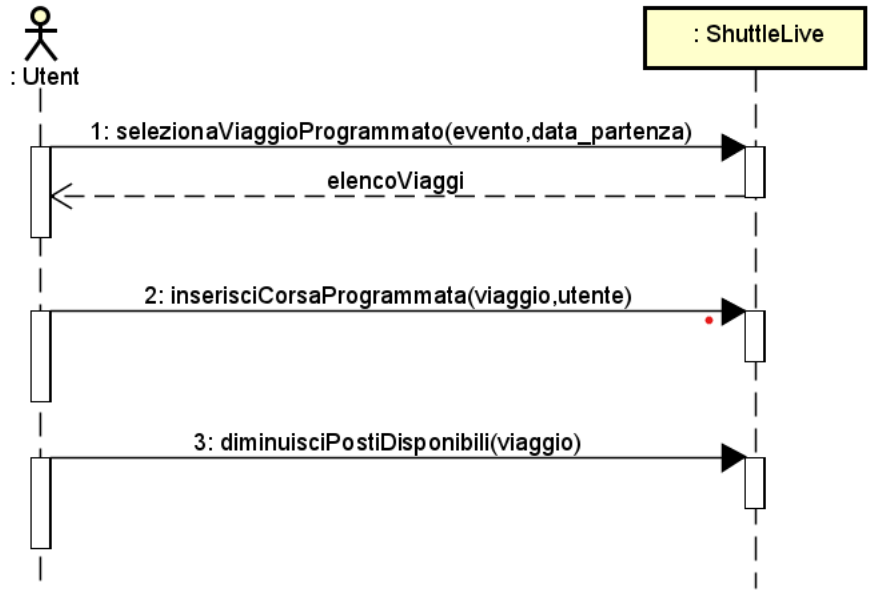
Successivamente vengono descritti gli SSD relativi ai due casi d'uso.

SSD UC3



In questa operazione viene effettuata la inserisciViaggio che consiste nel creare una nuova istanza viaggio associarla all'autista corrente e inserirla all'interno del Database.

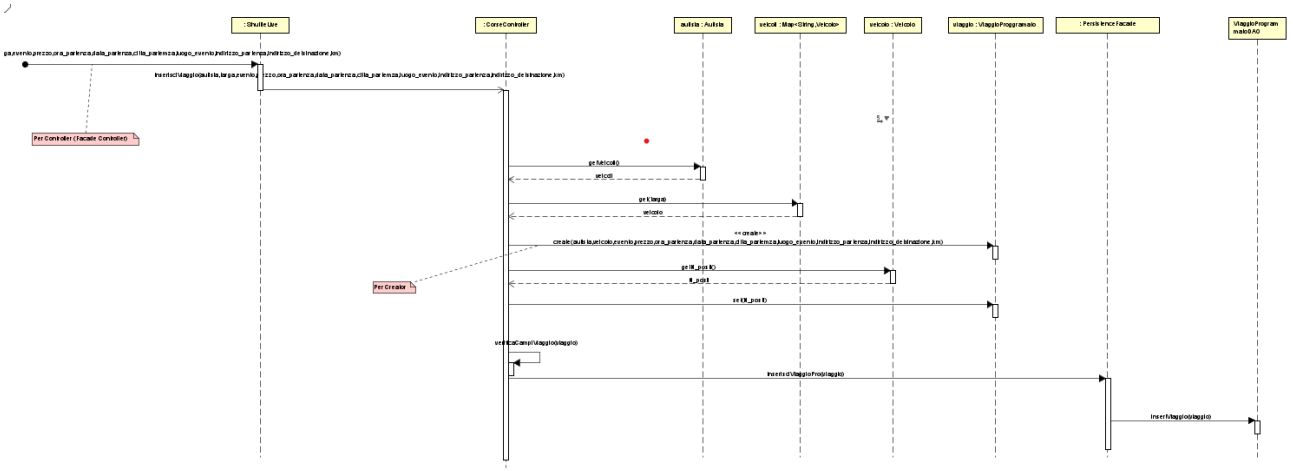
SSD UC4



Innanzitutto, viene effettuata la funzione selezionaViaggioProgrammato con la quale si vanno a cercare tutti i viaggi scremandoli per giorno ed evento inserito. Successivamente, dopo aver scelto il viaggio, l'utente viene aggiunto all'interno del viaggio e mediante la funzione inserisciCorsaProgrammata l'utente associato al viaggio (sorta di biglietto) viene inserito all'interno del database. Infine, la diminuisciPostiDisponibili diminuisce i posti disponibili del viaggio programmato e aggiorna tale valore anceh nel database.

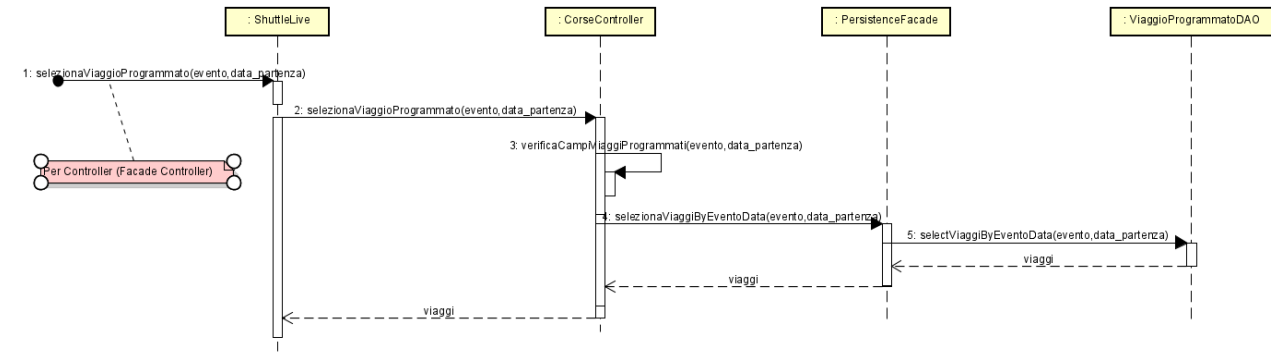
MODELLO DI PROGETTO

SD_1 InserisciViaggio



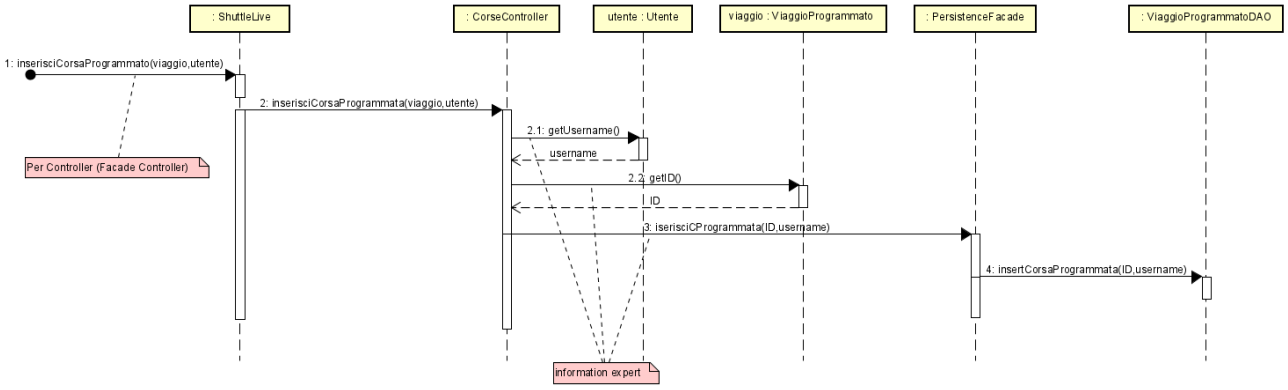
In questo diagramma di sequenza si va ad effettuare la creazione di un'istanza viaggio di ViaggioProgrammato. Per fare ciò facciamo uso del CorseController e del PersistenceFacade per interfacciarci con il database. Dopo aver passato i vari parametri alla funzione inserisciViaggio viene estrapolato dall'autista corrente il veicolo scelto indicando la targa, vengono ricavati i posti disponibili del viaggio dal numero di posti del veicolo selezionato. Dopodiché avviene la creazione del viaggio. I campi di quest'ultimo vengono controllati dalla funzione verificaCampiViaggio e inseriti all'interno del database.

SD_2 selezionaViaggioProgrammato



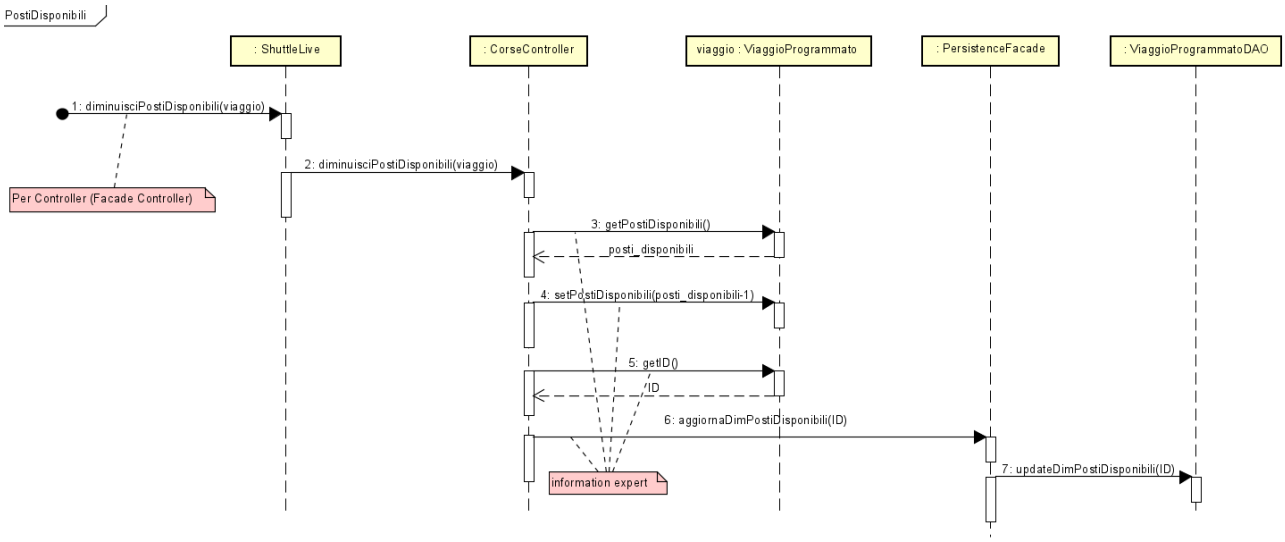
Qui viene effettuata la ricerca di viaggi programmati per l'evento e la data indicata. Il sistema si interfaccia a PersistenceFacade per ottenere la lista di viaggi programmati.

SD_3 inserisciCorsaProgrammata



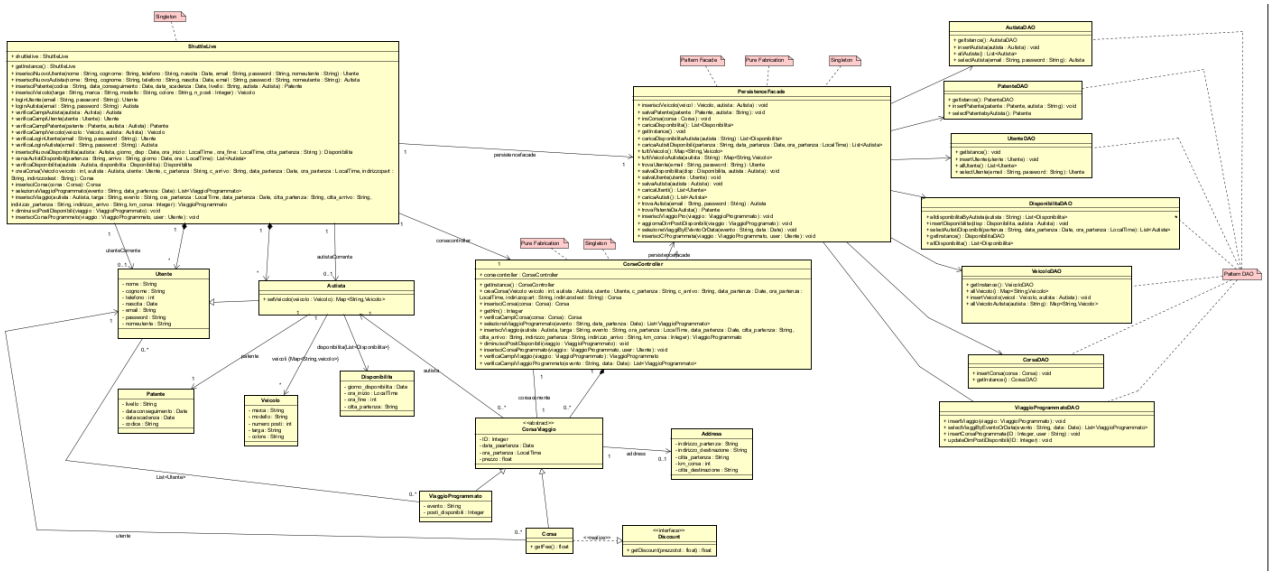
In questa operazione viene inserito l'utente all'interno del viaggio e viene inserito nel database la relazione tra l'utente e il viaggio programmato appena prenotato (una sorta di biglietto). Tale inserimento viene effettuato tramite l'ID del viaggio e l'username dell'utente.

SD_4 diminuisciPostiDisponibili



In questo diagramma avviene la riduzione dei posti disponibili. Innanzitutto viene ridotto di 1 il numero di posti disponibili, in seguito viene aggiornato il database con il nuovo valore di posti disponibili.

DIAGRAMMA DELLE CLASSI



TEST

In **testInserisciViaggio** viene creato un viaggio e se ne testa l'inserimento all'interno del database. Si fanno vari test in cui si valutano vari tipi di viaggio.

In **testSelezionaViaggioProgrammato** si testa la ricerca di viaggi tramite evento e data e torna una lista di viaggi programmati. Anche qui vengono eseguiti vari test in cui si prova a inserire una data non valida o un campo vuoto.

In **testInserisciCorsaprogrammata** si verifica il funzionamento dell'inserimento nel database della coppia viaggio utente.

In **testDiminuisciPostiDisponibili** si testa la riduzione dei posti disponibili in un determinato viaggio e il successivo aggiornamento del database.

Iterazione 4

Nella prima iterazione si è scelto di effettuare l'analisi, la progettazione, l'implementazione e il testing dei casi d'uso UC5: Gestisci Annullamento viaggio/corsa Autista e UC6: Visualizza storico corse/viaggi

MODELLO DI ANALISI

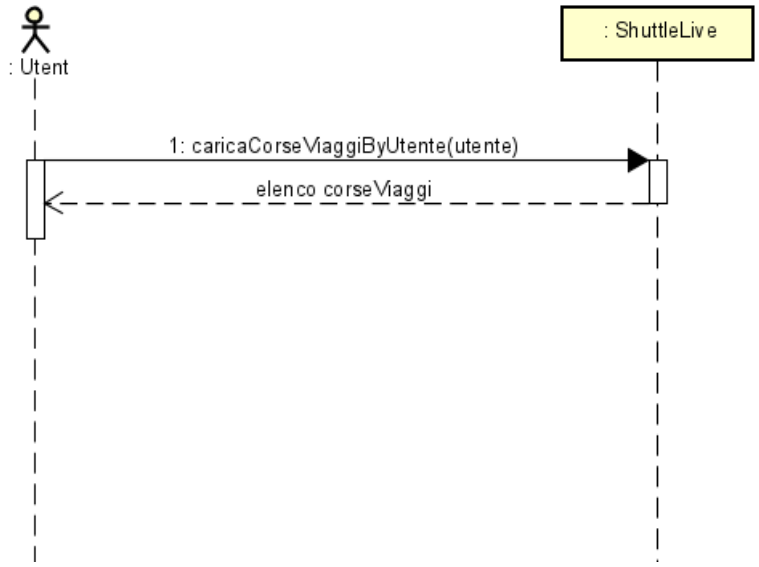
MODELLO DI DOMINIO

Il modello di dominio rimane identico a quello presentato nell'iterazione precedente.

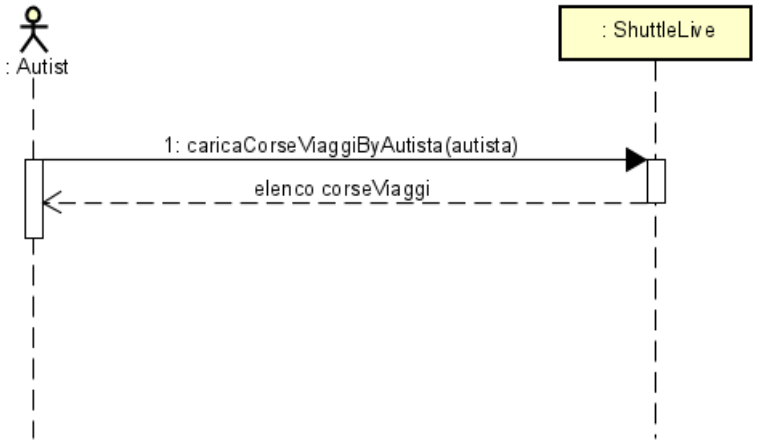
SSD

Di seguito vengono illustrati gli ssd dei casi d'uso elaborati in questa iterazione

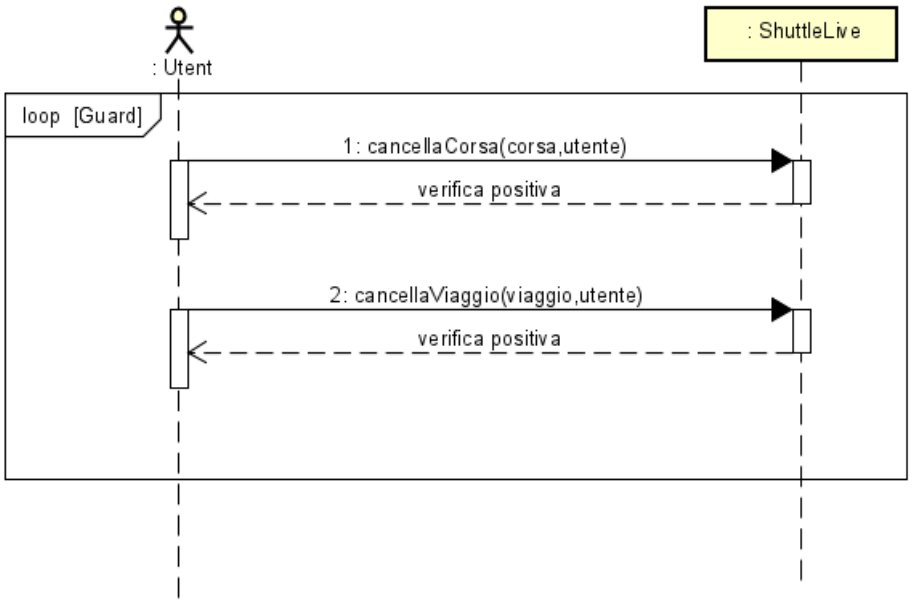
SSD UC5_1



La prima operazione che viene effettuata è quella di **caricaCorseViaggiByUtente()**, in cui si passa l'utente che vuole accedere al proprio storico viaggi/corse



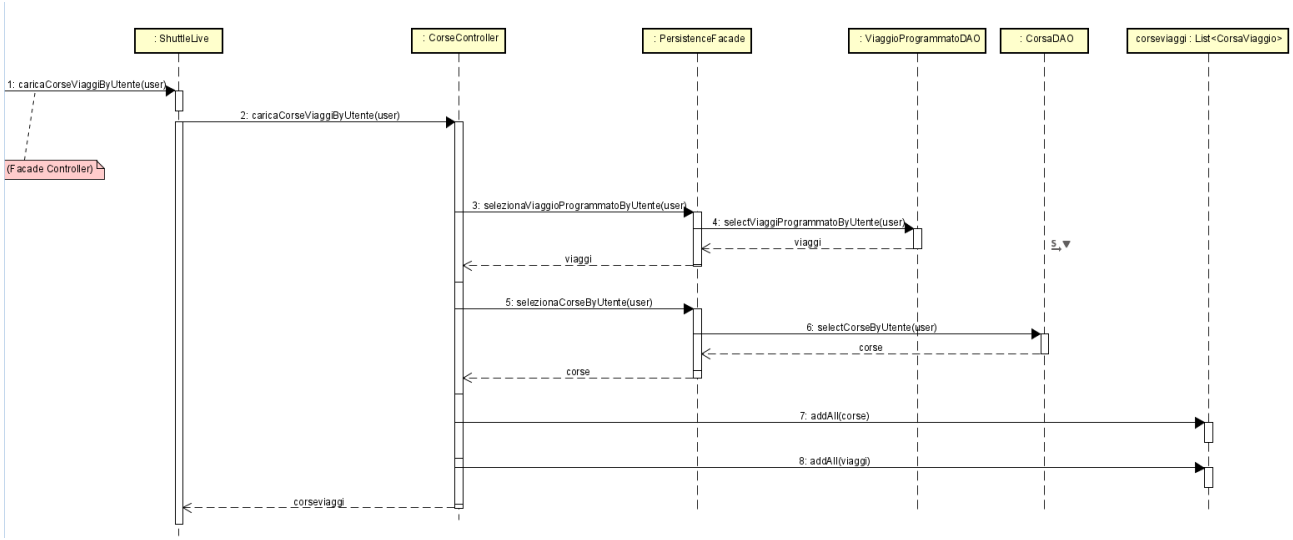
Analogo al precedente dal punto di vista autista



MODELLO DI PROGETTO

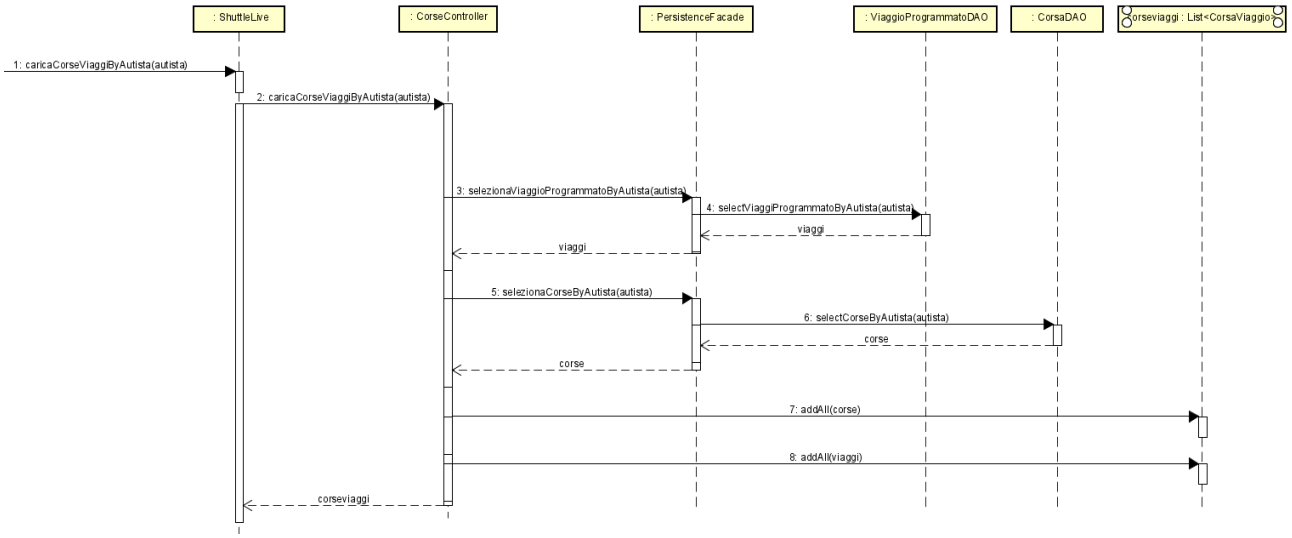
Di seguito vengono illustrati gli sd dei casi d'uso elaborati in questa iterazione

SD_1



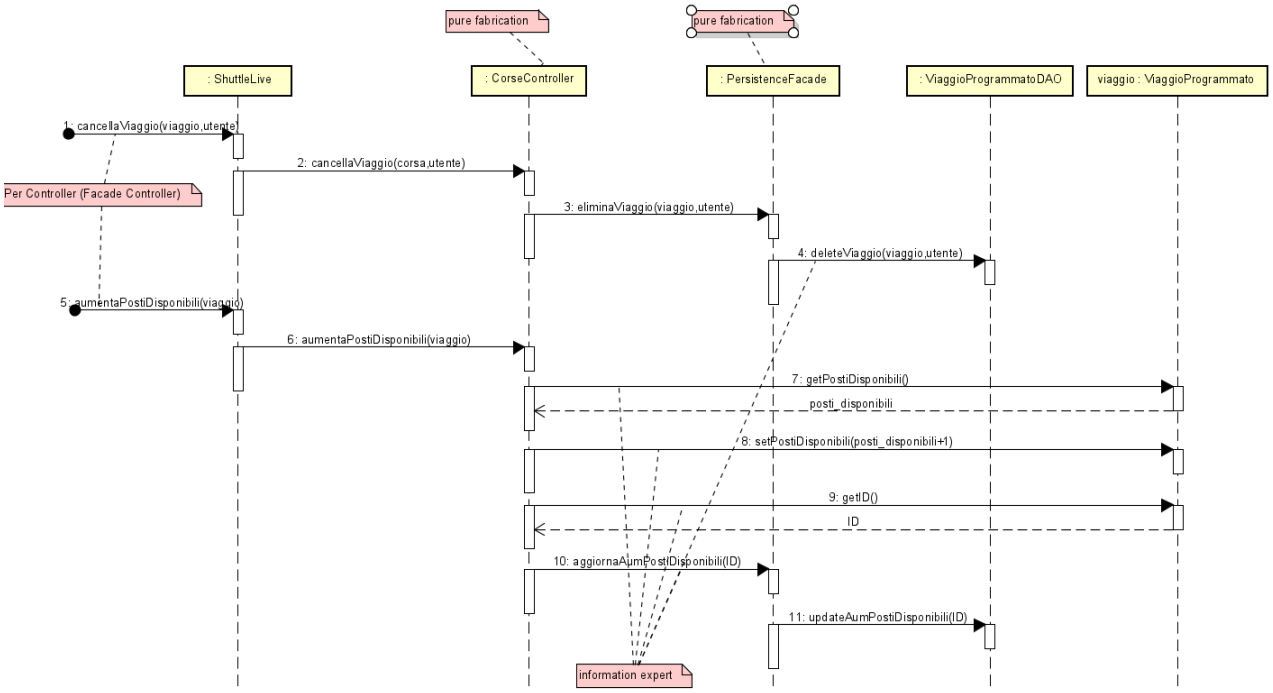
Tramite l'operazione CaricaCorseViaggiByUtente il sistema si interfaccia prima a CorseController che è incaricato di gestire le funzioni relative alle corse dopodiché a PersistenceFacade per accedere ai relativi DAO questo prende una lista di viaggi da ViaggioProgrammatoDAO e fa lo stesso con le corse aggiunge entrambe le liste in una lista di tipo corsaviaggio e la restituisce all'utente

SD_2



Analogo al precedente dal punto di vista autista

SD_3



Tramite la funzione cancellaviaggio passando dal CorseController e da PersistenceFacade

Andiamo a eliminare la prenotazione di un viaggio da parte dell'utente ma non il viaggio stesso che è composto da più utenti viene quindi fatta prima la cancellazione della prenotazione da parte dell'utente dopodiché tramite la getPostiDisponibili ci ricaviamo i posti disponibili in modo da poterli aumentare tramite la set infine una volta ricavato l'id del viaggio andiamo ad aggiornare i posti nel database

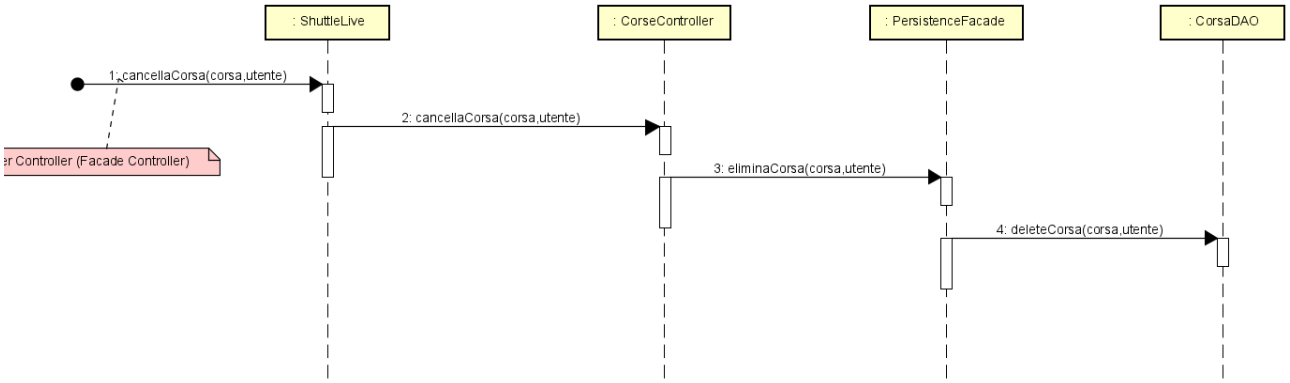
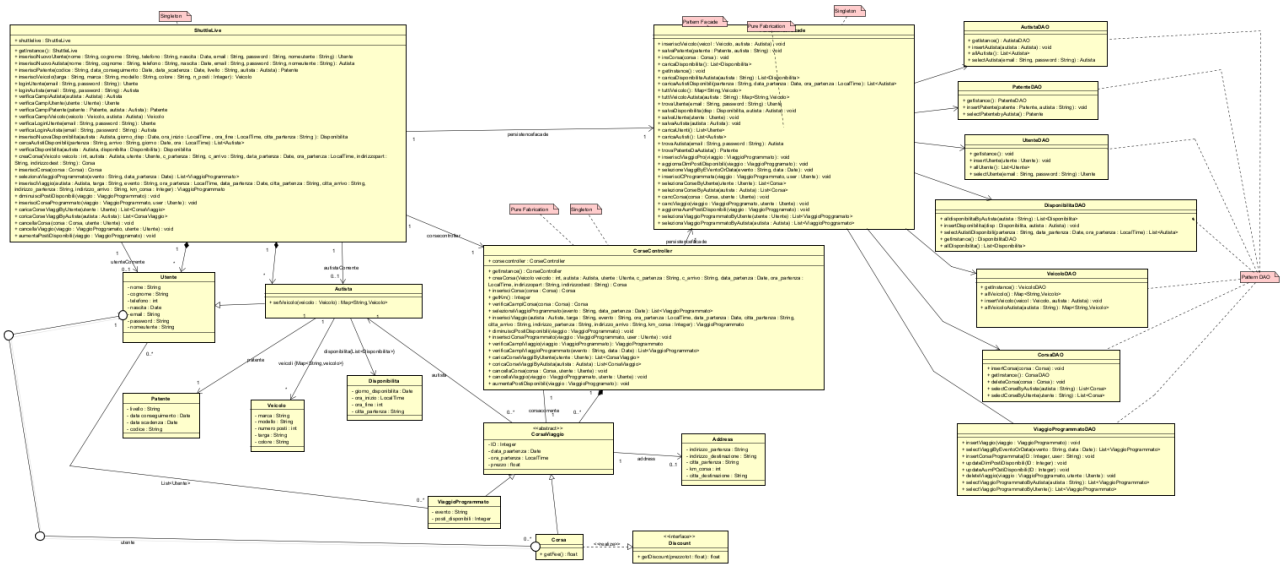


Diagramma delle classi



TEST

Tramite le funzioni **testCaricaCorseViaggiByUtente** e **testCaricaCorseViaggiByAutista** andiamo a testare il corretto caricamento dello storico viaggio dato l'autista/cliente.

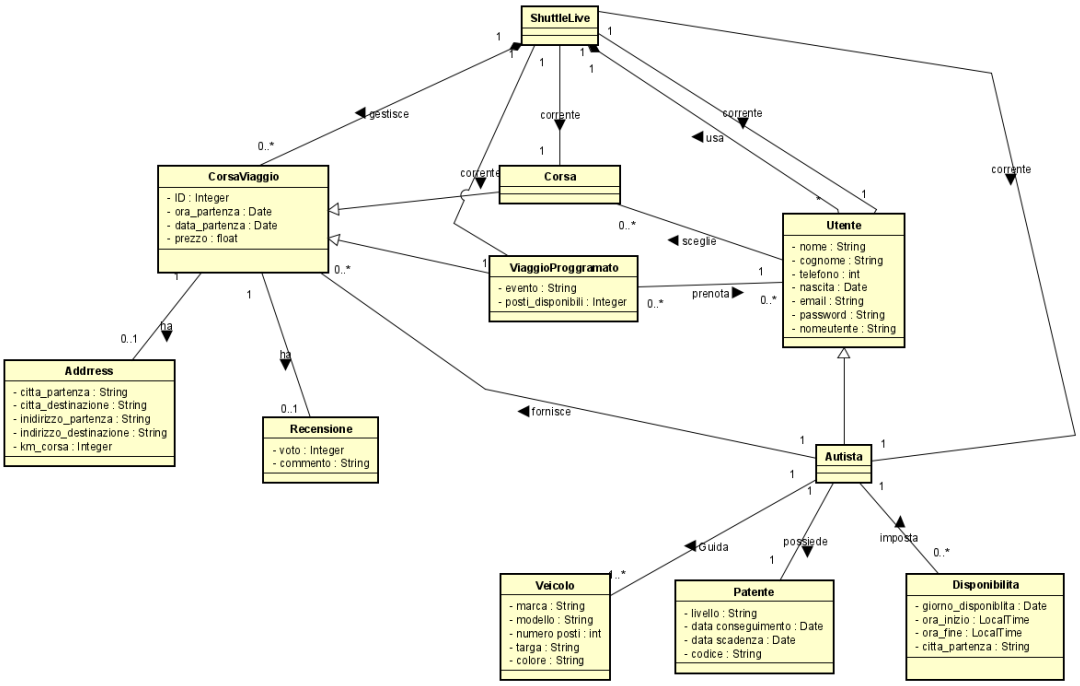
Tramite le funzioni **testcancCorse** e **testcancViaggio** andiamo a testare il corretto cancellamento della corsa o di un viaggio.

Iterazione 5

Nella prima iterazione si è scelto di effettuare l'analisi, la progettazione, l'implementazione e il testing dei casi d'uso UC7: Inserisci Recensione e UC8: Visualizza recensioni Autista.

MODELLO DI ANALISI

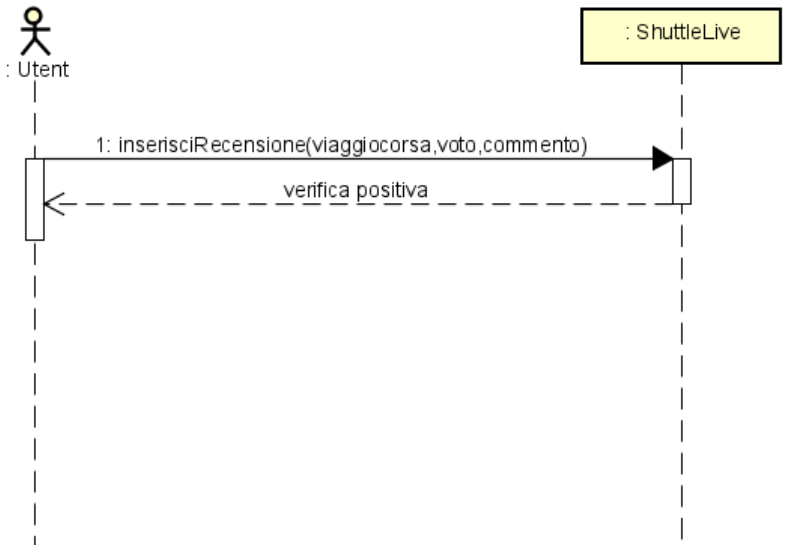
MODELLO DI DOMINIO FINALE



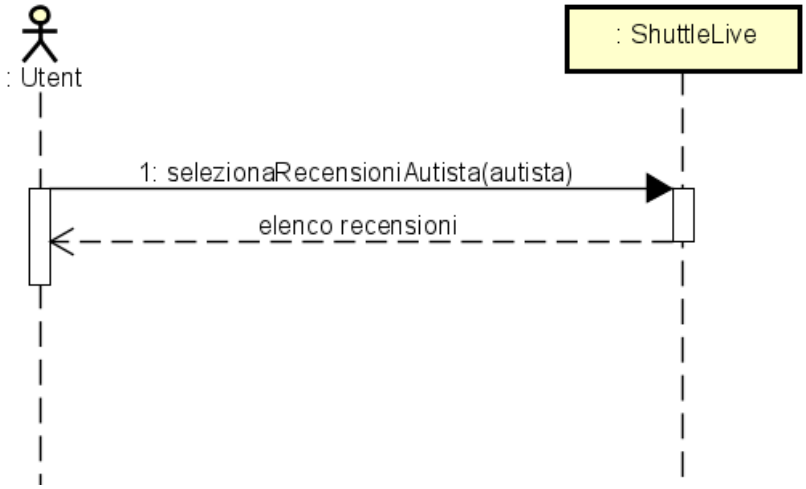
SSD

Di seguito vengono illustrati gli ssd dei casi d'uso elaborati in questa iterazione

SSD UC7



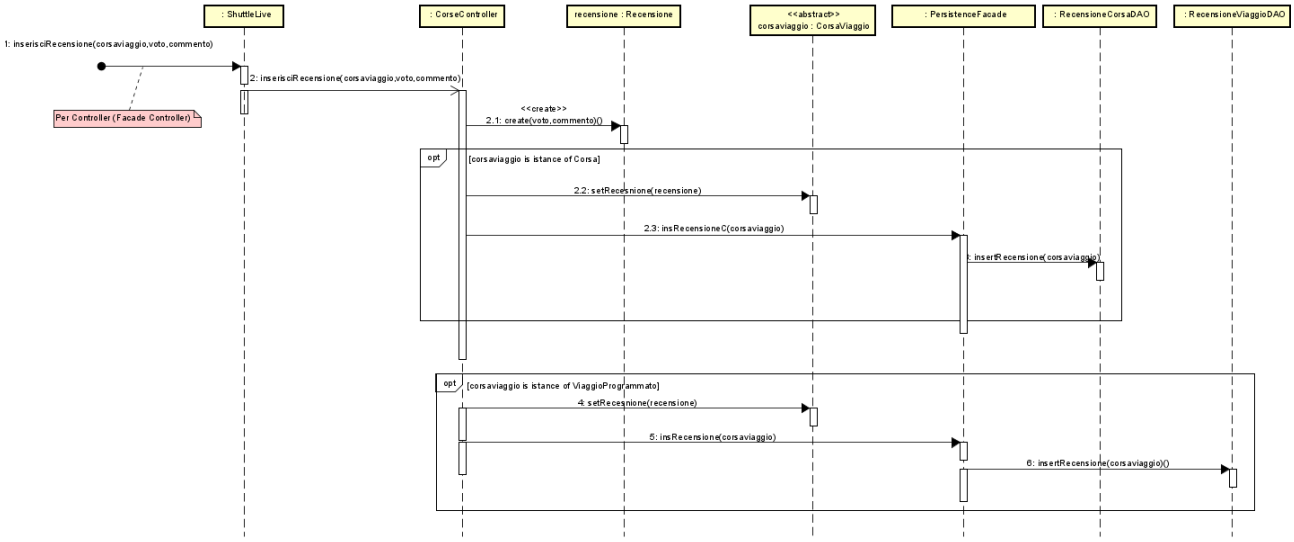
SSD UC8



MODELLO DI PROGETTO

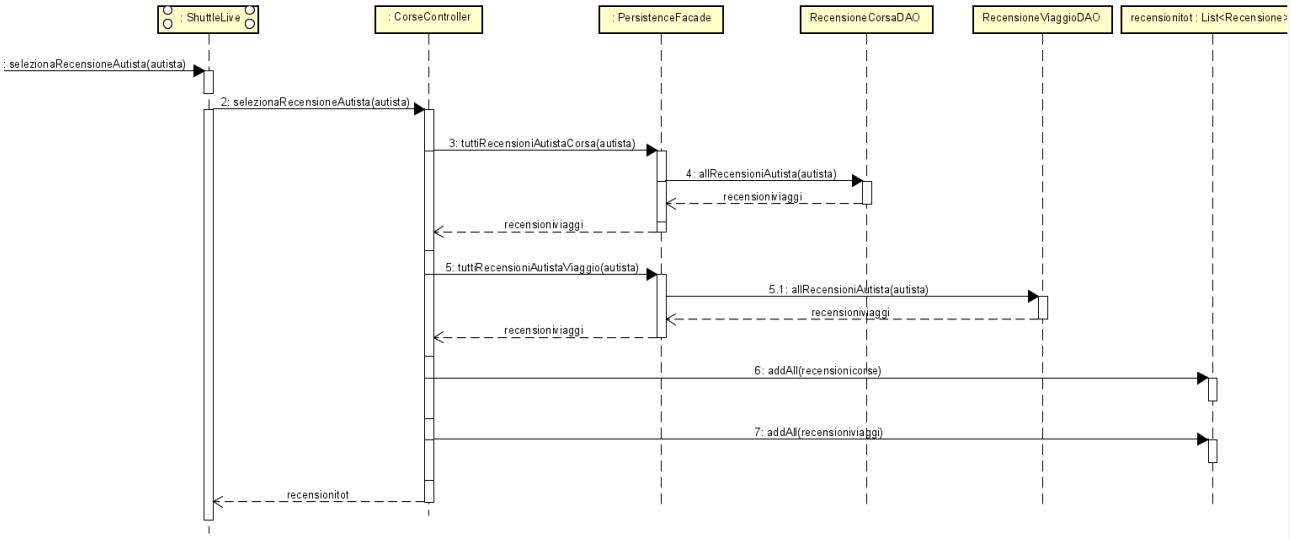
Di seguito sono riportati gli sd dei casi d'uso analizzati

SD_1



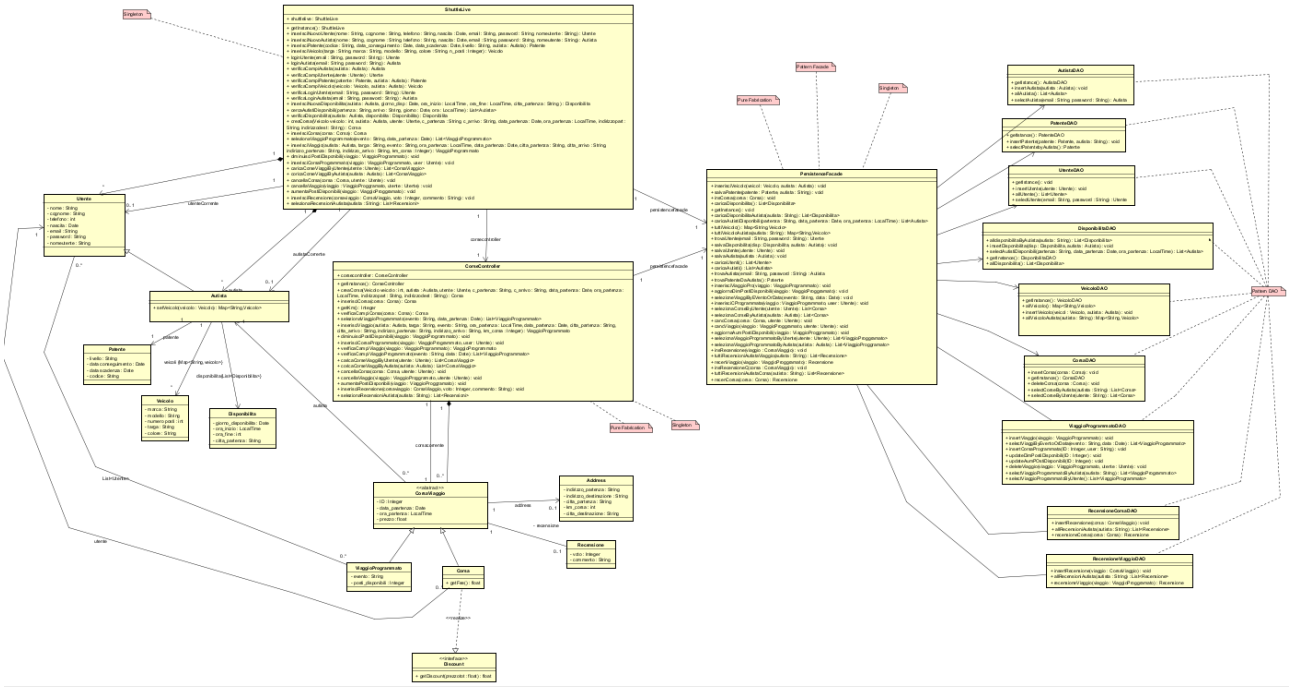
Tramite degli istance of viene inserita la recensione relativa a una corsa o a un viaggio

SD_2



Vengono prese le recensioni relative a una corsa quelle relative ad un viaggio per un determinato autista dai due dao e viene restituita una lista di recensioni totali per quell'autista.

DIAGRAMMA DELLE CLASSI FINALE



TEST

Tramite la funzione **testInserisciRecensione** viene testato che l'inserimento della recensione vada a buon fine, che i campi della recensione non siano vuoti,e che il voto sia compreso tra 1 e 5.

Tramite la funzione **testSelezionaRecensioniAutista** viene testato che questa funzione restituisca le recensioni relative a un determinato autista se questo è inserito correttamente e ha delle relative recensioni.