

# RELAZIONE PROGETTO BIKE SHARING

## 1 Descrizione del problema

### 1.1 Analisi e specifica dei requisiti

Con questo progetto si intende realizzare un applicazione di bikesharing, ovvero di utilizzo di biciclette, prelevabili in città in determinate stazioni, tramite abbonamento.

Il servizio disporrà di 4 applicazioni:

- Applicazione del totem: l'applicazione che verrà eseguita all'interno dei vari totem che controllano le rastrelliere
- Applicazione per gli utenti: dove è possibile monitorare le corse effettuate, e modificare i propri dati
- Applicazione per il personale del servizio: dove il personale del servizio bikesharing può effettuare delle operazioni di manutenzione e di aggiornamento del servizio.

#### **ABBONAMENTO:**

L'utente potrà scegliere tra abbonamento giornaliero, abbonamento settimanale o abbonamento annuale.

Qui le tariffe:

- Abbonamento giornaliero: 4.5€
- Abbonamento settimanale: 9€
- Abbonamento annuale: 36€

Per registrare l'abbonamento, una volta scelta la tipologia, bisognerà inserire un username, una password e una carta di credito. Se l'utente è uno studente potrà specificarlo e potrà accedere allo sconto studenti durante l'utilizzo del bikesharing che prevede l'utilizzo gratuito della bicicletta classica.

Se viene scelto l'abbonamento giornaliero o settimanale esso partirà dal primo prelievo di una bicicletta.

Una volta registrato all'utente verrà dato un codice univoco alfanumerico di 7 caratteri che lo identificherà all'interno del servizio.

#### **RASTRELLIERE:**

Le rastrelliere sono identificate con un nome che indica anche la locazione.

Per ogni rastrelliera ci sono 3 tipi di biciclette : bicicletta classica, bicicletta elettrica e bicicletta elettrica con seggiolino. La dimensione massima per una rastrelliera è di 30 morse mentre la dimensione minima è di 15 morse. Diverse rastrelliere potranno avere un diverso numero di morse elettriche.

Ci sono inoltre 2 tipologie di morse (classica e elettrica) e perciò è consentito mettere nella morsa elettrica solo biciclette elettriche o elettriche con seggiolino, come nella morsa classica è consentito mettere solo biciclette classiche.

#### **PRELIEVO BICICLETTE:**

Per prelevare le biciclette l'utente si recherà a un totem che gestisce una rastrelliera di biciclette. Nell'applicazione del totem l'utente digiterà il proprio codice utente e la propria password. A questo punto potrà scegliere tra 3 diverse tipologie di bici con corrispettive tariffe:

- Bicicletta classica: 0€ per i primi 30 minuti, poi 50 cent ogni mezz'ora fino a 2 ore di utilizzo. Dalle 2 ore in poi di utilizzo il prezzo sarà di 2€ all'ora.  
(se l'utente è uno studente tutta la corsa sarà gratuita).
- Bicicletta elettrica: 0.25€(0-29min), 0.75€(30min-59min),  
1.75€(1h-1.29h), 2.75€(1.30h-1.59h), poi 4 euro per ogni mezz'ora dopo le 2 ore di utilizzo
- Bicicletta elettrica con seggiolino: stessa tariffa della bicicletta elettrica normale.

Una volta scelta la tipologia, viene sganciata dalla prima postazione disponibile la bicicletta di quel tipo, e il numero di postazione viene mostrato sul totem. L'utente quindi potrà incominciare la sua corsa.

La corsa non deve durare più di 2 ore, infatti a chi al momento della restituzione della bicicletta ha superato le 2 ore di corsa viene assegnata una penalità. Se l'utente raggiunge le tre penalità viene sospeso dal servizio.

Se la corsa sta durando da più di 24 ore (oltre a ricevere una penalità alla consegna) viene assegnata una multa di 150€ all'utente.

#### **RESTITUZIONE BICICLETTE:**

Per restituire la bicicletta l'utente non dovrà far altro che agganciare la bicicletta in una morsa libera di una rastrelliera facendo attenzione a agganciare la bicicletta al giusto tipo di morsa (classica o elettrica).

Per verificare che la bicicletta è stata restituita nel modo corretto l'utente potrà digitare il proprio codice utente e password sul totem associato alla rastrelliera che, se la bici è stata agganciata correttamente mostrerà un messaggio di avvenuta restituzione.

Bisogna poi aspettare 5 minuti tra una restituzione e il prossimo prelievo.

Elemento di simulazione: quando il bici digiterà codice utente e password al totem, se ha una bici in uso, gli verrà data la possibilità di agganciare quella bici tramite UI.

#### **APPLICAZIONE UTENTI:**

L'utente potrà accedere anche ad un'applicazione utenti, in cui potrà visionare lo storico delle sue corse, cambiare password, segnalare eventuali danni a una bici, oppure cambiare metodo di pagamento.

#### **AREA SERVIZIO (lavoratori):**

Viene predisposta anche un'applicazione per il personale del bikemi che potrà utilizzarla per creare nuove stazioni, aggiungere o rimuovere biciclette, spostarle da una rastrelliera all'altra per bilanciare il numero tra le diverse rastrelliere e visionare statistiche riguardanti l'utilizzo del bikesharing.

#### **ELEMENTI DI SIMULAZIONE**

- **Gestione carte di credito:** nel momento di inserimento della carta di credito, una volta verificata la sua validità (numero di 16 cifre, cvv di 3 cifre, scadenza) viene generato all'interno di un database del bikesharing un record con le informazioni digitate.

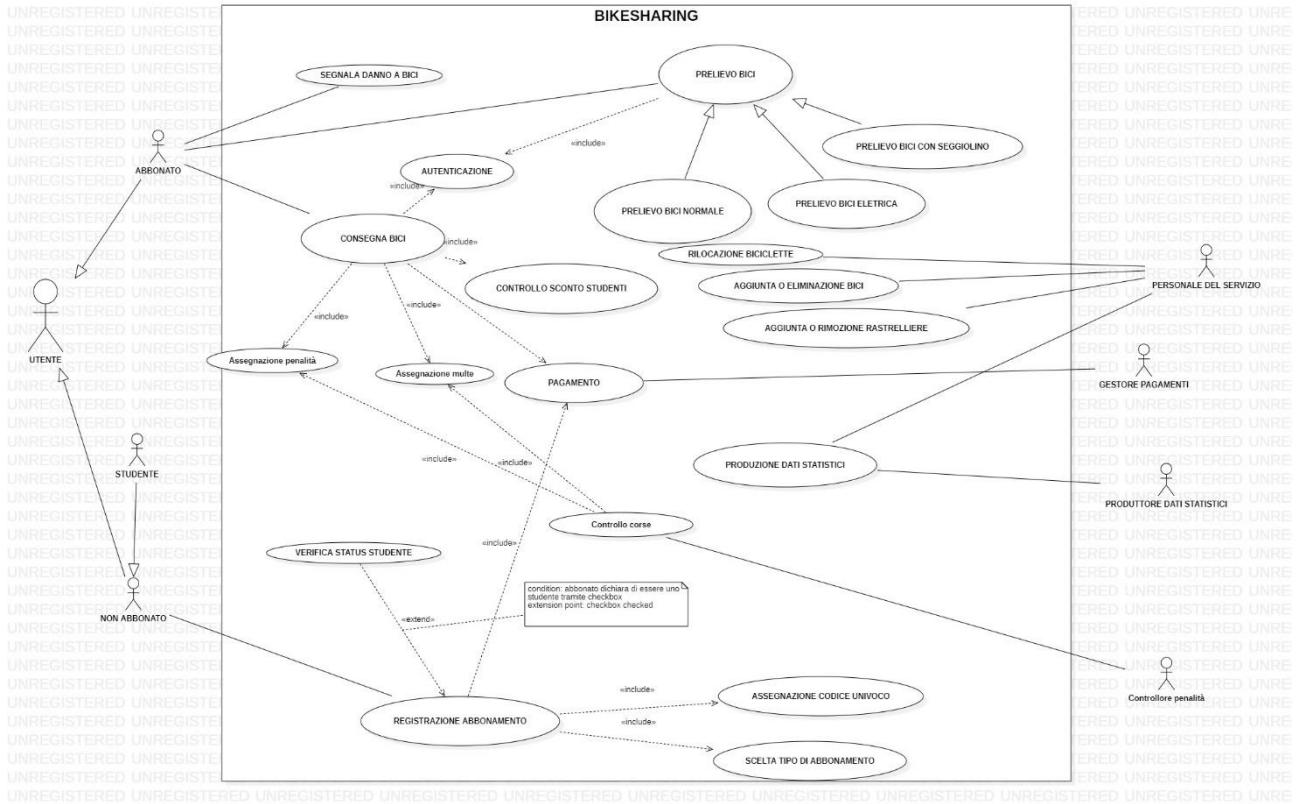
Il database viene usato sia dal bikesharing, per prelevare le informazioni della carta dell'utente, sia dal servizio di pagamento esterno (nella realtà le informazioni delle carte di credito dovrebbero risiedere in un database esterno della banca).

Nel database delle carte sono salvate le informazioni dei soldi spesi dalla carta (che in questo caso saranno spesi soltanto per il bikesharing) e un tetto massimo entro il quale la carta può spendere i soldi. Il tetto viene deciso a priori dall'applicazione.

- Pagamento:** il pagamento andrà quindi ad aumentare i soldi spesi sulla carta salvata nel database e il solo modo in cui non può andare a buon fine sarà se i soldi spesi più l'importo che si sta andando a pagare andrà sopra il tetto stabilito.

## 2 Progettazione del Sistema

### 2.1 Diagramma dei casi d'uso



#### CASI D'USO:

- PRELIEVO BICI:** caso d'uso del prelievo di una bicicletta da una rastrelliera da parte di un ABBONATO, include caso d'uso AUTENTICAZIONE
- PRELIEVO BICI NORMALE:** specializzazione del caso d'uso base PRELIEVO BICI , nel momento in cui viene scelto di prelevare una bici di tipo classico
- PRELIEVO BICI ELETTRICA:** specializzazione del caso d'uso base PRELIEVO BICI , nel momento in cui viene scelto di prelevare una bici elettrica
- PRELIEVO BICI SEGGIOLINO:** specializzazione del caso d'uso base PRELIEVO BICI , nel momento in cui viene scelto di prelevare una bici elettrica con seggiolino
- AUTENTICAZIONE:** caso d'uso che richiede all'ABBONATO di inserire le proprie credenziali nell'app del totem( codice utente e password)
- CONSEGNA BICI:** caso d'uso che si verifica quando un ABBONATO vuole restituire una bici nella rastrelliera, include il caso d'uso PAGAMENTO (pagamento della corsa), il caso d'uso AUTENTICAZIONE (autenticazione al totem) e il caso d'uso CONTROLLO SCONTO STUDENTI (nel caso che sia studente e la bicicletta in utilizzo sia di tipo classico la corsa è gratuita).
- PAGAMENTO:** caso d'uso che si verifica quando è richiesto un pagamento all'ABBONATO. Il pagamento viene gestito dal GESTORE PAGAMENTI

- **CONTROLLO SCONTI STUDENTI:** caso d'uso nel quale si verifica che l'abbonato in questione sia uno studente e la bici in utilizzo sia di tipo classico.
- **REGISTRAZIONE ABBONAMENTO:** caso d'uso che si verifica quando un attore NON ABBONATO vuole abbonarsi al servizio di bikesharing. INCLUDE il caso d'uso PAGAMENTO (pagamento abbonamento), il caso d'uso ASSEGNAZIONE CODICE UNIVOCO e SCELTA TIPO ABBONAMENTO.
- **ASSEGNAZIONE CODICE UNIVOCO:** caso d'uso che si verifica quando un NON ABBONATO si è appena iscritto e gli viene assegnato un codice univoco per utilizzare il servizio
- **SCELTA TIPO ABBONAMENTO:** caso d'uso che si verifica quando un NON ABBONATO deve decidere che tipo di abbonamento scegliere: giornaliero, settimanale o annuale
- **VERIFICA STATUS STUDENTE:** estensione del caso d'uso base REGISTRAZIONE ABBONAMENTO. Si verifica quando un NON ABBONATO dichiara di essere uno studente nel form di registrazione e si procede quindi alla verifica del suo status di studente (deve inserire università e numero di matricola, che verranno poi verificati dal sistema).
- **PRODUZIONE DATI STATISTICI:** caso d'uso osservabile dal personale del servizio bikesharing che visualizzano dati riguardanti il servizio generati da un PRODUTTORE DATI STATISTICI
- **CONTROLLO CORSE:** questo caso d'uso avviene periodicamente all'interno del servizio e serve per controllare che le biciclette non siano utilizzate oltre i limiti.
- **ASSEGNAZIONE PENALITA':** viene assegnata una penalità se un ABBONATO ha utilizzato una bicicletta da più di 2 ore
- **ASSEGNAZIONE MULTA:** viene assegnata una multa di 150 € se un ABBONATO sta utilizzando la bicicletta da più di 24 ore
- **RILOCAZIONE BICICLETTE:** caso d'uso che si verifica quando un attore del PERSONALE decide di rilocare una bicicletta da una rastrelliera ad un'altra
- **AGGIUNTA O ELIMINAZIONE BICI:** caso d'uso che si verifica quando un PERSONALE decide di aggiungere o eliminare una bici da una rastrelliera
- **AGGIUNTA O ELIMINAZIONE RASTRELLIERE:** caso d'uso che si verifica quando un PERSONALE decide di costruire una nuova rastrelliera, decidendo il luogo oppure eliminarne una

## 2.2 Descrizione degli scenari

*Tabella 1- Template per la descrizione di un caso d'uso.*

<b>Nome</b>	<b>PRELIEVO BICI</b>
<b>Scopo</b>	Utente abbonato vuole incominciare una corsa con una bicicletta
<b>Attore/i</b>	ABBONATO

<b>Pre-condizioni</b>	/
<b>Trigger</b>	Un utente abbonato schiaccia START sullo schermo del totem
<b>Descrizione sequenza eventi</b>	<ol style="list-style-type: none"> <li>1. L'utente inserisce il proprio codice utente e password</li> <li>2. L'utente seleziona la tipologia di bicicletta</li> <li>3. La bici della tipologia selezionata viene sganciata dalla rastrelliera</li> <li>4. Viene mostrata sullo schermo del totem la posizione della bicicletta appena sganciata</li> </ol>
<b>Alternativa/e</b>	<p>1a : codice utente o password risultano errati e viene visualizzato un messaggio di errore sul totem</p> <p>1b: l'utente è autenticato, ma ha già una corsa in corso e perciò gli viene chiesto se vuole agganciare la bici su quella rastrelliera</p> <p>1c: l'utente è autenticato, ma ha finito una corsa da meno di 5 minuti e perciò viene mostrato a schermo un messaggio di avvenuta consegna</p> <p>1d: l'utente è autenticato ,ma non è più abilitato a utilizzare il bikesharing e perciò gli viene mostrato un messaggio di errore</p> <p>3a : non ci sono abbastanza biciclette di quel tipo disponibili e perciò viene visualizzato un messaggio di errore sul totem</p>
<b>Post-condizioni</b>	Viene assegnata una bicicletta all'utente abbonato e preso il tempo di inizio corsa

<b>Nome</b>	<b>CONSEGNA BICI</b>
<b>Scopo</b>	Utente abbonato vuole consegnare una bici sulla rastrelliera
<b>Attore/i</b>	ABBONATO,GESTORE PAGAMENTI
<b>Pre-condizioni</b>	L'utente abbonato deve avere una corsa in corso
<b>Trigger</b>	L'utente abbonato aggancia bici alla rastrelliera in una certa posizione
<b>Descrizione sequenza eventi</b>	<ol style="list-style-type: none"> <li>1. Viene aggiunta la bicicletta alla rastrelliera nella posizione scelta dall'utente</li> <li>2. Viene calcolato il prezzo della corsa in base alla bici scelta e al tempo di utilizzo</li> <li>3. Viene controllato se è applicabile lo sconto studenti, nel caso che l'utente sia uno studente e la bicicletta sia classica</li> <li>4. Nel caso che la corsa abbia superato le 2 ore viene assegnata una penalità all'utente</li> <li>5. Nel caso che la corsa abbia superato le 24 ore viene assegnata una multa di 150 € all'utente</li> <li>6. Viene addebitato il costo della corsa (più eventuale multa) sulla carta dell'utente tramite il GESTORE PAGAMENTI</li> <li>7. (opzionale): l'utente può autenticarsi di nuovo sul totem responsabile della rastrelliera dove ha appena agganciato la sua bicicletta per confermare l'avvenuta restituzione</li> </ol>
<b>Alternativa/e</b>	<p>1a: la morsa della rastrelliera non è giusta per la bicicletta usata perciò il sistema segnala un errore</p> <p>6a: il pagamento viene rifiutato e l'abbonamento dell'utente abbonato viene sospeso</p>
<b>Post-condizioni</b>	Viene aggiunta una bicicletta alla rastrelliera e viene terminata e addebitata la corsa dell'utente abbonato

<b>Nome</b>	<b>REGISTRAZIONE ABBONAMENTO</b>
<b>Scopo</b>	Un utente vuole abbonarsi al servizio di bikesharing
<b>Attore/i</b>	NON ABBONATO,GESTORE PAGAMENTI
<b>Pre-condizioni</b>	/
<b>Trigger</b>	Un utente accede all'applicazione di bikesharing e clicca sul pulsante di registrazione
<b>Descrizione sequenza eventi</b>	<ol style="list-style-type: none"> <li>1. L'utente sceglie la tipologia dell'abbonamento(giornaliero, settimanale, annuale)</li> <li>2. L'utente sceglie e inserisce un username e una password</li> <li>3. L'utente può dichiarare o meno di essere uno studente tramite una checkbox</li> <li>4. Se il cliente dichiara di essere uno studente inserisce nome dell'università e numero di matricola per verificare il suo status di studente</li> <li>5. L'utente mette i dati della carta di credito</li> <li>6. Viene addebitato il costo dell'abbonamento sulla carta dell'utente tramite il GESTORE PAGAMENTI</li> <li>7. Viene mostrato su schermo codice univoco assegnato</li> </ol>
<b>Alternativa/e</b>	<p>4a : l'utente non risulta essere uno studente e perciò viene mostrato un messaggio di errore e viene reindirizzato alla pagina di registrazione iniziale</p> <p>5a: La data di scadenza della carta non è valida(meno di un mese nel caso di abbonamento giornaliero e mensile, meno di un anno nel caso di abbonamento annuale). Viene mostrato un messaggio di errore e l'utente viene reindirizzato alla pagina di registrazione iniziale</p>
<b>Post-condizioni</b>	Viene assegnato un codice univoco al nuovo utente con il quale iniziare ad utilizzare il servizio di bikesharing

<b>Nome</b>	<b>RILOCAZIONE BICICLETTE</b>
<b>Scopo</b>	Scegliere delle biciclette posizionate su una rastrelliera e spostarle su un'altra rastrelliera
<b>Attore/i</b>	PERSONALE DEL SERVIZIO
<b>Pre-condizioni</b>	Il personale del servizio deve essere autenticato
<b>Trigger</b>	Il personale va nella sezione di rilocazione nella applicazione per i lavoratori di bikesharing
<b>Descrizione sequenza eventi</b>	<ol style="list-style-type: none"> <li>1. Viene scelta una rastrelliera sorgente</li> <li>2. Viene scelta una rastrelliera destinazione</li> <li>3. Viene scelte una o più biciclette dalla rastrelliera destinazione selezionando per ognuna il tipo (classica, elettrica o seggiolino)</li> <li>4. Vengono posizionate le biciclette scelte nella rastrelliera destinazione</li> </ol>

<b>Alternativa/e</b>	4a: nella rastrelliera destinazione non ci sono abbastanza posti disponibili per tutte le biciclette scelte e viene mostrato un messaggio di errore sulla schermata 4b: non ci sono abbastanza biciclette nella rastrelliera sorgente e perciò viene segnalato un errore dal sistema
<b>Post-condizioni</b>	Vengono rimosse tot bici da una rastrelliera e vengono aggiunte tot bici su un'altra rastrelliera

<b>Nome</b>	<b>CREAZIONE RASTRELLIERA</b>
<b>Scopo</b>	Creare una nuova rastrelliera
<b>Attore/i</b>	PERSONALE DEL SERVIZIO
<b>Pre-condizioni</b>	Il personale del servizio deve essere autenticato
<b>Trigger</b>	L'utente del personale entra nell'opzione del menu "aggiungi/rimuovi rastrelliere"
<b>Descrizione sequenza eventi</b>	<ol style="list-style-type: none"> <li>Viene scelto un nome della rastrelliera (in base alla posizione in cui verrà collocata)</li> <li>Viene scelta una dimensione della rastrelliera (da 15 a 30 morse).</li> <li>Viene scelto il numero di morse elettriche su quella rastrelliera (che non potrà superare il numero di morse scelte appena prima).</li> <li>Viene quindi generata una rastrelliera vuota e salvata nel database</li> <li>Viene mostrato a schermo un messaggio di aggiunta rastrelliera avvenuta</li> </ol>
<b>Alternativa/e</b>	4a: il numero di morse supera il numero massimo (30) di morse consentite oppure è sotto il numero minimo (15) perciò viene mostrato un messaggio di errore sullo schermo 4b: il numero di morse elettriche supera il numero di morse selezionate e perciò viene mostrato un messaggio di errore sullo schermo
<b>Post-condizioni</b>	Viene creata una rastrelliera vuota con il nome,size scelta e tot morse elettriche scelti e salvata nel database

Per fare in modo che il personale si sbagli il meno possibile nella scelta delle dimensione della rastrelliera e del numero di morse elettriche, per entrambe è previsto mettere una choicebox.

<b>Nome</b>	<b>AGGIUNTA BICICLETTE</b>
<b>Scopo</b>	Aggiungere biciclette a una determinata rastrelliera
<b>Attore/i</b>	PERSONALE DEL SERVIZIO
<b>Pre-condizioni</b>	Il personale del servizio deve essere autenticato
<b>Trigger</b>	L'utente del personale entra nell'opzione del menu "aggiungi/rimuovi biciclette"
<b>Descrizione sequenza eventi</b>	<ol style="list-style-type: none"> <li>Viene scelta una rastrelliera tramite una choicebox</li> <li>Viene scelta la quantità di bici classiche, bici elettriche e bici con seggiolino che si vogliono aggiungere</li> <li>Se la rastrelliera ha abbastanza morse disponibili vengono generate le biciclette scelte e aggiunte alla rastrelliera</li> <li>Viene mostrato a schermo un messaggio di avvenuta aggiunta biciclette</li> </ol>

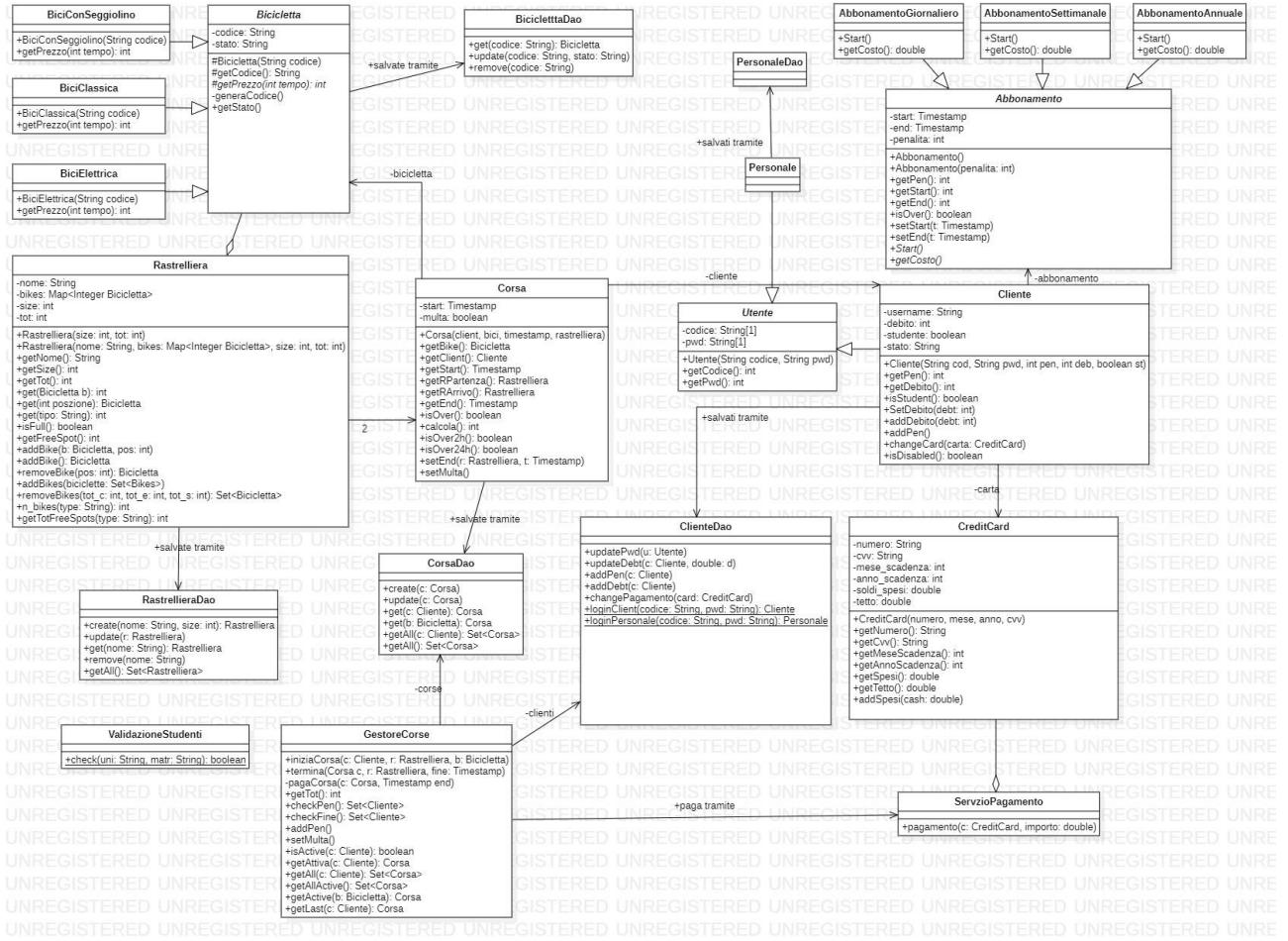
<b>Alternativa/e</b>	4a: la rastrelliera non ha abbastanza morse disponibili e viene mostrato un messaggio di errore
<b>Post-condizioni</b>	Vengono create tot biciclette, salvate nel database e aggiunte alla rastrelliera scelta

Per poter creare una rastrelliera piena di biciclette si dovranno combinare i due scenari.

<b>Nome</b>	<b>SEGNALAZIONE DANNO A BICICLETTA E AZIONI PERSONALE DEL SERVIZIO</b>
<b>Scopo</b>	Segnalare danno a una bicicletta
<b>Attore/i</b>	ABBONATO, PERSONALE DEL SERVIZIO
<b>Pre-condizioni</b>	L'utente deve essere autenticato
<b>Trigger</b>	L'utente del personale entra nell'opzione del menu "SEGNALA DANNO A BICICLETTA"
<b>Descrizione sequenza eventi</b>	<ol style="list-style-type: none"> <li>1. Viene scelto il codice della bici che si vuole segnalare (l'utente viene aiutato mostrandogli le ultime biciclette che ha usato)</li> <li>2. Scrive il danno e lo invia</li> <li>3. Se la bicicletta è in stato 'OK', lo stato cambia e viene riportato il danno segnalato dall'utente. Se invece lo stato è già diverso da 'OK' lo stato riportato diventa il danno segnalato concatenato con gli altri danni segnalati</li> <li>4. Il personale del servizio può scegliere di rimuovere la bici dalla sua rastrelliera, in questo modo non potrà più essere utilizzata.</li> </ol>
<b>Alternativa/e</b>	2a: la bicicletta non esiste e viene mostrato un messaggio di errore 4a: se la bicicletta non è agganciata a una rastrelliera non può essere rimossa
<b>Post-condizioni</b>	Invio danno: viene registrato nel database delle biciclette il danno segnalato e nel caso che c'erano altri danni segnalati si concatena con essi Rimozione bici: l'assegnazione bici rastrelliera viene rimossa

Si può immaginare che quando la bici danneggiata viene rimossa dalla rastrelliera viene portata in magazzino per essere riparata. Questa simulazione, non include un "magazzino delle bici" e perciò nel momento in cui viene disassociata dalla rastrelliera non potrà più essere utilizzata (a meno di riassociala a mano nel database) . L'idea iniziale era infatti quella di eliminarla dal database, ma così facendo anche i dati delle corse venivano compromessi.

## 2.3 Diagramma delle classi (modello di progetto)



### Spiegazione del diagramma:

Le biciclette sono rappresentate tutte da un codice univoco di 7 caratteri e hanno uno stato che è una stringa che può essere settata nel caso che ci sia un danno alla bicicletta, di default è settato a "OK".

I vari tipi di biciclette (**BiciClassica**, **BiciConSeggiolino**, **BiciElettrica**) sono perciò delle classi concrete che estendono la classe astratta **Bicicletta**.

Ogni classe concreta di **Bicicletta** implementa `getPrezzo(int minuti)` che restituisce il prezzo dato il minutaggio. Ogni bici, infatti, avendo una tariffa specializzata implementerà il metodo in modo diverso.

Le varie biciclette sono raggruppate in più rastrelliere. Le rastrelliere sono rappresentate da un nome, che indica anche la sua locazione e da una `Map<Integer, Bicicletta>` che associa ad un intero (posizione sulla rastrelliera) una bicicletta e ha vari metodi osservatori utili per l'implementazione del sistema, come `getFreeSpot(String type)` che restituisce la prima posizione libera adatta per quel tipo di bici, utile per il personale che si occupa della rilocazione, oppure `getBike(String type)` che restituisce la posizione in cui è presente una bicicletta di un certo tipo utile per l'utente che chiede una bici su quella rastrelliera. Inoltre ha anche metodi modificatori come `addBike` e `removeBike` che aggiungono e rimuovono una singola bicicletta dato il tipo o la posizione e `addBikes` e `removeBikes` che servono per aggiungere e togliere più biciclette alla volta.

Ogni utente del sistema di bikesharing ha un codice univoco e una password che sia un cliente oppure membro del personale.

Perciò sia la classe Cliente che la classe Personale implementano la stessa classe astratta Utente.

La classe Cliente ha due attributi Abbonamento e CreditCard entrambi implementati in due classi diverse. L'Abbonamento incapsula e gestisce le informazioni di inizio e fine abbonamento.

Ci sono tre diverse implementazioni di Abbonamento, ovvero AbbonamentoGiornaliero, AbbonamentoSettimanale e AbbonamentoAnnuale, e ogni implementazione fornisce un metodo per iniziare l'abbonamento (e quindi settare il timestamp di inizio e fine) e per fornire il costo. La CreditCard invece, gestisce i dati della carta di credito dell'utente.

Abbonamento inoltre quando viene inizializzato può avere il timestamp di inizio a null, infatti nel caso di abbonamenti giornalieri e settimanali l'abbonamento parte dal primo prelievo. Quando viene effettuato il primo prelievo verrà fatto partire tramite il metodo Start().

Il cliente inoltre ha inoltre l'attributo username, che viene scelto in fase di registrazione e un attributo debito per memorizzare gli eventuali soldi che non è riuscito a pagare al bikesharing. Se il debito è negativo il cliente ha l'abbonamento sospeso.

La classe Corsa rappresenta appunto la corsa di un utente con una certa bicicletta. Ha quindi come attributi la Bicicletta, il Cliente, la Rastrelliera di partenza (solo il nome), la Rastrelliera di arrivo (solo il nome), il timestamp di partenza e il timestamp di arrivo. I nomi delle rastrelliere servono solo come informazione, così da estrarre le informazioni sulla corsa in modo più compatto ed efficiente.

Nel caso che la corsa non sia finita la Rastrelliera di arrivo e il timestamp di fine corsa saranno nulli. Un'istanza della corsa può calcolare quindi il tempo di corsa, il prezzo da pagare (verificando sconti in base al cliente) e altri utili metodi osservatori.

Con il metodo setMultà viene settato un valore booleano che indica che quella corsa è stata multata. Può essere utile per sapere se è già stata controllata già dal server centrale che ha assegnato una multa all'utente e quindi di non controllare ulteriormente questo aspetto a fine corsa oppure per sapere semplicemente se la corsa è stata multata così da poterlo segnalare all'utente.

L'oggetto Corsa ha già tutti gli elementi per capire se la corsa è multata oppure no, ma poiché volevo che la cosa fosse gestita più dall'esterno ho preferito lasciare metodi osservatori per il minutaggio e il metodo SetMultà.

La classe GestoreCorse si occupa di coordinare le operazioni per iniziare e terminare una corsa con i vari aggiornamenti di database (dialogando con oggetti DAO di Cliente e Corsa) e gestione pagamenti.

Inoltre con il metodo SetMultà parsa tutte le corse attive e addebita una multa ai clienti che hanno una corsa superiore alle 24 ore.

Il ServizioPagamento rappresenta un servizio esterno al sistema che effettua le transazioni con le carte di credito. Di fatto aggiorna un database di carte posto all'interno del servizio.

Anche ValidazioneStudenti rappresenta un servizio esterno al sistema che viene chiamato in fase di registrazione per sapere se, l'utente che sta dichiarando di essere uno studente è effettivamente uno studente valido.

I dati persistenti (per Cliente, Personale, Bicicletta, Corsa, Rastrelliera, Carta) sono gestiti da oggetti DAO (Data Access Object) che vengono utilizzati per la gestione dei dati persistenti del sistema.

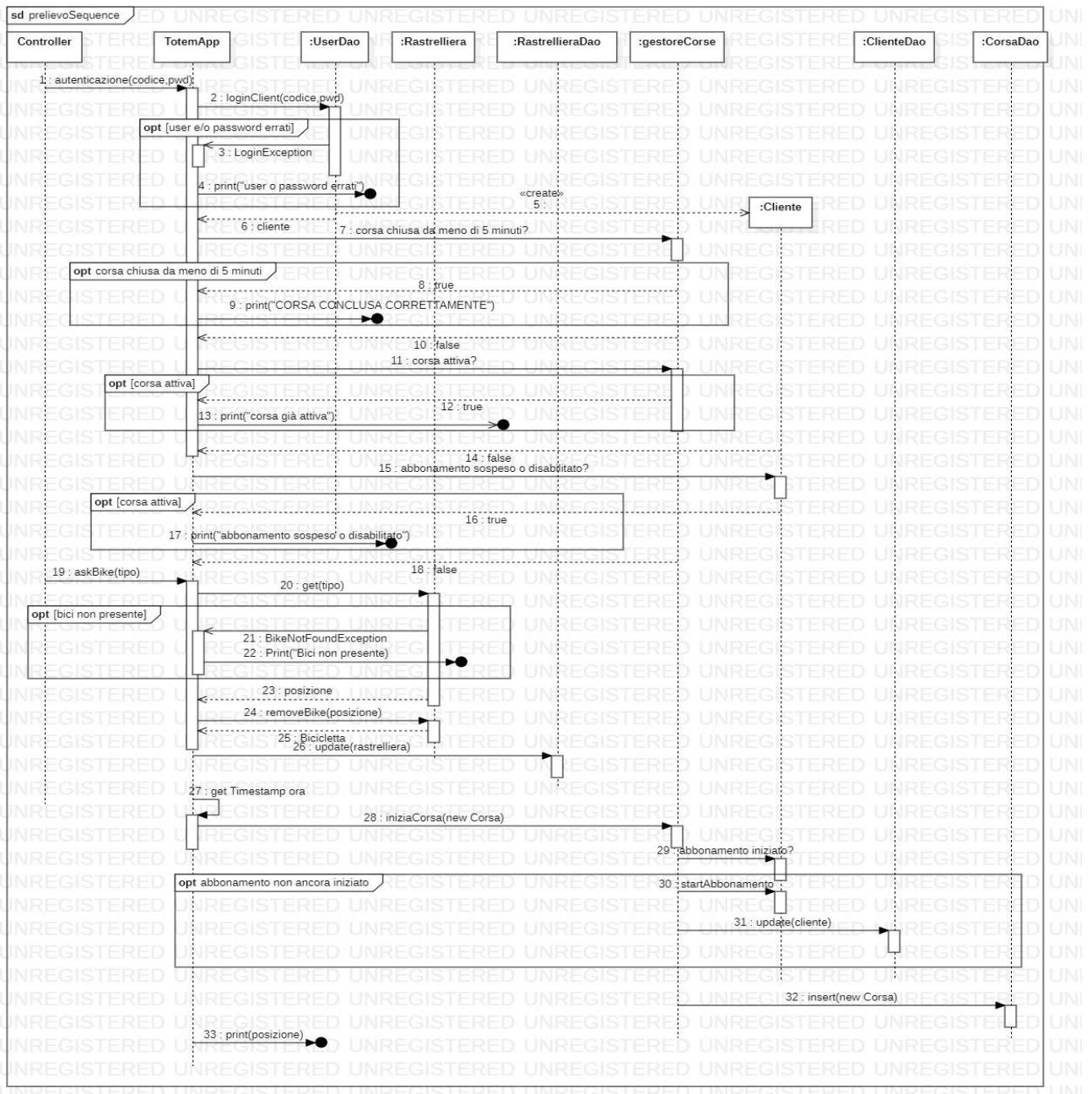
## 2.4 Diagrammi di sequenza

I diagrammi di sequenza, per motivi di lettura del codice più facile, sono descritti utilizzando già le classi e il modello del diagramma delle classi finale contenuto nella sezione 3.

I Controller non vengono specificati, ma generici, per pulizia, poiché a volte ce n'è più di uno.

Servono solo da tramite per generare chiamate a classi del modello del sistema. Le classi "App" sono l'interfaccia per il modello e perciò prendono quasi tutte le chiamate generate dai vari controller e dialogano con tutte le classi del modello del sistema.

### SCENARIO : PRELIEVO BICICLETTA



Spiegazione diagramma:

il TotemApp, dopo che il cliente digita codice e password autentica il cliente tramite UserDao che in caso positivo restituisce un oggetto Cliente.

Dopo aver autenticato il Cliente dialoga con il GestoreCorse per capire lo stato del Cliente che può aver finito una corsa da meno di 5 minuti, o avere ancora una corsa attiva e dialoga con il Cliente per capire se è abilitato. In questo ordine, infatti anche un cliente disabilitato deve avere diritto di agganciare una bicicletta o sapere se l'aggancio di essa è andato a buon fine.

Se nessuna di queste condizioni viene verificata (che terminano di fatto la sequenza), quando viene chiamato il metodo askBike(tipo), nel caso che la rastrelliera associata ha disponibilità vengono fatte le chiamate alla Rastrelliera per rimuovere la bici richiesta e in seguito fatto l'update del database delle rastrelliere tramite RastrellieraDao.

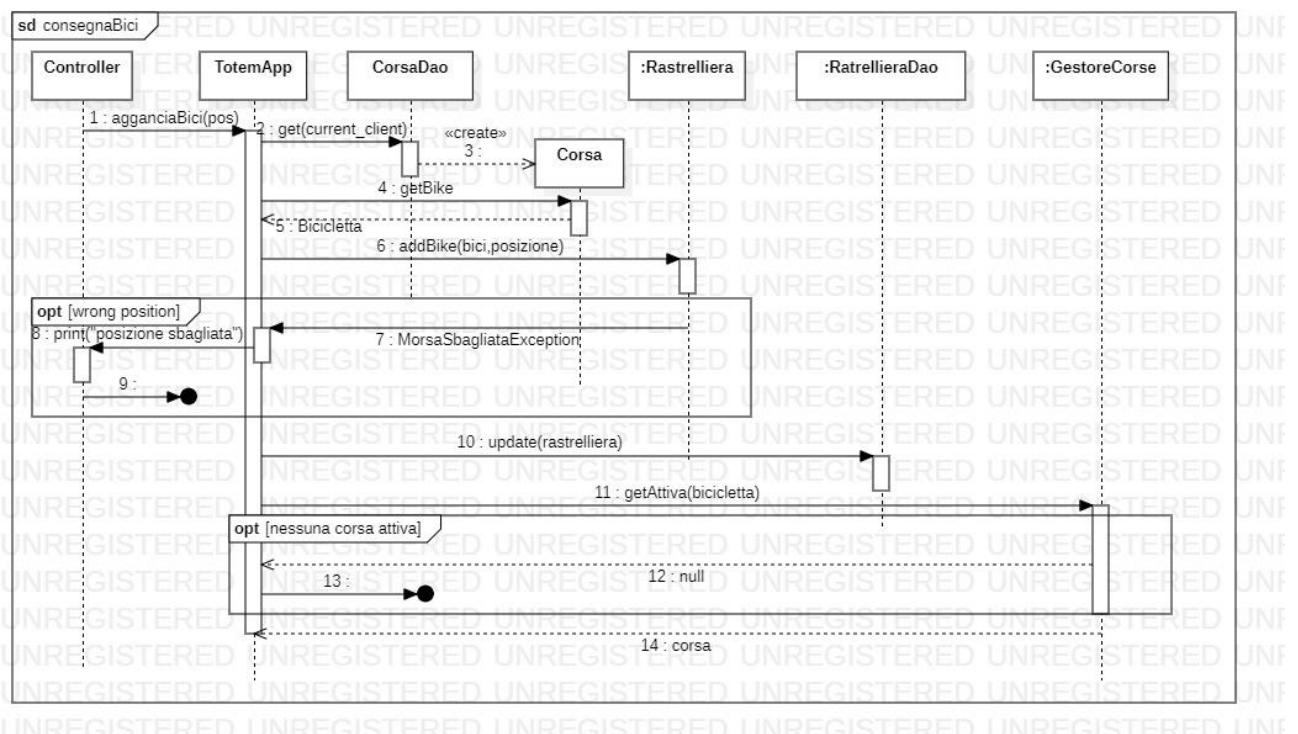
Verrà poi iniziata la corsa chiamando il metodo iniziaCorsa() del GestoreCorse, che inizia l'abbonamento del Cliente se verifica che non è ancora cominciato (abbonamento settimanale o giornaliero) e inserisce la corsa nel database delle corse chiamando CorsaDao.

Viene poi stampata la posizione della bicicletta rimossa a schermo.

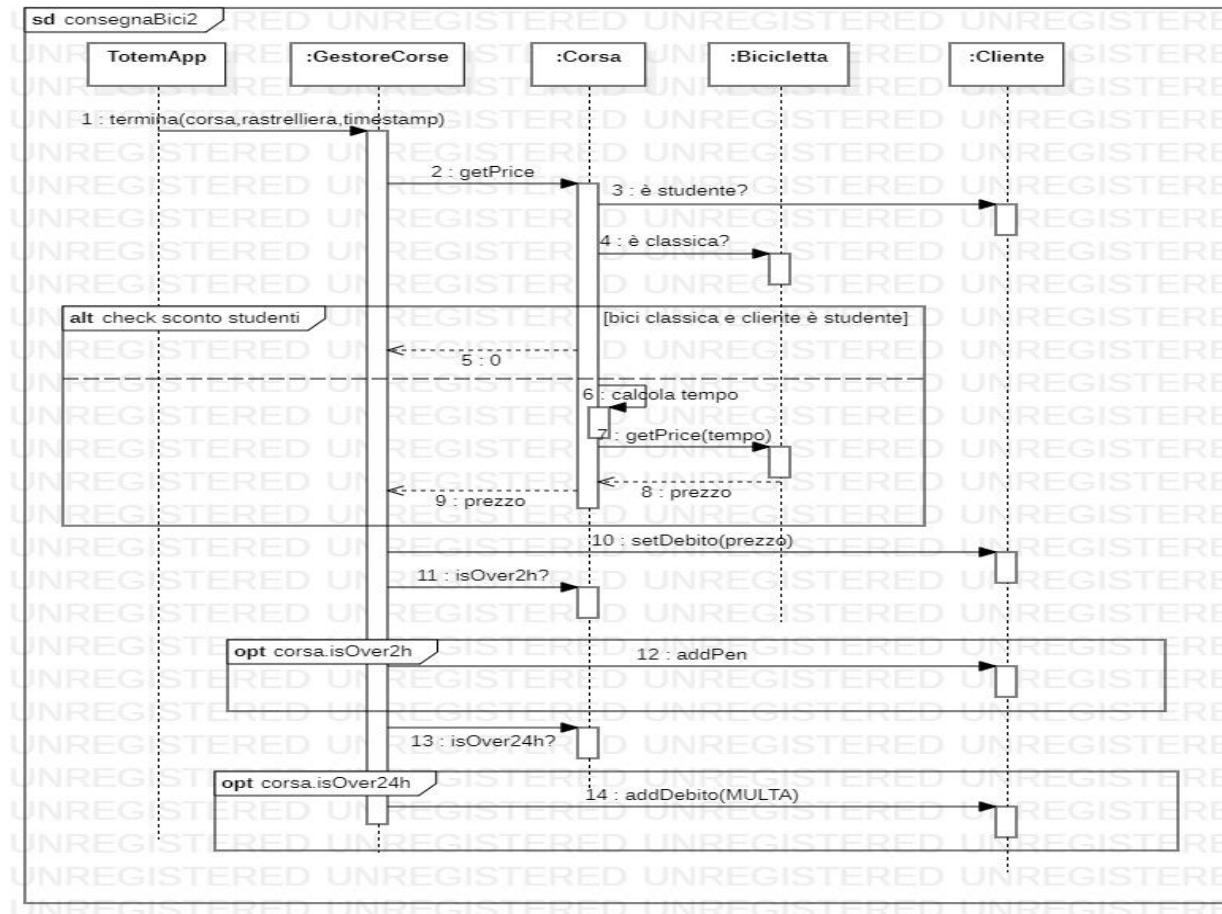
## SCENARIO: CONSEGNA BICI

Ho deciso di dividere il diagramma di sequenza in 3 parti per questioni di leggibilità, perciò per ogni parte mostro solo le classi coinvolte.

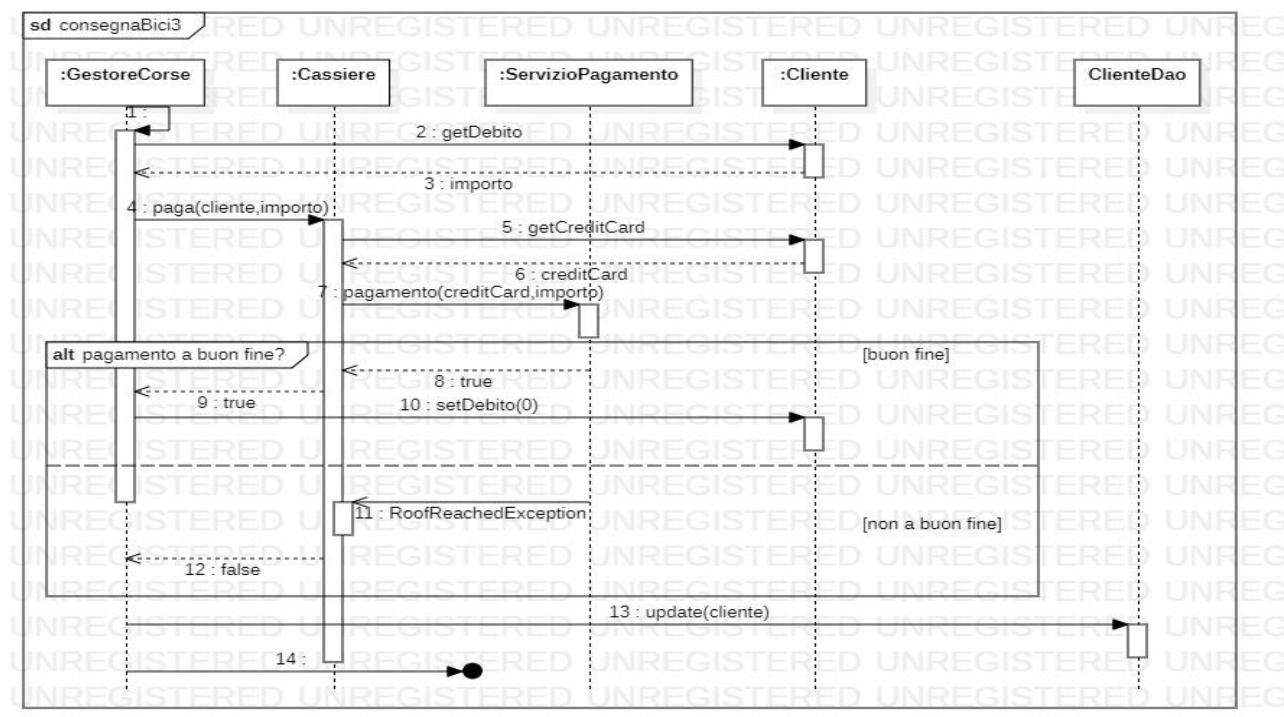
### PARTE 1 (aggancio bicicletta)



## PARTE 2 (terminazione corsa)



## PARTE 3 (pagamento corsa)



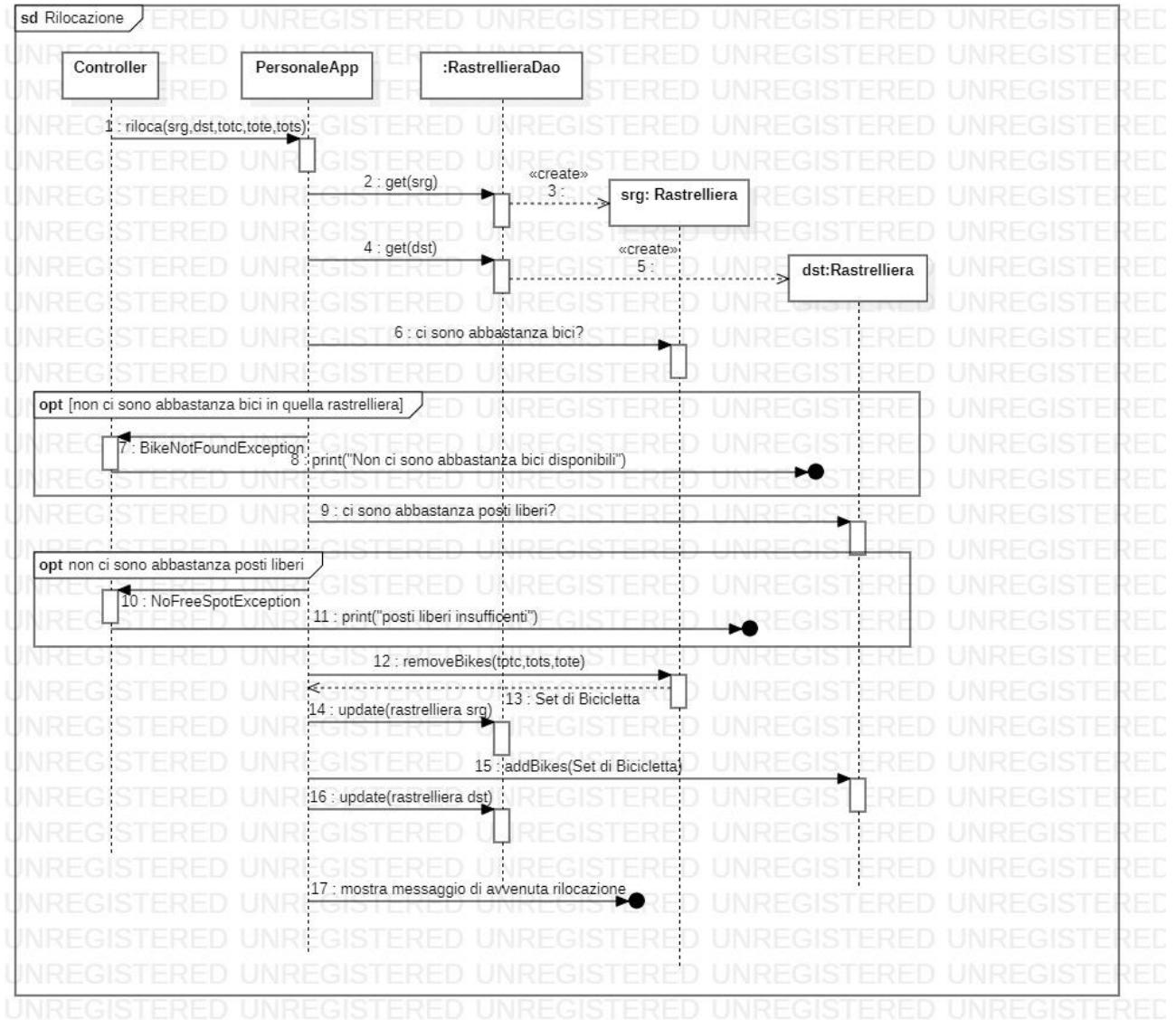
### **ELEMENTI DI SIMULAZIONE:**

All'inizio della sequenza il controller chiama la funzione agganciaBici(int posizione) del TotemApp. In uno scenario reale si presume che un qualche sensore comunichi al totem il codice o un identificativo della bicicletta che viene agganciata. In questo caso perciò per simulare la realtà all'inizio il totemApp dovrà chiedere al DAO di Corsa la corsa attiva dell'utente che sta consegnando la bicicletta e da lì capire la bicicletta che si sta andando a consegnare.

### **SPIEGAZIONE DIAGRAMMA:**

Nel caso che la morsa alla quale l'utente sta agganciando la bici l'oggetto Rastrelliera manda una MorsaSbagliataException che indica che il tipo di bicicletta non è adatto al tipo di morsa. Si provvede quindi a mandare un messaggio di errore sullo schermo e a terminare la sequenza. Se la morsa è corretta la prima cosa che viene fatta è comunicare all'oggetto Rastrelliera associato al Totem l'aggiunta della bici e l'update del database delle rastrelliere tramite RastrellieraDao. Viene quindi estratta tramite GestoreCorse la corsa associata a quella bicicletta e estratto il prezzo (l'oggetto corsa provvede anche a verificare se c'è uno sconto studenti). Dialogando con l'oggetto Corsa si capisce anche se dare una penalità o una multa all'utente. Se la multa è già stata settata nella corsa voldire che il sistema ha già provveduto ad essa e non si fa nulla. Una volta quindi settato il debito dell'utente si provvede al pagamento tramite la classe Cassiere che comunica con il ServizioPagamento. Se il pagamento va a buon fine viene settato il debito dell'utente a 0. Come ultima cosa viene quindi fatto l'update del Cliente tramite ClienteDao.

## SCENARIO: RILOCAZIONE BICICLETTE

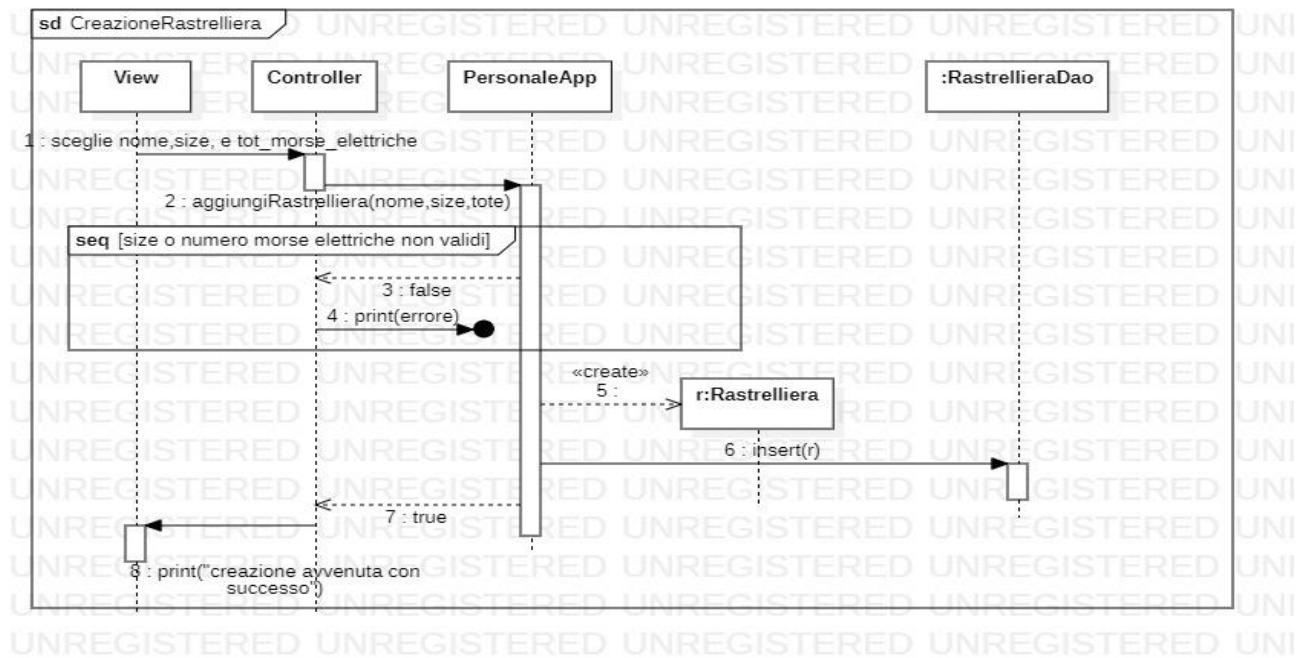


### SPIEGAZIONE DIAGRAMMA:

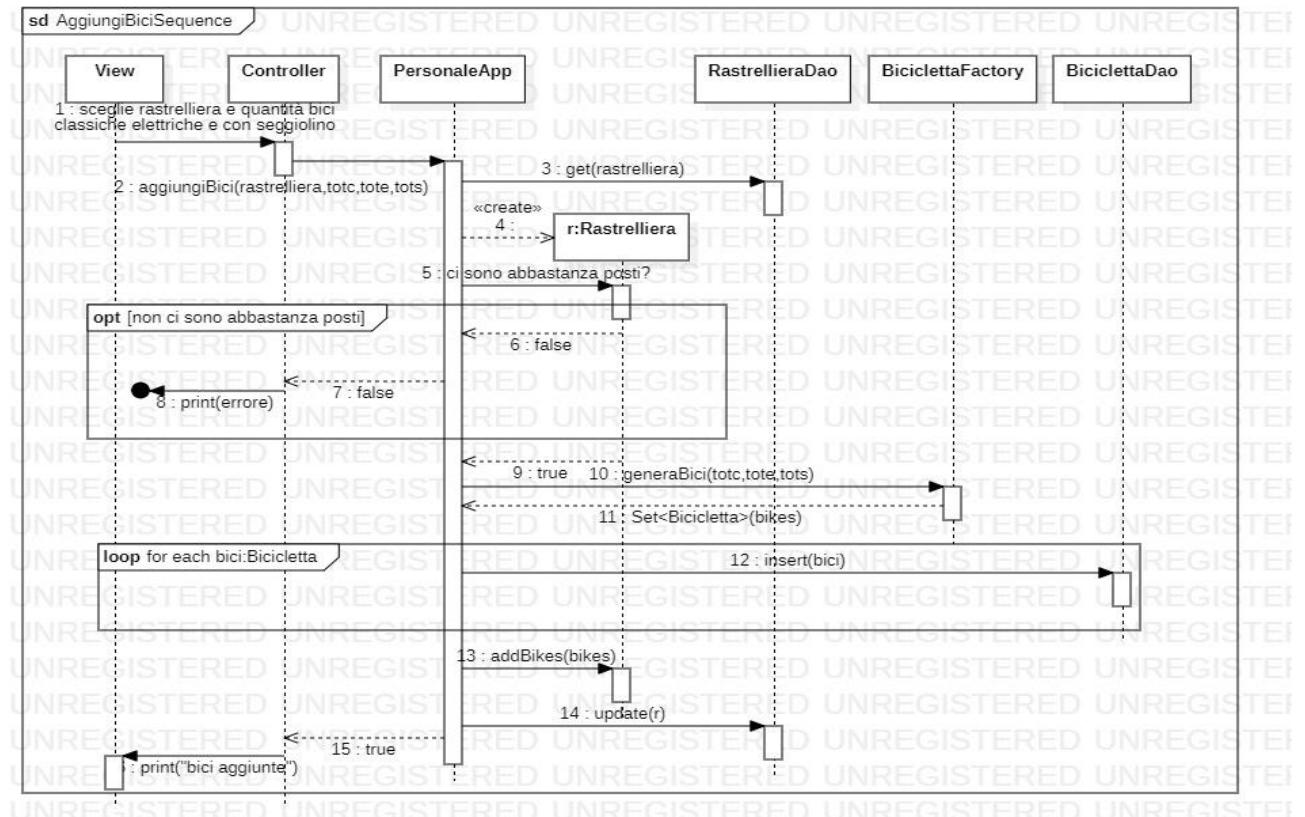
Il controller chiama la funzione `riloca()` del PersonaleApp che prende come argomento un oggetto `Rastrelliera` (sorgente) un oggetto `Rastrelliera` (destinazione) e 3 interi che rappresentano il numero di biciclette classiche, di biciclette elettriche e di biciclette classiche che si vogliono spostare dalla rastrelliera sorgente alla rastrelliera destinazione.

Dopo aver estratto le due rastrelliere tramite l'oggetto `RastrellieraDao` vengono utilizzati i metodi osservatori dei due oggetti `Rastrelliera` per capire se ci sono abbastanza biciclette nella rastrelliera sorgente e abbastanza posti nella rastrelliera destinazione (in caso contrario viene terminata la sequenza). A questo punto si utilizzano i metodi `removeBikes()` che rimuove le biciclette richieste e restituisce il `Set<Bicicletta>`, rappresentante delle bici rimosse, e `addBikes()` che aggiunge il `Set<Bicicletta>` alla rastrelliera. Viene quindi fatto l'update delle rastrelliere tramite un oggetto `RastrellieraDao` e viene così completata la rilocazione.

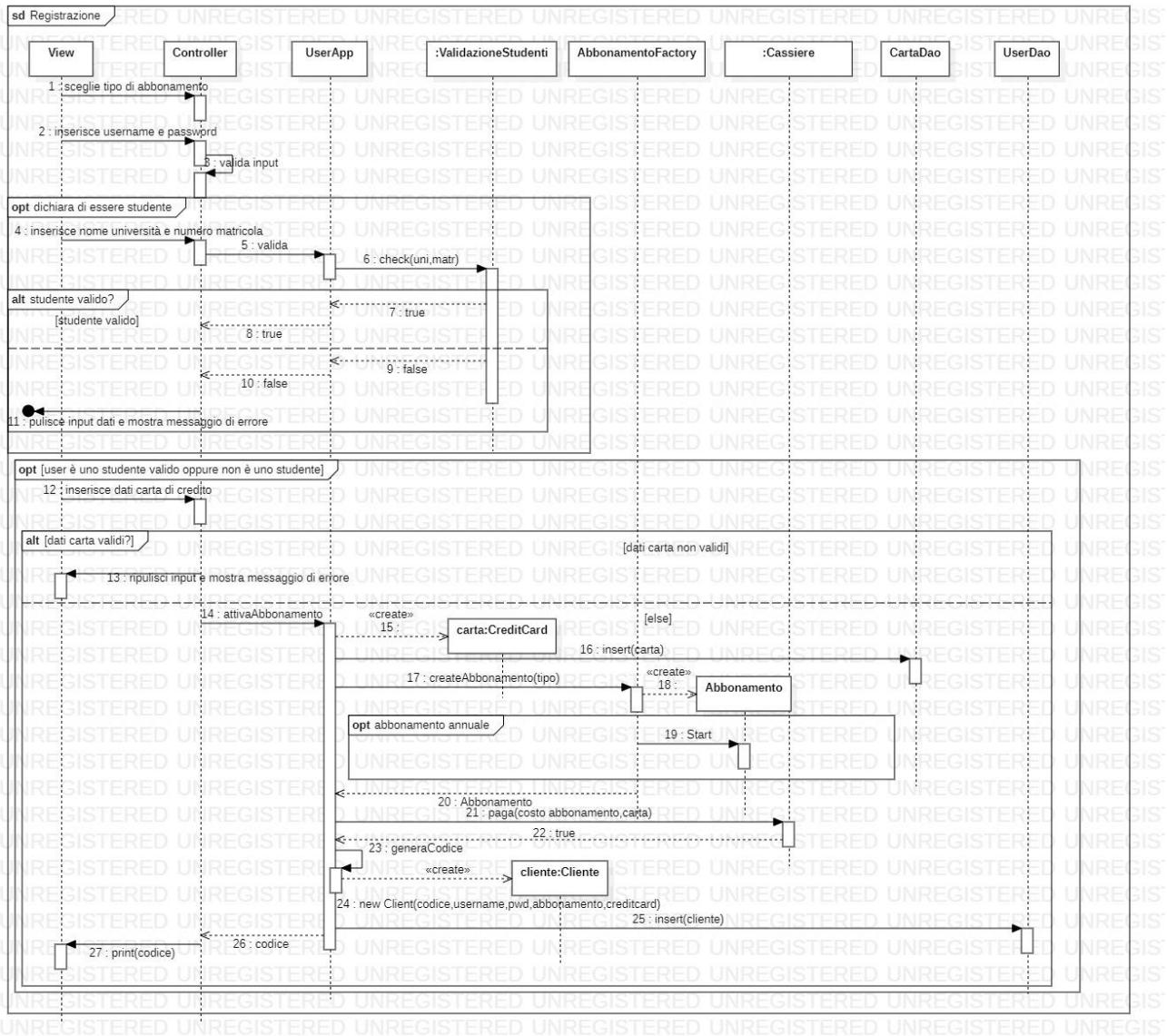
## SCENARIO: CREAZIONE RASTRELLIERA



## SCENARIO: AGGIUNTA BICICLETTE



## SCENARIO: REGISTRAZIONE



### SPIEGAZIONE DIAGRAMMA:

All'inizio della registrazione l'utente sceglie il tipo di abbonamento, username, password. Gli input vengono validati dal Controller. Viene poi eventualmente validato lo status di studente tramite una chiamata a UserApp che chiama il metodo `valida()` dell'oggetto `ValidazioneStudente`.

Inseriti poi i dati della carta viene fatta una chiamata a UserApp del metodo `attivaAbbonamento` che provvede a creare un oggetto `CreditCard`.

Elemento di simulazione: la carta viene salvata nel database delle carte tramite l'oggetto `CartaDao` che inserisce una carta che ha 0 come valore per i soldi spesi e 200 come tetto massimale. Nella realtà questa informazione non si dovrebbe sapere.

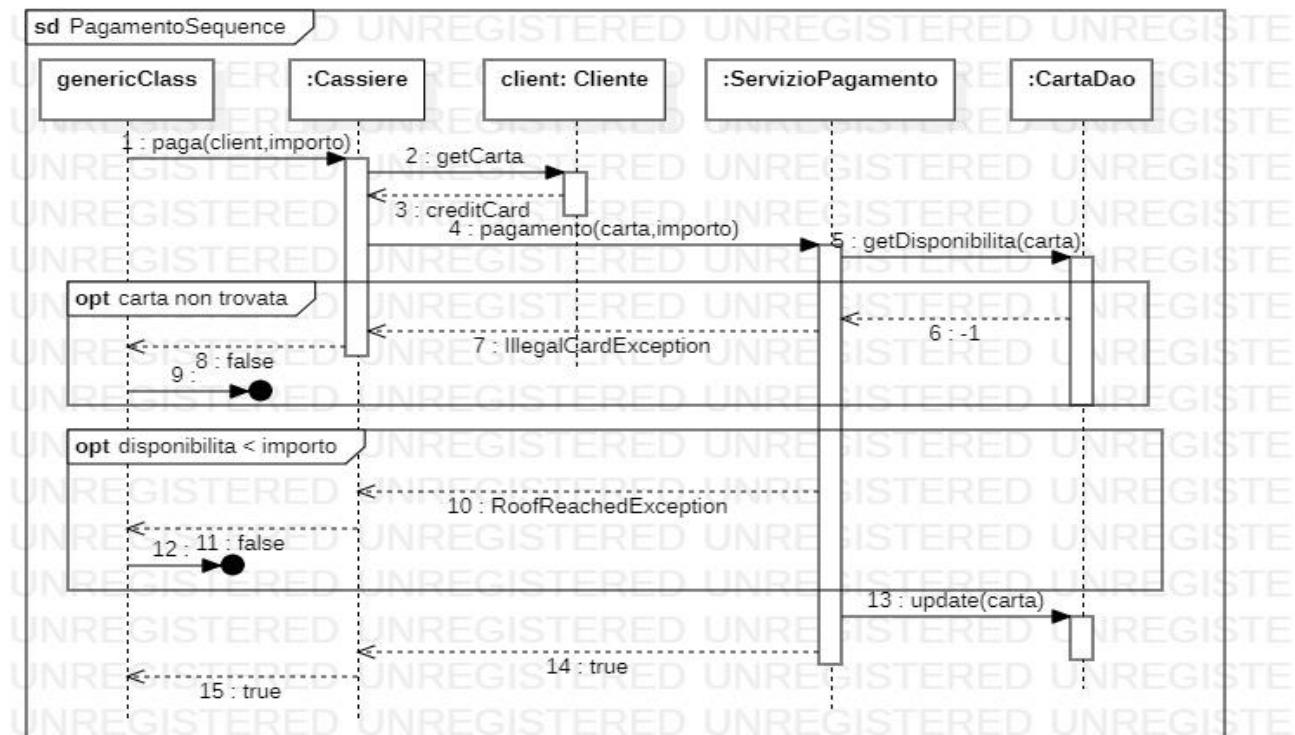
Viene poi creato un oggetto `Abbonamento` tramite una chiamata a `AbbonamentoFactory` del metodo `createAbbonamento(tipo_abbonamento)`, che nel caso che sia un abbonamento annuale provvede anche a farlo partire.

A questo punto viene tentato il pagamento. Nella sequenza viene mostrato un pagamento andato a buon fine, ma viene verificato che sia andato a buon fine tramite la chiamata all'oggetto `Cassiere` che si occupa di gestire i pagamenti dialogando con l'oggetto `ServizioPagamento` (nel prossimo diagramma spiego come funzionano i pagamenti in generale).

Viene quindi generato un codice univoco per l'utente e creato un oggetto Cliente che viene quindi inserito nel database tramite l'oggetto ClienteDao.

Il pagamento viene, per motivi di simulazione interamente gestito dal sistema. La classe Cassiere fa da tramite per dialogare con ServizioPagamento che rappresenta un servizio esterno, in modo da rendere più flessibile l'accoppiamento tra il servizio esterno e il sistema che dovrà quindi solo continuare a chiamare le stesse funzioni di Cassiere.

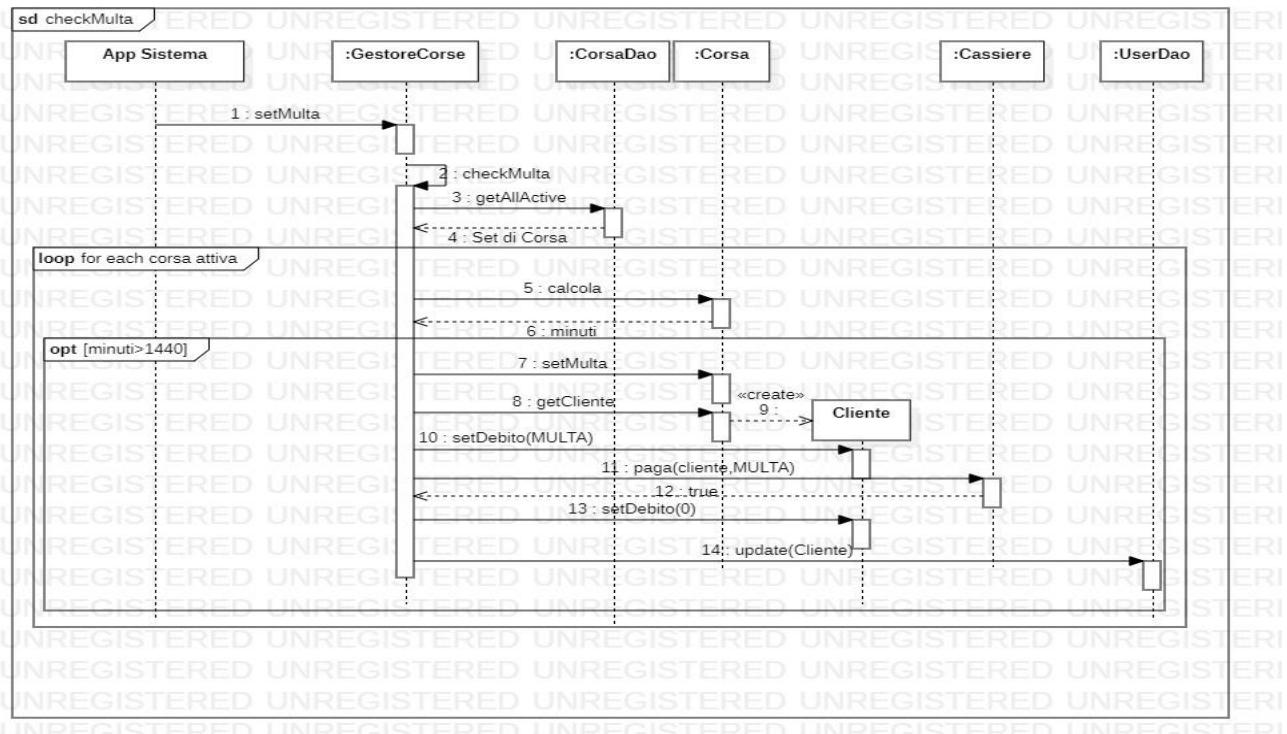
### SCENARIO:PAGAMENTO



### SPIEGAZIONE DIAGRAMMA:

Il ServizioPagamento come già accennato in precedenza, per simulare una transazione, aggiornerà il database delle carte (che è lo stesso utilizzato dal sistema per estrarre informazioni delle carte). La prima cosa che chiederà a CartaDao è la disponibilità di quella carta che verrà calcolata come (tetto – soldi\_spesi). Se la disponibilità è minore dell'importo che si vuole addebitare la transazione viene negata (RoofReachedException). Altrimenti viene fatto l'update della carta aggiungendo l'importo e Cassiere riporterà true come valore finale.

## SCENARIO: CONTROLLO E ASSEGNAMENTO MULTE

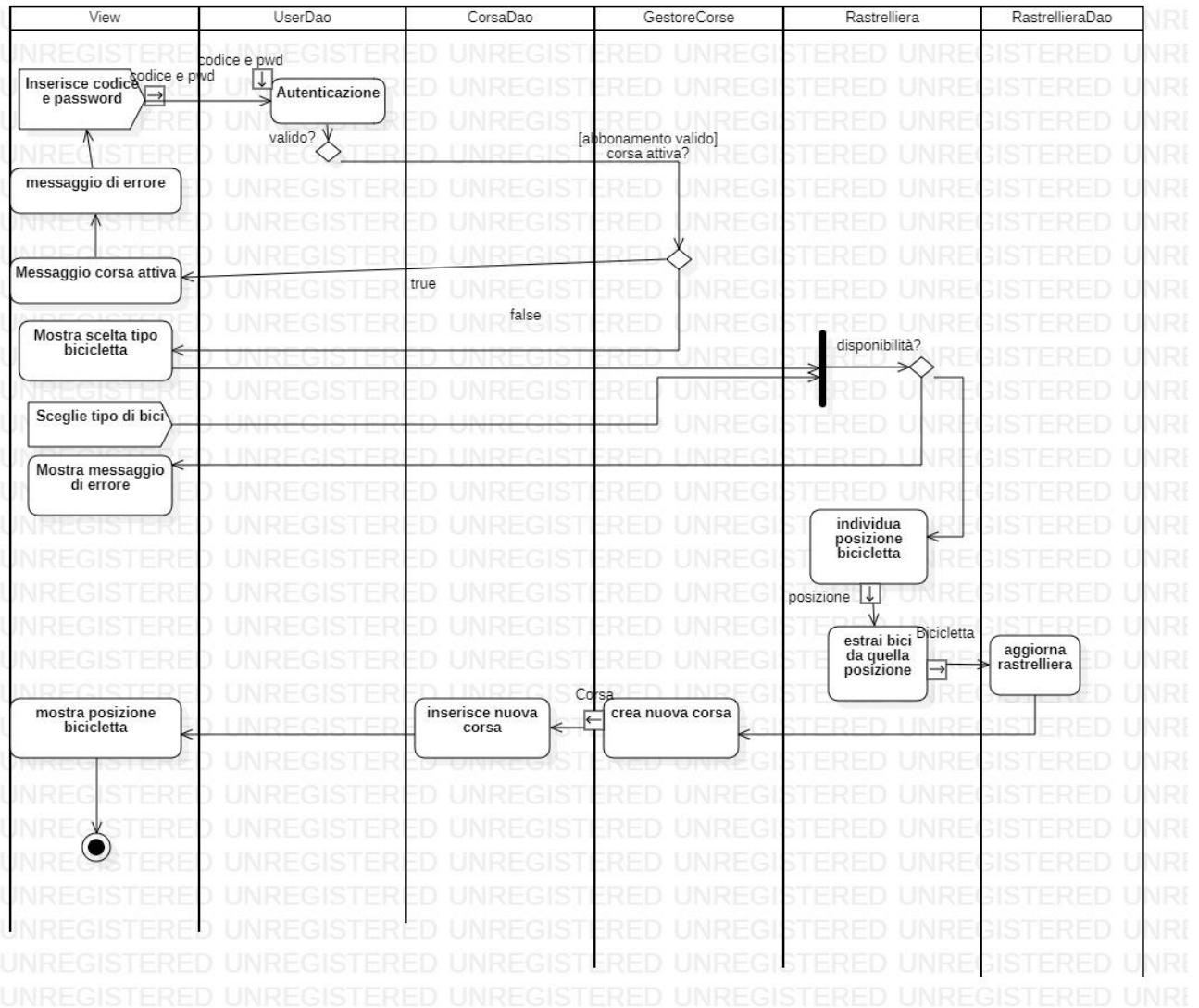


### SPIEGAZIONE DIAGRAMMA:

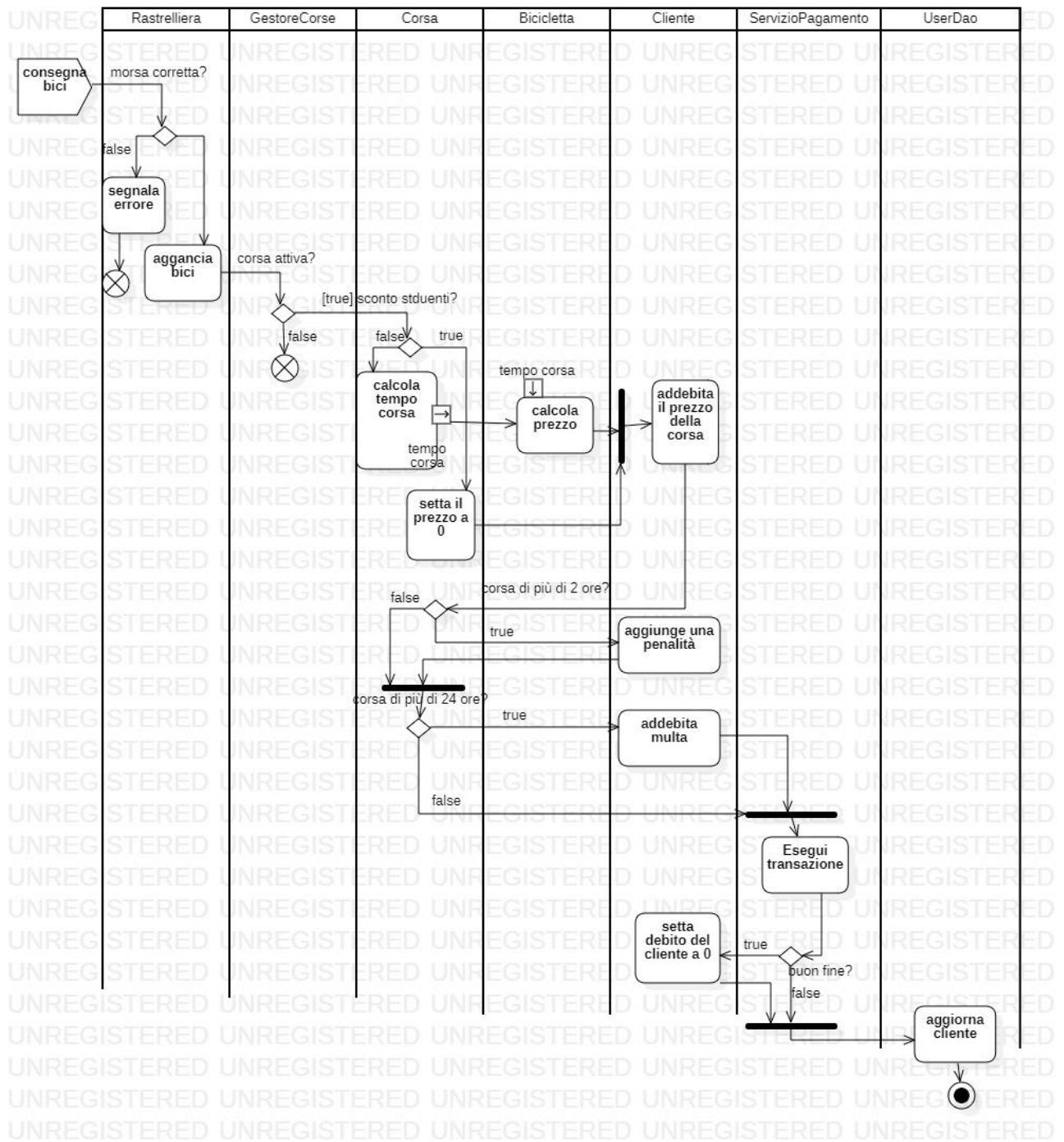
Questa sequenza è svolta dall'app del sistema in loop (ogni minuto).

## 2.5 Diagrammi delle attività

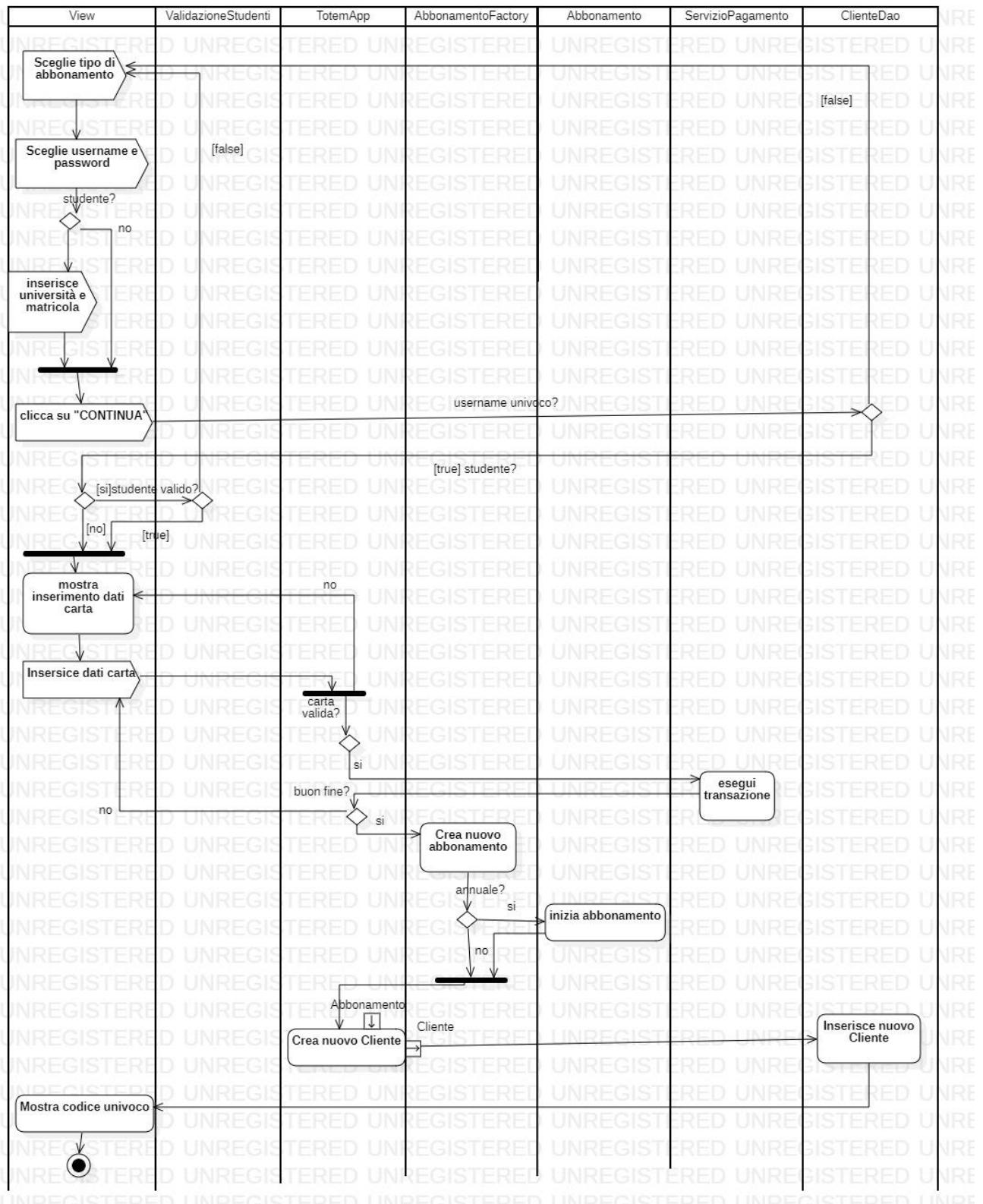
### SCENARIO: PRELIEVO BICI



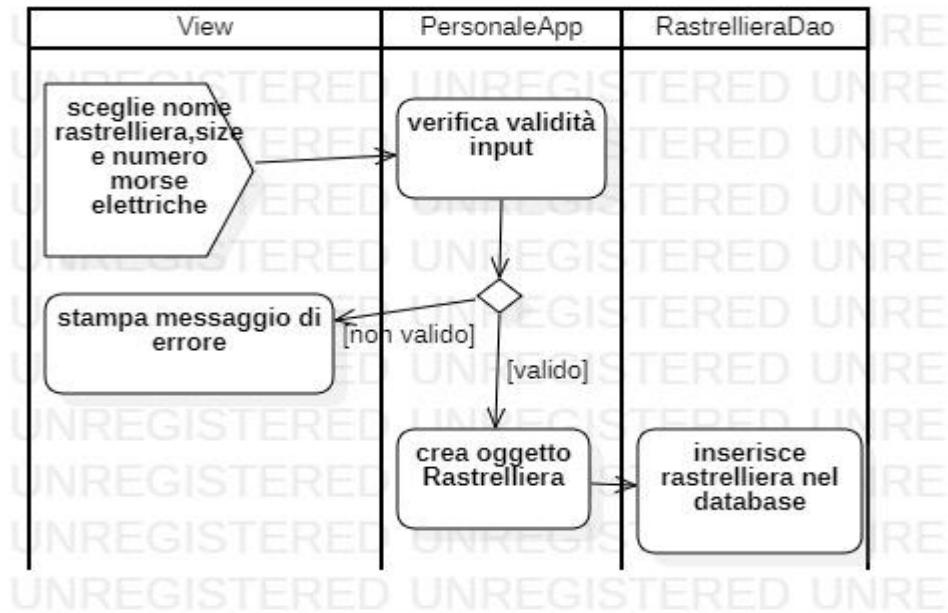
## SCENARIO: CONSEGNA BICI



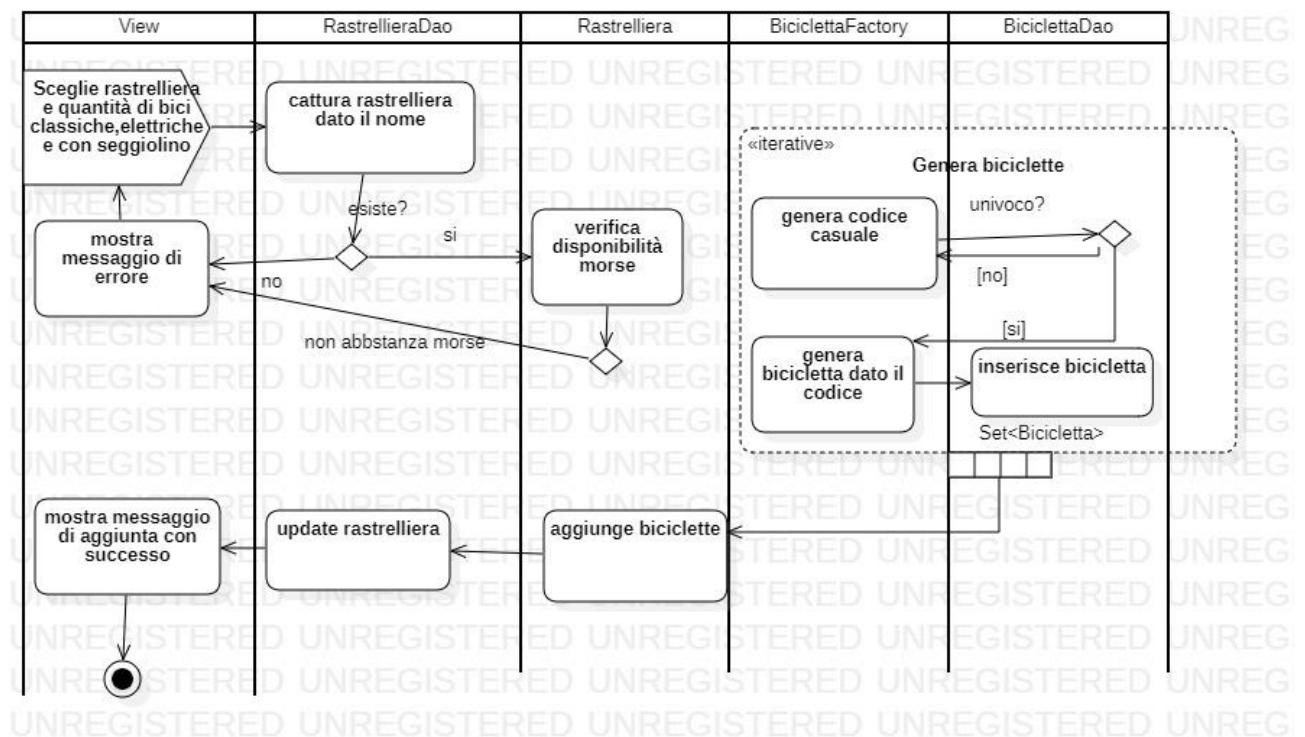
## SCENARIO:REGISTRAZIONE



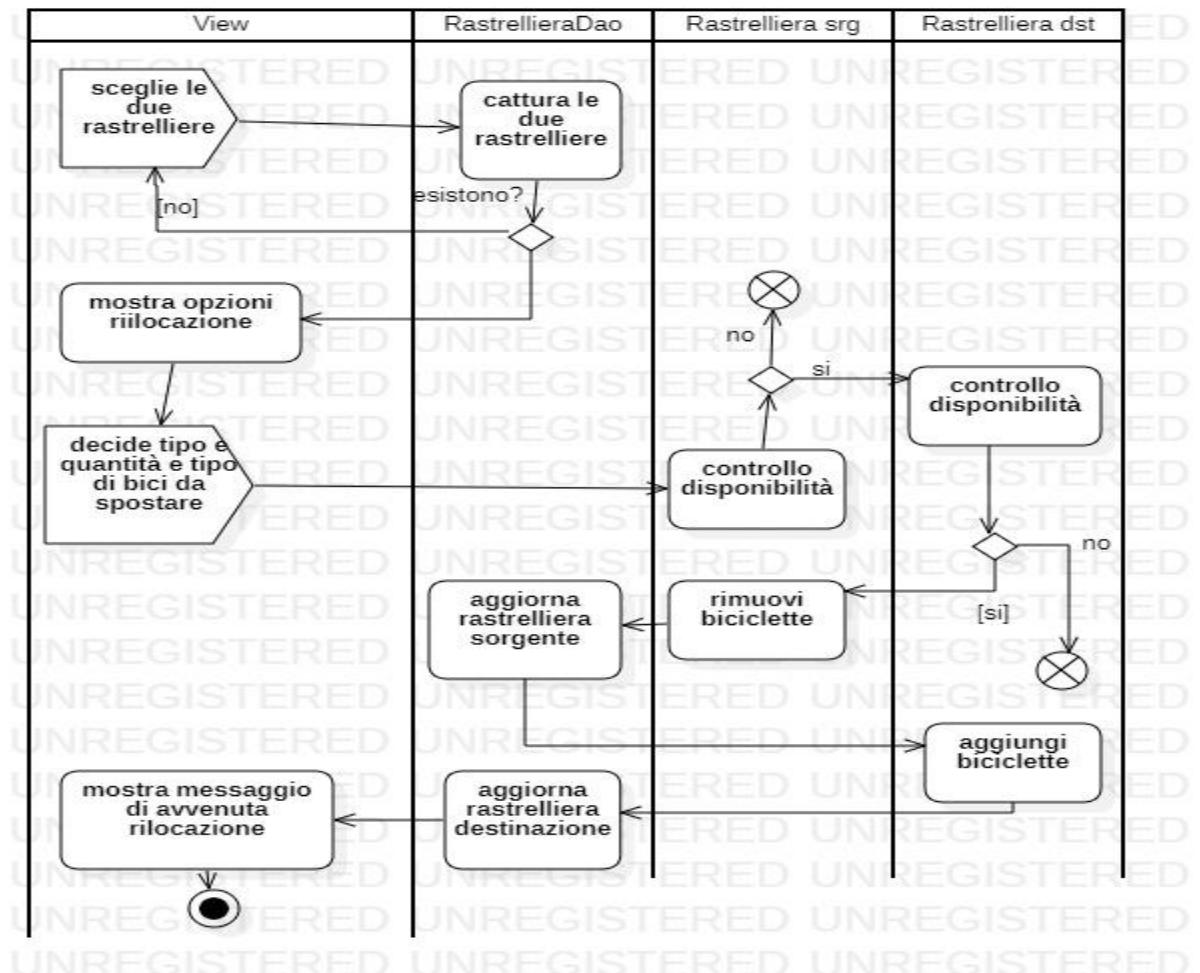
## CREAZIONE RASTRELLIERA



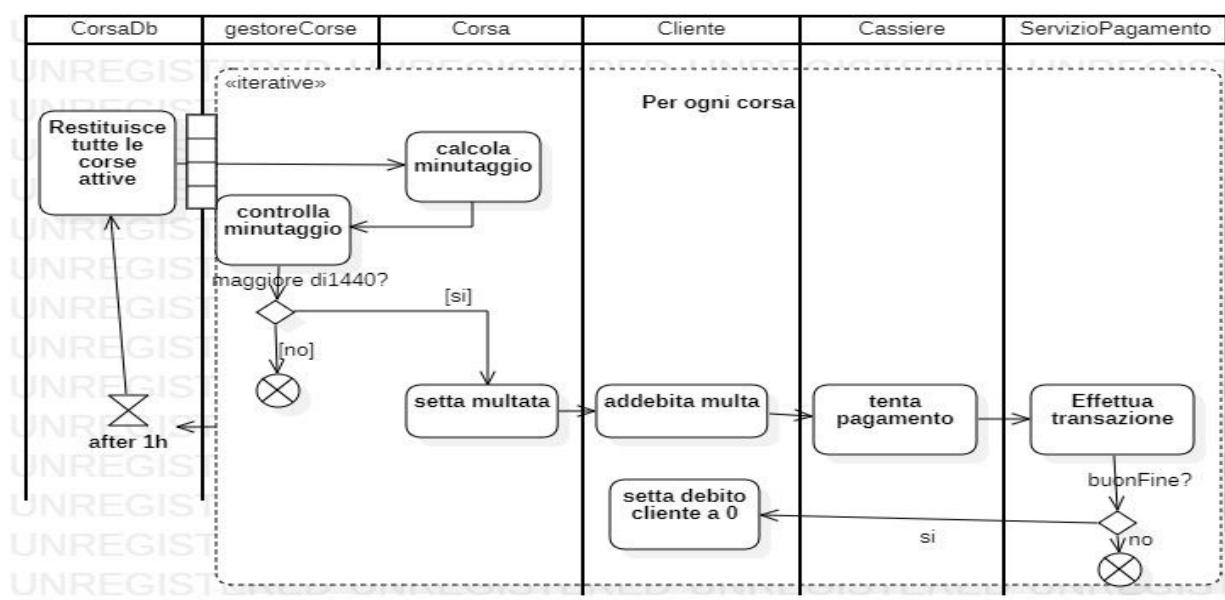
## AGGIUNTA BICICLETTE



## SCENARIO: RILOCAZIONE



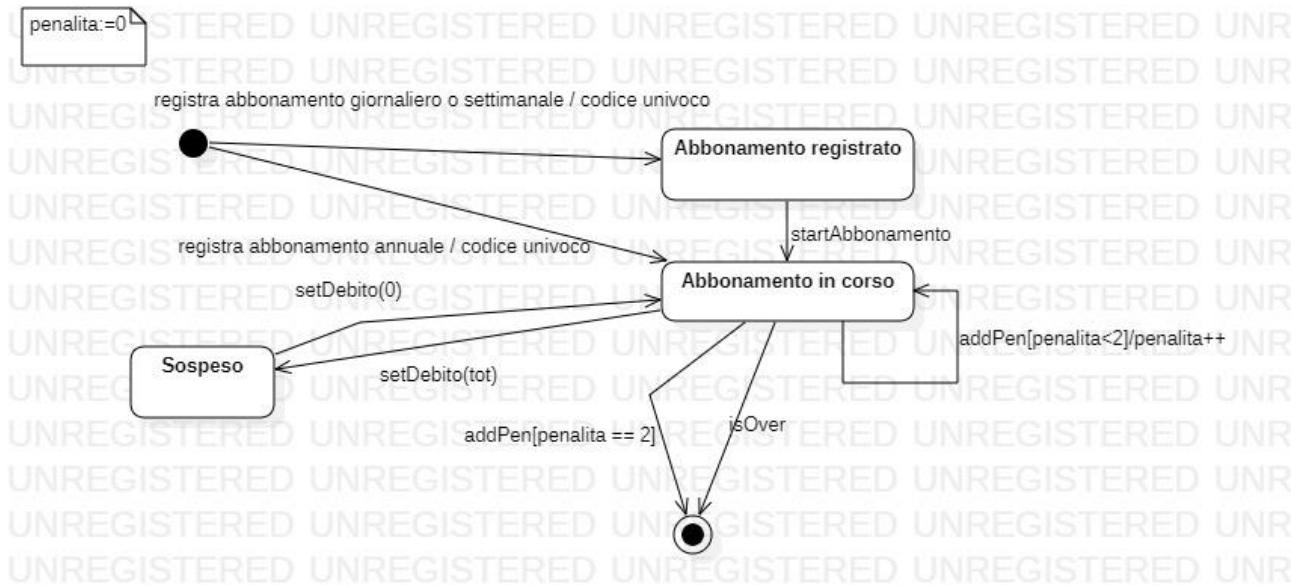
## SCENARIO: CONTROLLO E ASSEGNAZIONE MULTA



## 2.6 Macchine di stato

Tutte le transazioni da uno stato all'altro in questa macchina di stato avvengono tramite call event, ovvero a chiamate a funzioni dell'oggetto in questione.

### Macchina di stato per abbonamento Cliente



Per abbonamento si intende l'abbonato al quale è associato un abbonamento bikesharing

Lo stato iniziale dell'abbonamento si intende quando ancora non è stato registrato.

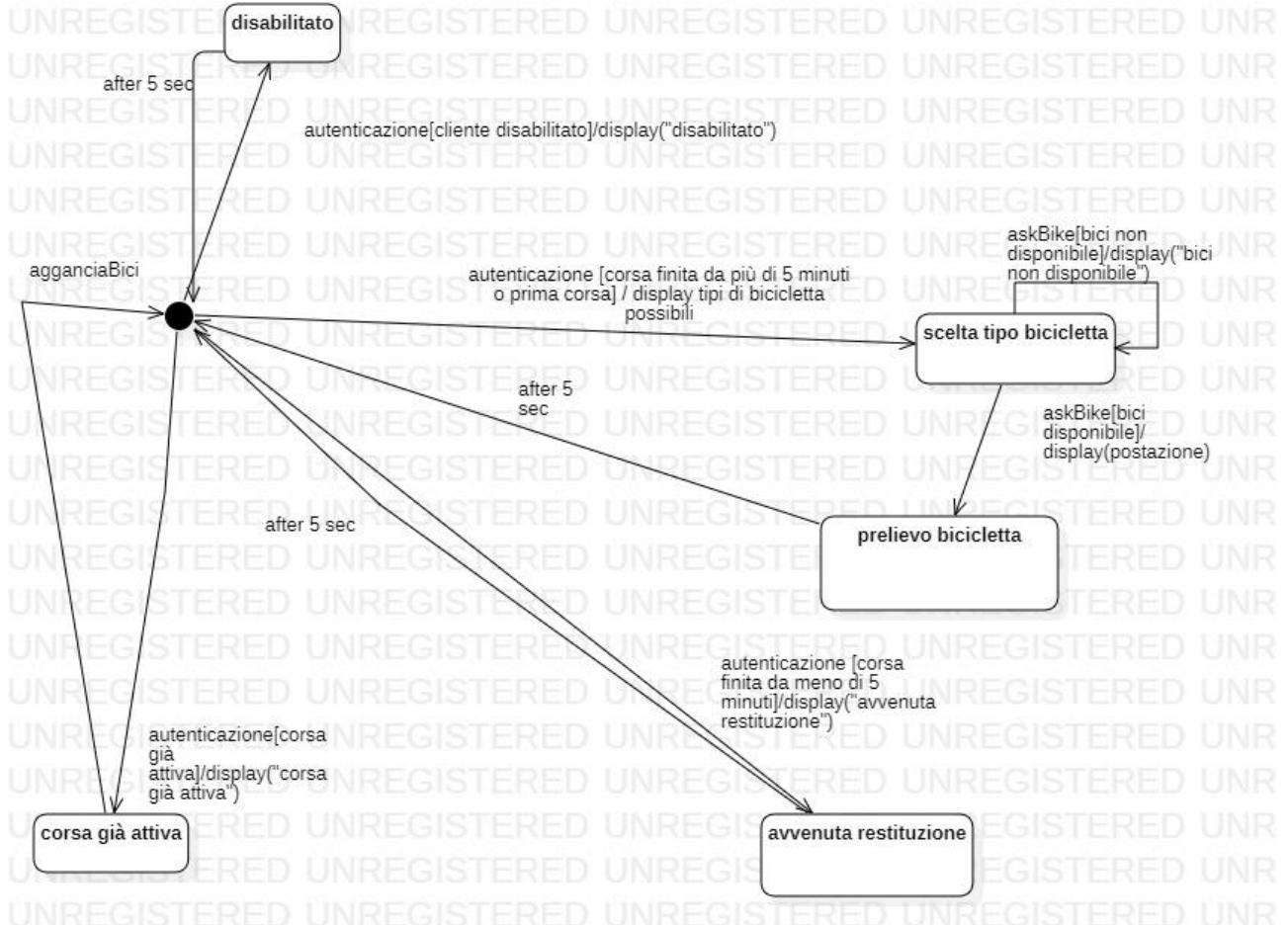
Nel momento in cui viene registrato un abbonamento viene mostrato un codice univoco. Se l'abbonamento registrato è di tipo annuale comincia subito e perciò si passa allo stato "Abbonamento in corso", se invece è settimanale o giornaliero si aspetta la prima corsa e si rimane allo stato "Abbonamento Registrato". Quando il cliente preleverà la sua prima bicicletta verrà chiamato startAbbonamento che comincerà l'abbonamento verrà settata la data di scadenza e si passa allo stato "Abbonamento in corso"

STATO Sospeso: ci si passa ogni volta che il cliente deve fare un pagamento e perciò viene settato un debito del cliente. Tentando poi di pagare, se il pagamento va a buon fine, il debito del cliente verrà settato a 0 (tramite setDebito(0) ) e ritornerà a "Abbonamento In Corso".

Allo stato finale dell'abbonamento e perciò allo stato irreversibile ci si passa se il cliente ha un abbonamento scaduto oppure ha egualgiato o superato le 3 penalità

Le transazioni di questa macchina di stato avvengono tramite call event oppure tramite eventi temporali (after 5 sec).

## Macchina di stato del TotemApp

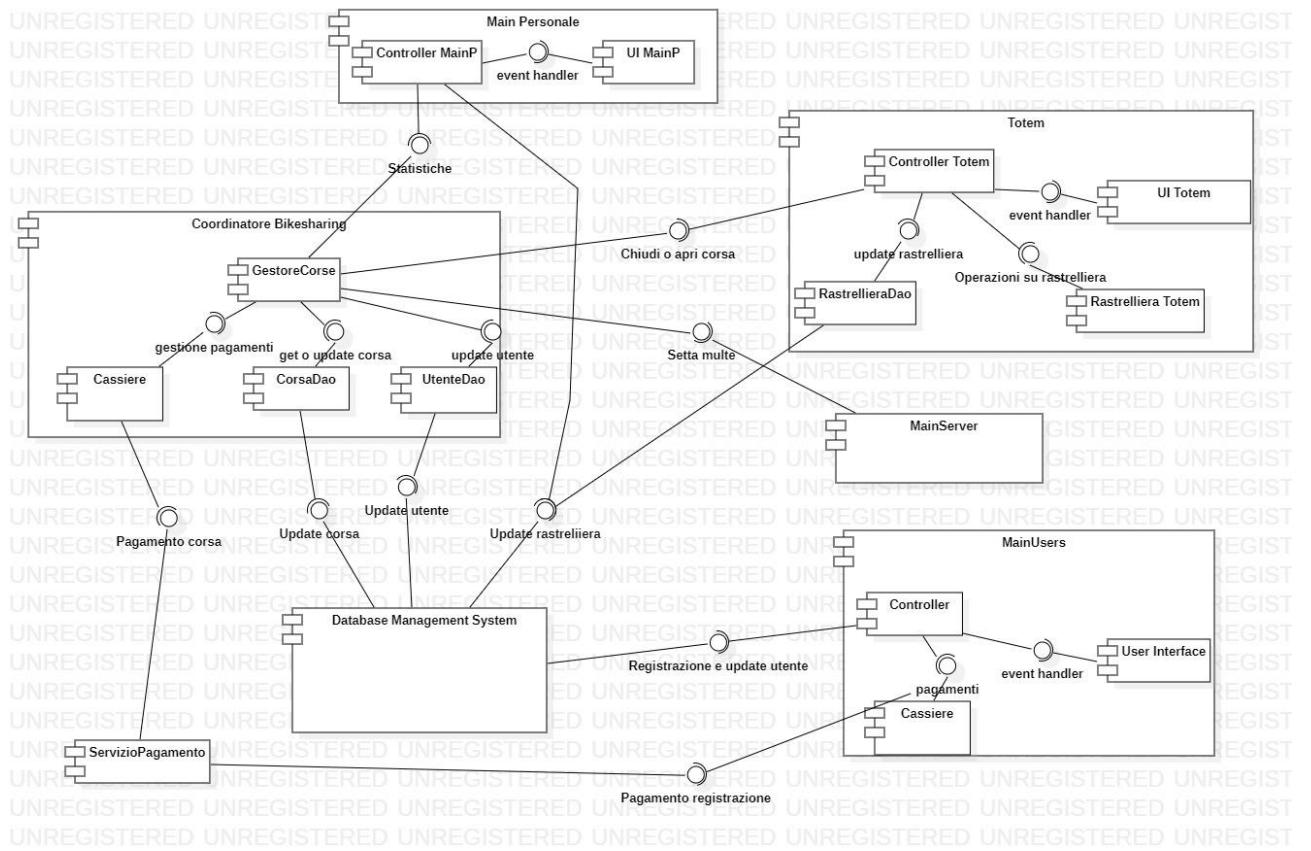


Questa macchina di stato rappresenta gli stati che può assumere un totem nei confronti di un cliente che vuole prelevare una bicicletta a seconda del suo stato nel servizio bikesharing.

Gli stati possibili sono:

- Stato iniziale: il totem aspetta un login
- “disabilitato”: il totem rimane per 5 secondi in uno stato in cui non può operare perché il cliente è disabilitato e mostra al cliente un messaggio di errore.
- “corsa già attiva”: il totem passa in uno stato in cui chiede al cliente dove vuole aggangiare la bicicletta. Se il cliente quindi aggancia la bici (call event: agganciaBici) ritorna allo stato iniziale
- “avvenuta restituzione”: il totem rimane per 5 secondi in uno stato in cui non può operare perché il cliente ha finito da poco una corsa e perciò mostra un messaggio di avvenuta restituzione al cliente
- “scelta tipo bicicletta”: nessuno dei casi precedenti viene attivato e il totem passa in uno stato di scelta tipo bicicletta dove viene richiesto all’utente il tipo di bicicletta. Una volta scelta il totem mostrerà per 5 secondi la posizione della bici scelta e infine tornerà nello stato iniziale.

## 2.7 Diagramma dei componenti

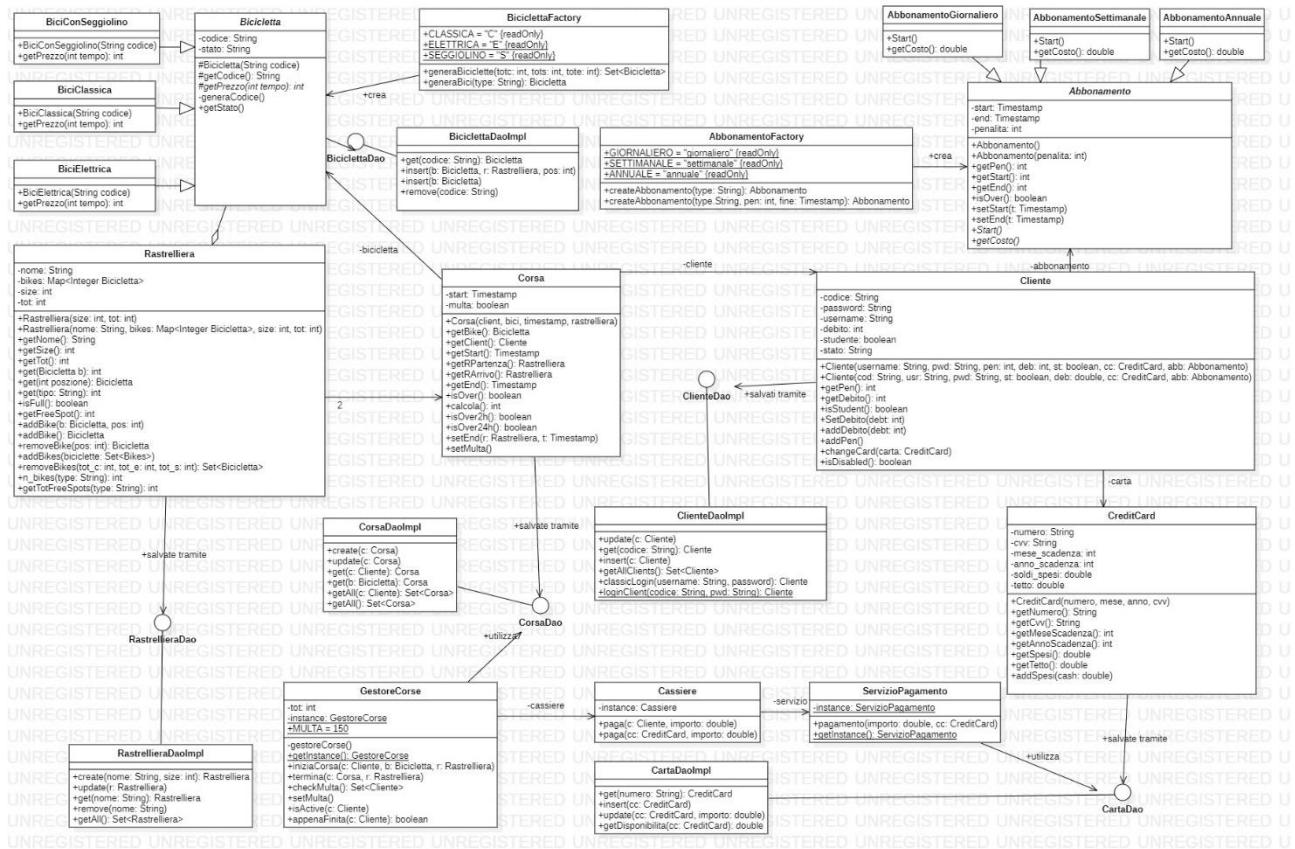


### 3 Implementazione del sistema

#### 3.1 Diagramma delle classi (modello di programma)

Per questioni di ordine ho deciso di fare 4 diagrammi delle classi per rappresentare la relazioni tra le varie classi nel modello di programma.

#### DIAGRAMMA DEL MODEL



Questo è il diagramma delle classi definitivo per quanto riguarda il “model” dell’applicazione.

Queste classi verranno poi utilizzate per implementare l’effettivo utilizzo di essa.

Le classi DAO sono state sostituite da un’interfaccia e da un’implementazione di essa, in modo da rendere più lasco l’accoppiamento tra le classi e il Data Access Object con cui comunicano.

Sono state aggiunte due nuove classi Factory (BiciclettaFactory e AbbonamentoFactory) che rendono più facile la creazione di abbonamenti e biciclette (dato il tipo di bicicletta e il tipo di abbonamento). BiciclettaFactory inoltre dialoga anche con il DAO di Bicicletta in quanto ha anche la responsabilità di generare un codice univoco per una bicicletta.

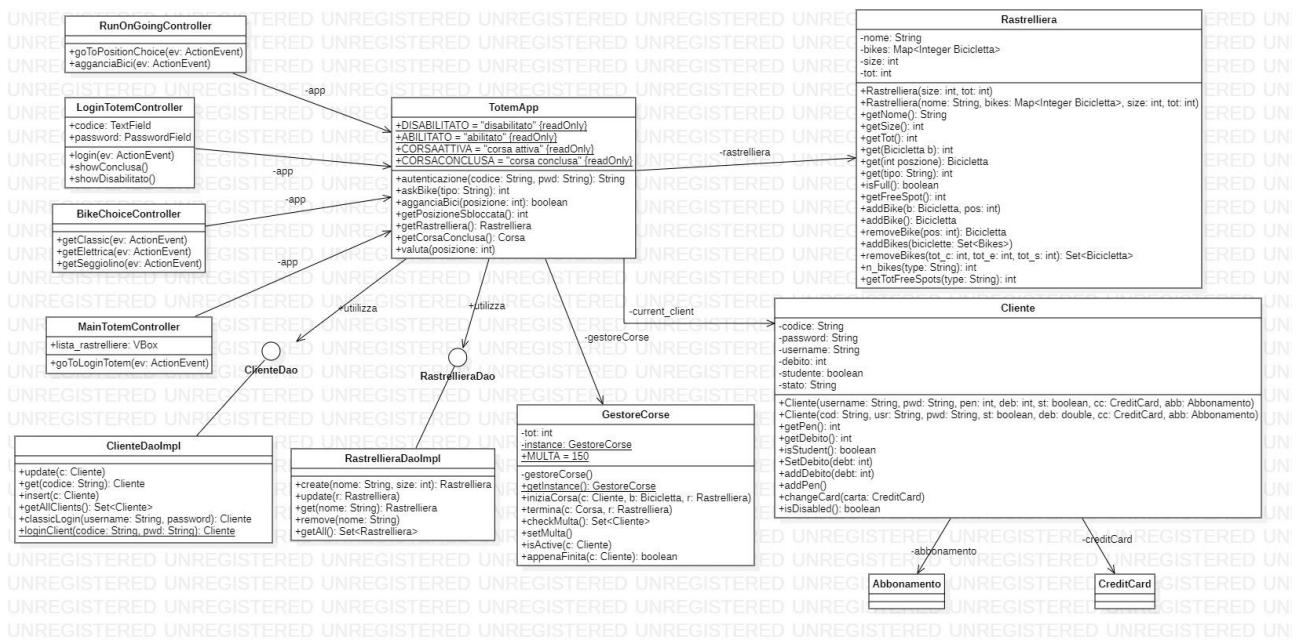
I tipi di bicicletta e tipi di abbonamenti come costanti (stringhe) sono definiti nelle rispettive Factory. Il motivo di ciò è perché i metodi Factory prendono come argomento proprio i tipi per creare gli oggetti ed è la classe Factory responsabile di sapere tutti i tipi possibili di implementazione delle rispettive classi astratte. Perciò è plausibile che sia la classe stessa Factory a definire i tipi.

Questo crea per esempio un’accoppiamento tra Rastrelliera e BiciclettaFactory, per tutti i metodi che prendono come argomento il tipo di bici.

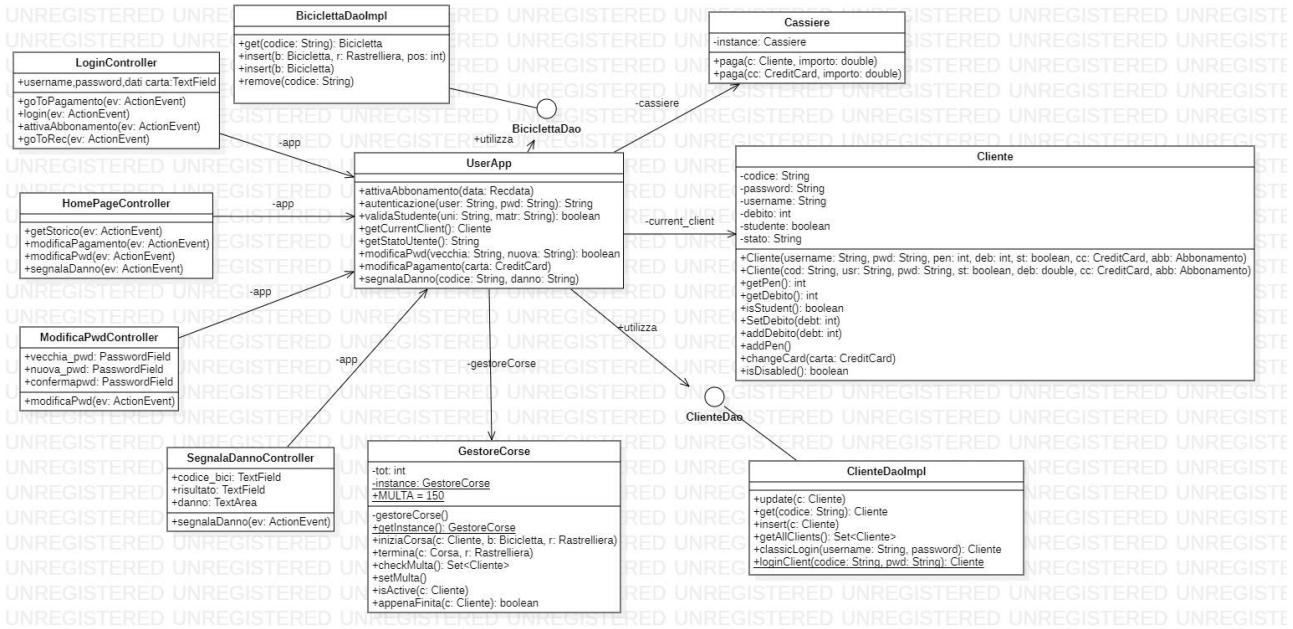
E' stata aggiunta anche una classe Cassiere che si interpone tra una classe che vuole utilizzare il ServizioPagamento e il ServizioPagamento stesso. In questo modo solo il Cassiere sa effettivamente come gestire il pagamento e tutte le altre classi così sono meno accoppiate con il ServizioPagamento che simula un servizio esterno che effettua in pratica le transazioni con le carte di credito.

I prossimi diagrammi sono visti dal punto di vista delle varie applicazioni (AppTotem, AppUser, AppPersonale), esse dialogano con i controller da una parte e dialogano e modificano il model dall'altra. Sono, in pratica un'interfaccia per i controller per comunicare con il model in modo da rendere più flessivo l'accoppiamento tra i controller dell'applicazione e l'applicazione stessa. In questi diagrammi sono solo mostrate le classi più utilizzate dalle varie App. Per alcuni metodi osservatori, i controller, per questioni di efficienza dialogano direttamente con il model senza passare dall'App. Soprattutto quando si vogliono mostrare una lista di informazioni provenienti dal database i controller dialogano direttamente con oggetti DAO.

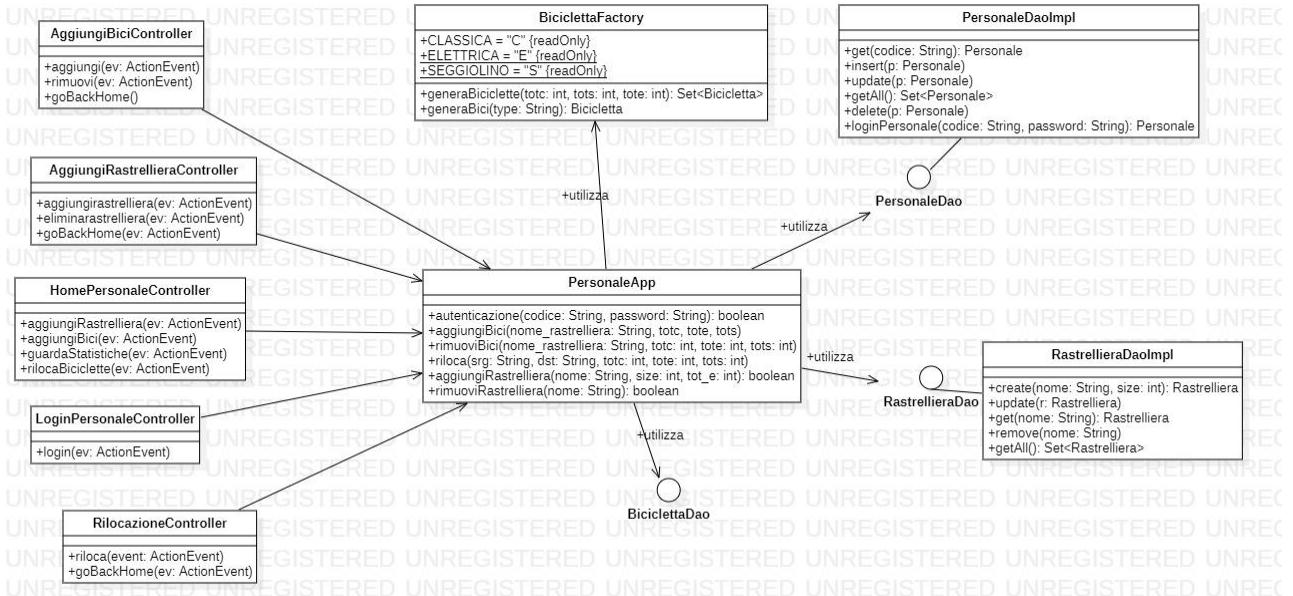
## DIAGRAMMA DELLE CLASSI (TOTEM)



## DIAGRAMMA DELLE CASSI (USER APP)



## DIAGRAMMA DELLE CLASSI (PERSONALE APP)



### 3.1.1 Discussione dei Design Pattern utilizzati

- Model view controller:** il pattern utilizzato per l'interazione con l'utente è il model view controller.

La view è gestita da delle pagine FXML che ho editato tramite SceneBuilder. Per ogni pagina FXML è associato uno e un solo controller.

I controller (associati ognuno a una vista) invece si occupano di gestire gli eventi della UI, catturare gli input inseriti dall'utente e fare delle chiamate ai metodi del model.

Le classi “App” (UserApp, PersonaleApp, TotemApp) sono degli oggetti di **Pure fabrication** per deresponsabilizzare i controller nel modificare il model e quindi per avere più low coupling. In questo modo queste classi fungono da interfaccia tra i controller e il model.

Per model si intendono quindi tutte le classi che servono per encapsulare i dati e fare le operazioni che osservano e modificano il model.

- Observer/Publish-Subscribe:** questo pattern viene utilizzato praticamente in automatico con l'uso di SceneBuilder. Infatti l'event source sono i componenti visuali creati da

SceneBuilder, mentre i Listener sono i controller sono gli event listener che scattano non appena succede l'evento al quale si sono "iscritti"

- **DAO:** per accedere e modificare i dati persistenti dell'applicazione viene utilizzato il Data Access Object pattern. Questo pattern garantisce un lasco accoppiamento tra la vista e la modifica degli oggetti estratti dal database e la gestione effettiva dei dati persistenti. Infatti, per esempio, la Rastrelliera, come oggetto, contiene le informazioni su tutte le biciclette agganciate, tipi di morse ecc. Nel database però, per questioni di efficienza, sono i record della tabella bicicletta che contengono le informazioni della posizione e rastrelliera alla quale la bici è agganciata. Nella tabella delle rastrelliere sono solo contenute le informazioni sulla dimensione della rastrelliera e sul numero di morse elettriche presenti. L'oggetto RastrellieraDao perciò provvede a creare, modificare le informazioni relative a un oggetto Rastrelliera performando operazioni su due tabelle effettive del database (tabella delle biciclette e tabella delle rastrelliere).

Tutte le classi che perciò vogliono ottenere o modificare informazioni dal database dovranno fare una semplice chiamata all'oggetto DAO ignorando totalmente la struttura interna del database. Inoltre anche se dovesse cambiare la struttura interna del database il model reggerebbe comunque, infatti basterebbe cambiare l'implementazione dell'interfaccia DAO alla quale il model si appoggia

- **Singleton:** la classe GestoreCorse e la classe Cassiere sono implementate seguendo il pattern Singleton in quanto non è necessaria più di una istanza di ognuna delle due classi.
- **Indirection:** la classe Cassiere implementa un pattern di indirezione in quanto serve per separare il model dall'effettiva gestione dei pagamenti. Il model si limita a chiamare il metodo paga() dell'oggetto Cassiere che risponde con un valore booleano per dire che la transazione è andata a buon fine oppure no.

In questo modo si cerca di rendere più lasco l'accoppiamento con l'implementazione del pagamento tramite il "servizio esterno" ServizioPagamento e se esso cambiasse struttura si dovrebbe cambiare solo l'implementazione della classe Cassiere e il model continuerebbe a funzionare.

- **Factory method:** AbbonamentoFactory e BiciclettaFactory implementano il design pattern del Factory method, infatti sia Abbonamento che Bicicletta sono due interfacce che sono implementate ognuna da 3 classi concrete :

(Abbonamento: AbbonamentoAnnuale, AbbonamentoSettimanale, AbbonamentoAnnuale)

(Bicicletta: BiciclettaClassica, BiciElettrica, BiciConSeggiolino)

Queste classi di Pure Fabrication permettono di non doversi preoccupare di specificare l'implementazione concreta dell'interfaccia quando si vuole generare un'istanza di Bicicletta o di Abbonamento, ma semplicemente di specificare il tipo nel metodo generatore della Factory.

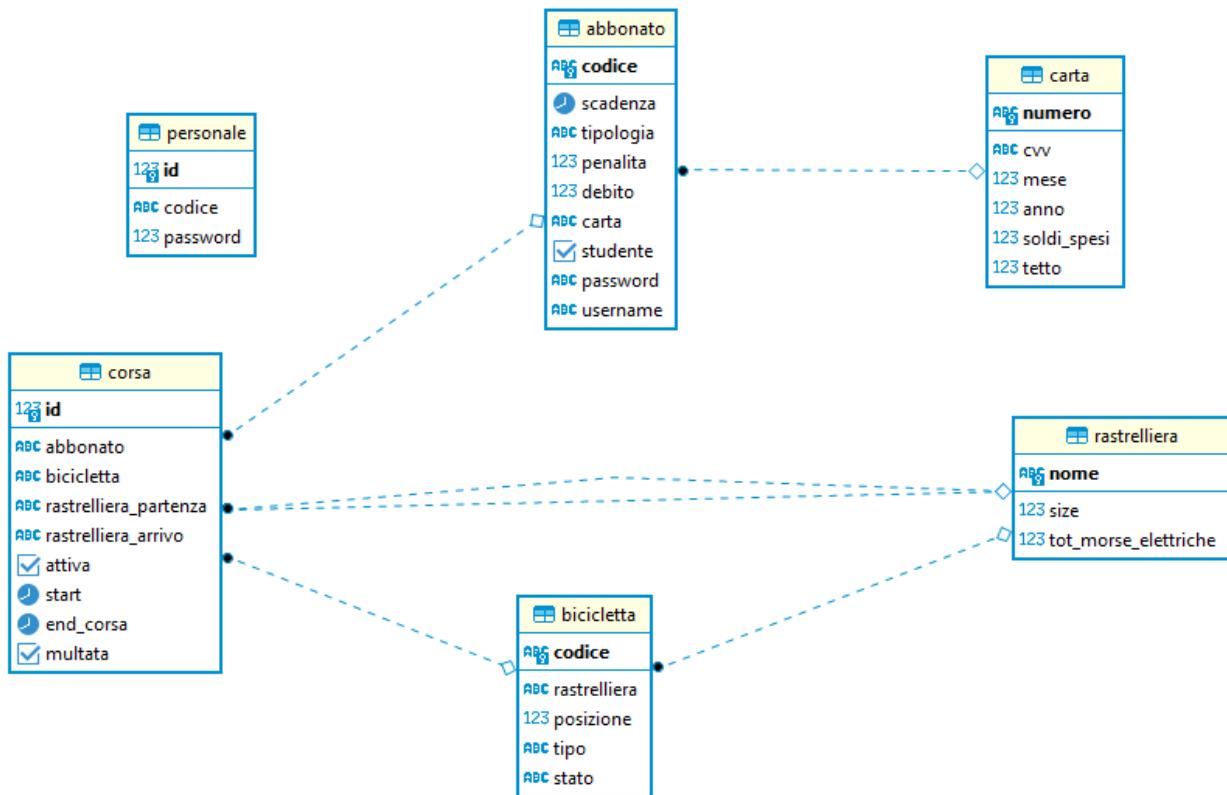
AbbonamentoFactory e BiciclettaFactory contengono tre attributi finali e statici che specificano il nome dato ai tipi di classi concrete, che verrà usato per applicare i metodi factory.

Per BiciclettaFactory: CLASSICA = "C", ELETTRICA = "E" e SEGGIOLINO "S".

Per AbbonamentoFactory:

BiciclettaFactory, inoltre, oltre a generare una Bicicletta di un dato tipo dato il codice della bicicletta e il tipo, permette anche di generare una bicicletta nuova dato solamente il tipo. Per essere certo dell'univocità del codice scelto per la bicicletta nuova essa si interfaccia con BiciclettaDao. BiciclettaFactory contiene anche un metodo per generare insieme di biciclette dato il numero di biciclette classiche, il numero di biciclette elettriche e il numero di biciclette con seggiolino.

### 3.2 Gestione dei dati persistenti



Ogni corsa ha un id come chiave primaria e ha 4 chiavi esterne, 2 per la rastrelliera di partenza e per la rastrelliera di arrivo, per l'abbonato che fa la corsa e per la bicicletta usata nella corsa.

Nella tabella dell'abbonato invece sono contenute tutte le informazioni sul cliente

(codice -> chiave primaria ,username -> chiave univoca,password,penalita,debito) e tutte le informazioni sull'abbonamento(tipologia, inizio e fine). Contiene inoltre una chiave esterna(carta) che punta alla tabella delle carte dove oltre alle informazioni base sulla carta di credito(numero,cvv,mese e anno) contiene la colonna "soldi\_spesi" dove sono aggiunti soldi sulla carta ad ogni transazione e la colonna "tetto" dove è specificato il tetto massimo.

Nella tabella "bicicletta" oltre alle informazioni sulla bicicletta (codice -> chiave primaria, tipo e stato) è anche specificata la rastrelliera eventuale alla quale è aggangiata la bicicletta (chiave esterna alla tabella rastrelliera) e in quale posizione.

Nella tabella rastrelliera invece ci sono solo tre colonne e specificano il nome della rastrelliera (chiave primaria), la dimensione(size) e il numero di morse elettriche. Con queste informazioni, infatti si sa la dimensione della rastrelliera, quante morse classiche e quante morse elettriche sono presenti nella rastrelliera.

Le password nella tabella abbonato sono in chiaro solo per una questione di testing, in uno scenario reale verrebbero salvate tramite funzioni di hash.

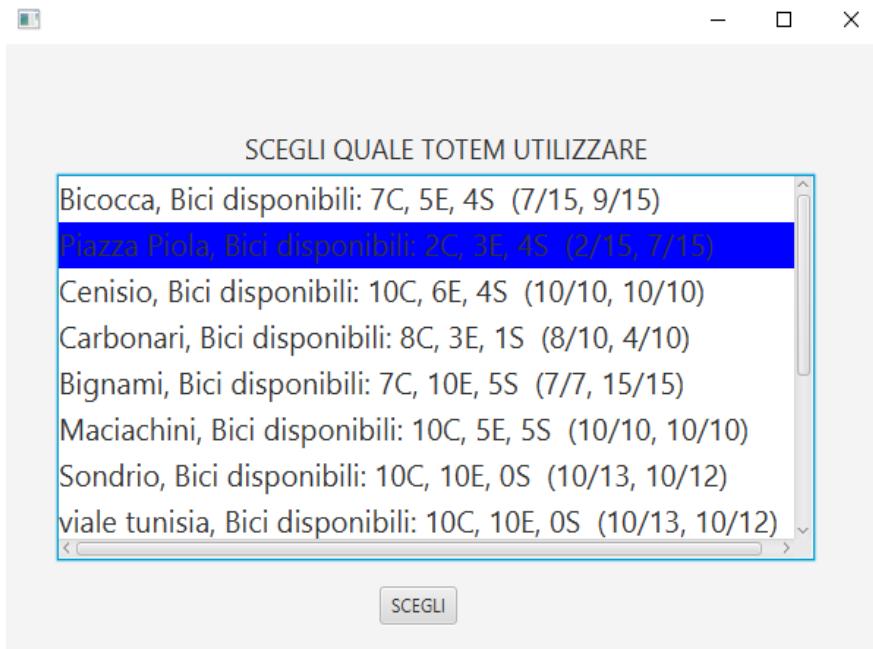
### 3.3 Descrizione dell'Interfaccia Grafica

Sono mostrate le interfacce grafiche dell'applicazione per gli utenti, per il personale del servizio e l'applicazione dei totem.

Per fare in modo che tutti i controller comunichino con i rispettivi oggetti "App" ho impostato UserData dello stage e ho messo lì dentro l'oggetto App. In questo modo tutti i controller, accedendo allo stage in modo statico, possono dialogare con l'App rispettiva.

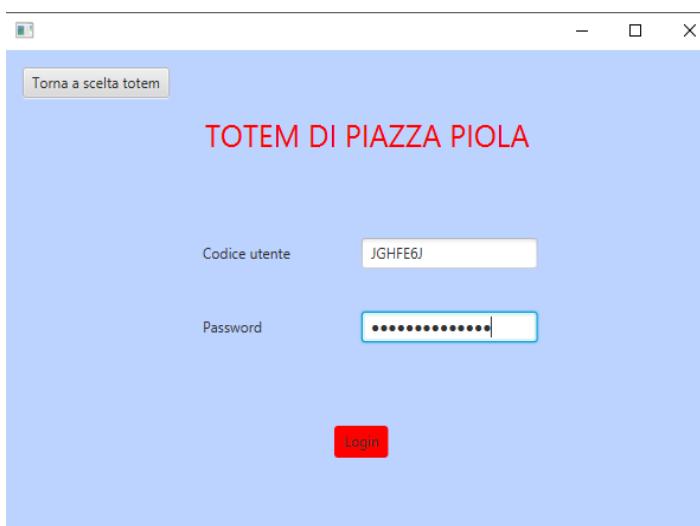
## INTERFACCIA TOTEM

### 1. SCELTA TOTEM



Vengono mostrate le possibili rastrelliere in cui si vuole andare. Per scegliere una rastrelliera basta selezionare la riga che la descrive e cliccare scegli. Ogni riga contiene: nome rastrelliera, numero di biciclette classiche,elettriche e con seggiolino disponibili. Tra parentesi sono indicate il numero di morse classiche libere sul totale di morse classiche e il numero di morse elettriche libere sul totale di morse elettriche.

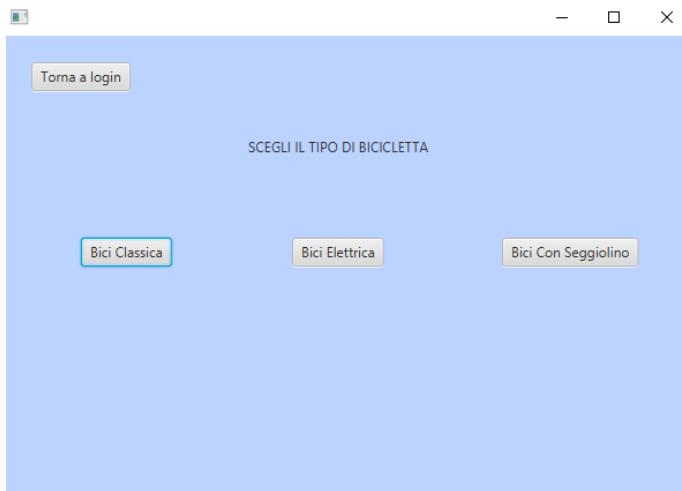
### 2. AUTENTICAZIONE



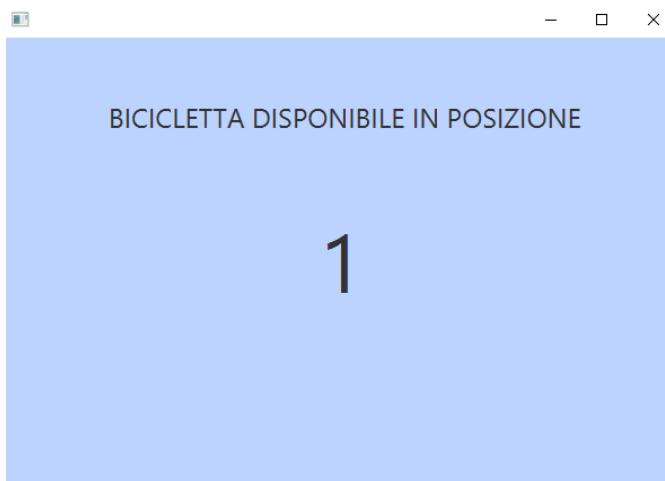
Per autenticarsi basta digitare codice utente e password nei textfield corrispondenti e cliccare su login. Se lo username o la password non sono corretti verrà visualizzato un messaggio di errore sotto il tasto di login.

Una volta autenticati possono mostrarsi 3 scenari diversi (a,b,c,d).

### 3. SCELTA TIPO DI BICI (scenario a)



Per selezionare un tipo di bici basta cliccare il tasto corrispondente a quel tipo.  
Se non c'è quel tipo di bici disponibile viene mostrato un messaggio di errore nella parte inferiore dello schermo.



Se la bicicletta è disponibile viene mostrato la posizione in cui la bicicletta è stata sganciata dalla morsa.

Per mostrare la posizione non è stata creata un'altra pagina fxml, ma semplicemente viene messo visibile un Pane che sostituisce quello di scelta del tipo di bici che viene quindi reso invisibile.

Dopo 5 secondi si viene reindirizzati automaticamente alla pagina di login.

#### 4. SCELTA MORSA ALLA QUALE AGGANCIARE LA BICICLETTA (scenario b)

Questa schermata appare nel caso che l'utente ha già una corsa in corso.

E' stata messa per motivi di simulazione in quanto nella realtà l'utente aggancerebbe la bicicletta fisicamente.



Vengono mostrate le morsa presenti nella rastrelliera e il loro stato (se non sono occupate viene mostrato se sono classiche o elettriche).

Selezionando una morsa essa viene colorata di blu e cliccando la tasto 'Aggiorna bicicletta su questa rastrelliera' la bicicletta viene agganciata, la corsa dell'utente termina e si viene reindirizzati alla pagina di login.

Nel caso che un utente selezioni una morsa sbagliata per il tipo di bici che ha, il tasto Aggiorna si disattiva e viene mostrato un messaggio di errore.

#### 5. CORSA TERMINATA (scenario c)

Questo schermata si presenta se l'utente ha finito una corsa da meno di 5 minuti.



Questa schermata rimane attiva per 5 secondi, si viene poi reindirizzati alla pagina di login. Anche in questo caso non c'è una pagina fxml dedicata, ma solo un pane che si sovrappone a quello precedente.

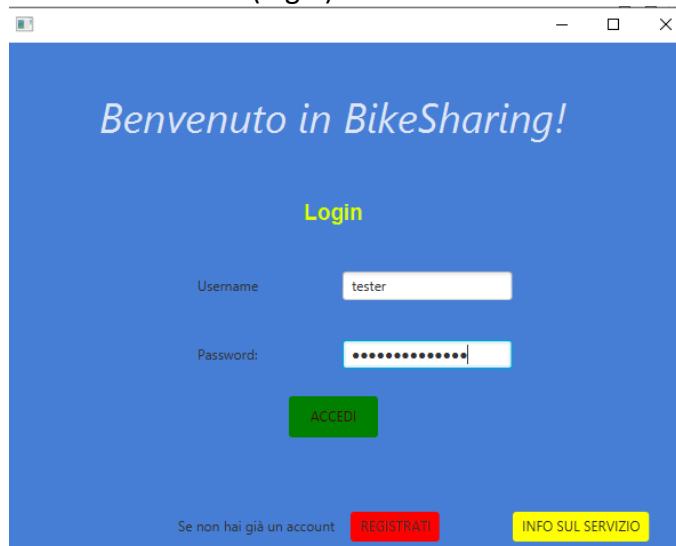
## 6. UTENTE NON ABILITATO (scenario d)



Anche in questo caso non c'è una pagina fxml dedicata, ma solo un pane che si sovrappone a quello precedente.

## INTERFACCIA APPLICAZIONE PER UTENTI

Interfaccia iniziale (login)



Viene richiesto username e password, se sono corretti e si clicca su ACCEDI si viene portati sul menu principale.

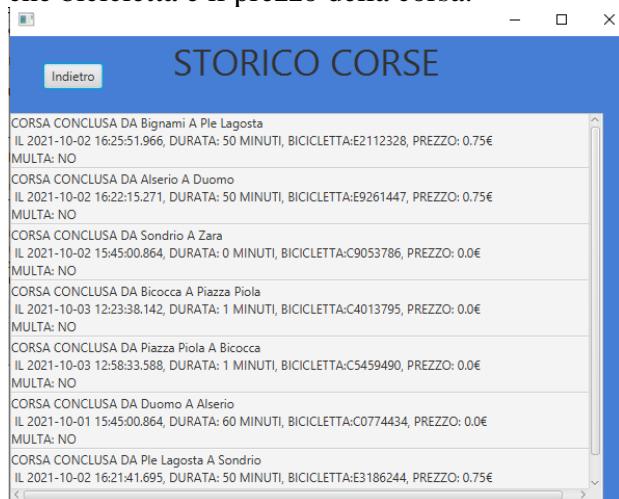
Ci sono anche altri due tasti: “Registrati” che porta all’interfaccia per la registrazione al servizio bikesharing e “Info sul servizio” che porta a una schermata che specifica delle informazioni sul servizio di bikesharing.

## Menu Principale



Dal menu principale dell’utente ci sono 4 funzionalità offerte all’utente.

- Storico corse: viene specificato per ogni corsa: da dove a dove, timestamp di inizio, durata, che bicicletta e il prezzo della corsa.



- Segnala danno a bicicletta: viene data la possibilità all’utente di segnalare un danno ad una bicicletta. Per segnalare il danno alla bicicletta bisogna selezionare il codice di essa tramite una choicebox che mostra le biciclette utilizzate nelle 3 corse passate. Se il codice non esiste viene dato un messaggio di errore, sennò appare un messaggio di conferma segnalazione.



- Modifica password: viene semplicemente richiesta la vecchia password, la nuova password e una conferma della nuova password.
- Modifica metodo di pagamento: vengono richiesti i dati della carta che, se validi (la validità della carta viene specificata nella registrazione).

### Registrazione utente:

**Registrazione**

Torna a LOGIN

Tipi di abbonamento:

GIORNALIERO(6\$)

SETTIMANALE(20\$)

ANNUALE(36\$)

Username: tester\_di\_prova

Sei uno studente?  Si  No

UNIVERSITÀ: Bicocca

Numero di matricola: 123456

Password: \*\*\*\*\*

Conferma password: \*\*\*\*\*

PROCEDI

### Validità degli input:

- Lo username non può essere vuoto e non deve essere già utilizzato
- La password deve essere uguale o superiore ai 6 caratteri
- Il numero di matricola deve essere uguale a 6 cifre
- L'università non può essere nulla

Se uno di questi vincoli non viene rispettato viene mostrato un messaggio di errore nella parte inferiore dello schermo.

Se gli input sono validi e si clicca su procedi si viene portati alla pagina di registrazione pagamento in cui vengono richiesti i dati della carta di credito.

Indietro

INSERIMENTO DATI CARTA

Ti stai per iscrivere all'abbonamento annuale, procedendo con il pagamento ti verranno addebitati 36\$ e potrai da subito utilizzare il servizio bikesharing con il codice fornito e la password da te scelta

Numero carta: 3880123439485977

Genera

CVV: 122

Scadenza: 11 / 22

ATTIVA ABBONAMENTO BIKESharing

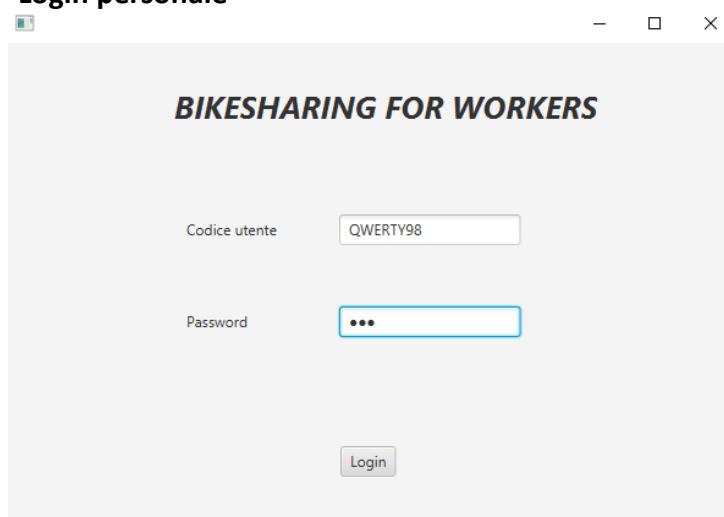
Per comodità( dato che siamo in un ambiente di simulazione) ho dotato questa interfaccia di un tasto Genera che genera un numero di carta di credito valido (16 cifre). Il cvv e la scadenza vanno invece per forza inseriti a mano. Il cvv deve essere per forza di 3 cifre.

La carta inoltre nel caso di abbonamento annuale non può scadere tra meno di un anno, e se l'abbonamento è settimanale o giornaliero non potrà scadere tra meno di un mese. Nel caso che gli input non siano validi viene mostrato un messaggio di errore, se invece sono validi e si clicca su Attiva Abbonamento si viene portati su una pagina di successo registrazione che mostrerà il codice univoco assegnato all'utente.

## INTERFACCIA PER PERSONALE DEL SERVIZIO

Per il personale del servizio le credenziali devono essere conosciute a priori, non esiste un portale di registrazione.

### -Login personale



User: QWERTY98 e Password:123 è un utente realmente esistente nel database del personale del servizio.

Se le credenziali sono valide si viene portati sul menu del personale del servizio.

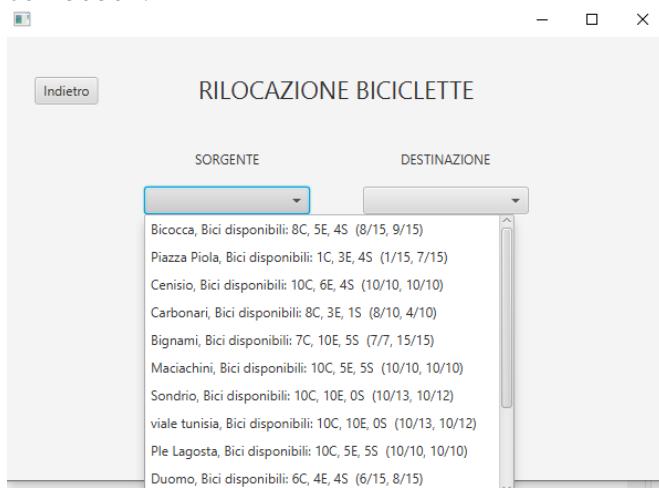


Ci sono 5 funzionalità fornite al personale del servizio:

- **Rilocazione Biciclette**



Viene scelta una rastrelliera sorgente e una rastrelliera destinazione tramite due combobox.



Per ogni rastrelliera vengono specificate il numero di biciclette presenti per ogni tipo e tra parentesi (morse classiche libere/totale morse classiche, morse elettriche libere / totale morse elettriche). In questo modo l'utente del personale è più facilitato nella scelta delle rastrelliere.

Se il numero di biciclette è eccessivo oppure se la rastrelliera di destinazione non contiene abbastanza morse libere viene mostrato un messaggio di errore.

In caso di buon fine invece viene mostrato un messaggio di successo e le combobox vengono aggiornate.

Indietro RILOCAZIONE BICICLETTE

SORGENTE DESTINAZIONE

classiche elettriche seggi

Numero biciclette: 6 4 3

RILoca

Rilocazione andata a buon fine

- **Aggiunta/rimozione biciclette:** con una combobox viene scelta la rastrelliera da cui si vuole aggiungere o rimuovere delle biciclette e viene specificato poi il numero di biciclette classiche, elettriche, e con seggiolino che si vogliono aggiungere/rimuovere.

Indietro AGGIUNGI/RIMUOVI BICICLETTE

AGGIUNGI

Classiche: 10 Elettriche: 10 Seggi: 5 Rastrelliera: Bicocca, Bici dispon...

AGGIUNGI

RIMUOVI

Classiche: 10 Elettriche: 10 Seggi: 5 Rastrelliera:

RIMUOVI

- **Creazione/rimozione rastrelliera:** viene richiesto il nome, la dimensione e il numero di morse elettriche per creare una rastrelliera. Sia la dimensione, che il numero di morse elettriche viene scelto tramite una combobox. La combobox del numero di morse elettriche viene popolata solo dopo aver selezionato la dimensione della rastrelliera.

Indietro AGGIUNGI NUOVA RASTRELLIERA

Nome: Rastrelliera di prova Dimensione: 15 Totale morse elettriche 8

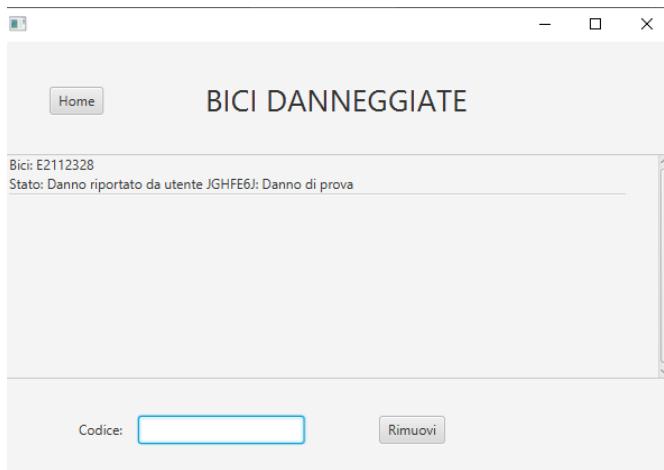
AGGIUNGI

ELIMINA RASTRELLIERA

Bicocca ELIMINA

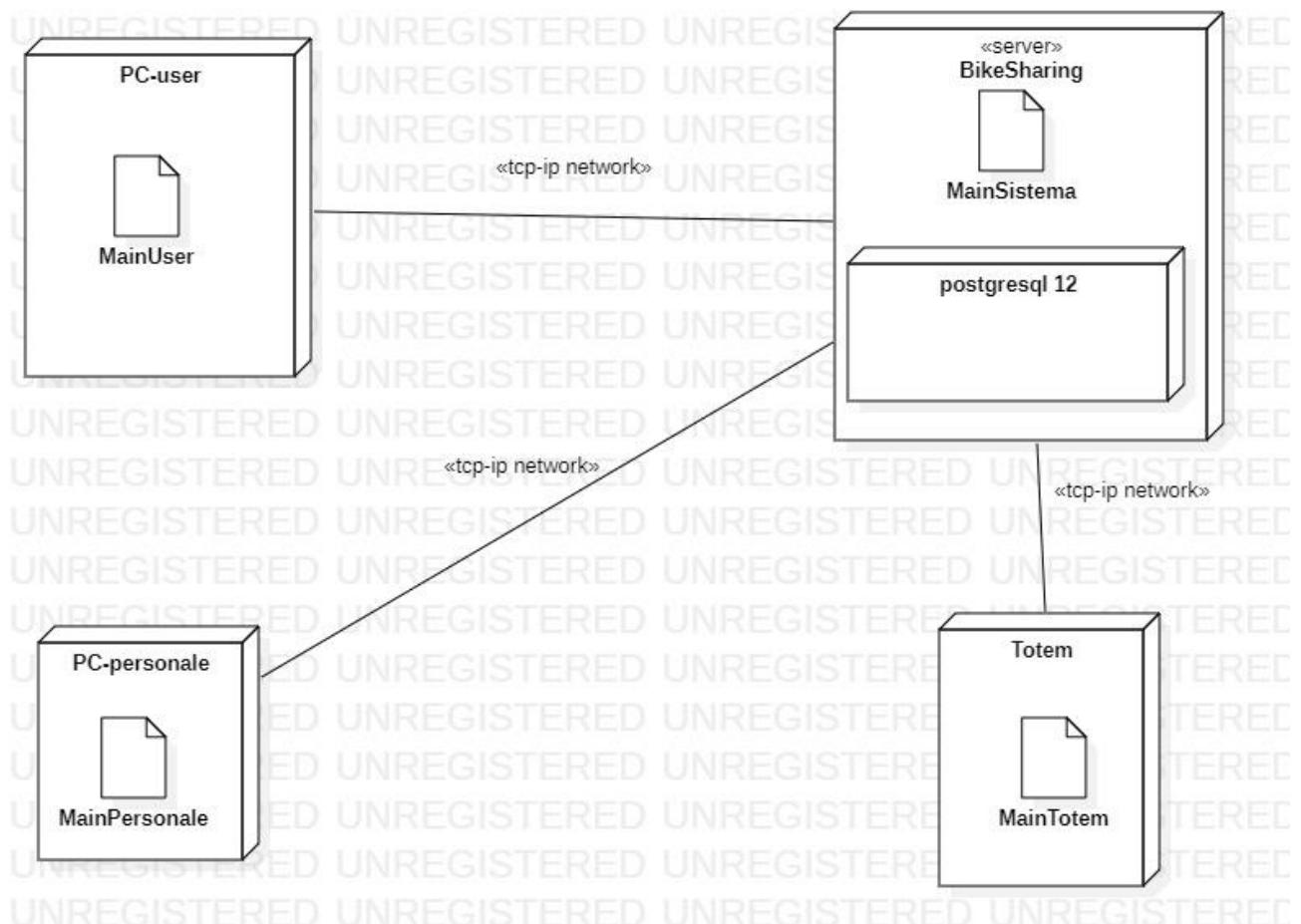
Validazione input: la rastrelliera deve essere almeno di 5 caratteri

- **Stato biciclette:** in questa schermata viene mostrato lo stato delle biciclette che è diverso da “OK”. Viene data quindi la possibilità di rimuovere una bicicletta dato il codice. Se la bicicletta esiste ed è agganciata a una rastrelliera (non è associata ad una corsa) viene rimossa e viene mostrato un messaggio di avvenuta rimozione.



- **Statistiche sul servizio:** vengono visualizzate alcune statistiche tra cui il numero di abbonati, percentuale di studenti abbonati, numero medio di biciclette usate, tipo di bicicletta più usato.

### 3.4 Diagramma di deployment



I 4 file (MainTotem, MainUser, MainPersonale, MainSistema) sono i 4 main che vengono runnati sulle varie macchine.

Tutti i componenti fanno riferimento al server, per ottenere e modificare i dati, dove è presente un database postgresql.

### 3.5 Specifica e verifica dei vincoli

Vincoli OCL delle principali classi di BikeSharing: Corsa, Abbonamento, Rastrelliera, Cliente

//invariante della classe corsa : l'inizio (timestamp) della corsa è sempre impostato e se la fine (timestamp) è impostata allora è sicuramente minore dell'inizio

context Corsa{

    inv: start <> null

    inv: end <> null implies self.start.getTime() < self.end.getTime()

}

//metodo della corsa che calcola il minutaggio della corsa. Se la fine della corsa non è impostata //calcola il tempo dall'inizio a ora

Context Corsa::calcola():integer{

    Self.end == null implies result = DateTime.now().nrOfMinutes – start.nrOfMinutes

    Self.end <> null implies result = end.nrOfMinutes – start.nrOfMinutes

}

//metodo di corsa che calcola il prezzo della corsa. Viene impostato lo sconto studenti nel caso che //il cliente della corsa sia uno studente

Context Corsa::getPrice():double{

    result = bicicletta.getPrezzo(calcola())

    cliente.isStudent() AND BiciclettaFactory.getType(bicicletta).equals("CLASSICA") implies  
    result = 0

}

//Nel caso che l'abbonamento sia annuale l'inizio e la fine devono per forza essere impostati

Context AbbonamentoAnnuale{

    Inv: start <> null AND end <> null

    Inv: start = end.minusDays(365)

}

//inizializza abbonamento settimanale

Context AbbonamentoSettimanale::Start(){

    Pre: start = null and end = null

    Post: start = DateTime.now()

    Post: end = start.plusDays(7)

}

//inizializza abbonamento giornaliero

Context AbbonamentoGiornaliero::Start(){

```

Pre: start = null and end = null
Post: start = DateTime.now()
Post: end = start.plusDays(1)
}
//invariante Rastrelliera: la dimensione della mappa <Integer,Bicicletta> (bikes) deve essere
//uguale all'attributo size
Context Rastrelliera{
    Inv: Bikes.size() == size;
}

//rimozione di una bicicletta data la sua posizione. Il totale di biciclette viene diminuito di 1
Context Rastrelliera::removeBike(posizione:integer):Bicicletta{
    Post:Bikes.get(posizione) <> null implies result = bikes.get(posizione)
    Post: self.getTot() = self.getTot@pre() - 1
}

//invariante della classe Cliente
Context Cliente{
    Inv: penalita <= 3
    Inv: codice.size () == 7
    Inv: password.size() >= 6
    Inv: username <> null
    Inv: abbonamento <> null
    Inv: carta <> null
}
//setta il debito di un Cliente
Context Cliente :: setDebito(importo:double){
    Post: debito = importo
}
//aggiunge un importo al debito di un Cliente
Context Cliente:: addDebito(importo:double){
    Post: debito = debito@pre + 1
}
//aggiunge una penalità a un Cliente
Context Cliente:: addPen(){
    Penalita < 3 implies Penalita = penalita@pre + 1
}

// il codice di una bici deve essere per forza di 8 caratteri
Context Bicicletta{
    Inv: codice.size() == 8
}

```

Alcuni di questi vincoli sono stati mappati in JML nelle rispettive classi.  
Nei commenti delle varie classi si possono trovare inoltre informazioni aggiuntive quali le invarianti di rappresentazioni e pre e post condizioni di alcuni metodi.

### 3.6 Descrizione del testing

Per il testing dell'applicazione ho utilizzato le librerie di Junit 5.

Sono presenti due package di testing: un package dove ho testato le classi principali del model e un test dove ho testato proprio le funzioni delle applicazioni finali per users, personale servizio e totem.

Generando esse dati sul database ho creato, tramite test, dati fittizzi che alla fine di essi ho distrutto.

I test cercando di coprire tutte le funzionalità principali del sistema. Non vengono testati a fondo i DAO e neanche i controller.

### 3.7 Note per l'installazione e l'utilizzo

Come ambiente di sviluppo per creare quest'applicazione ho utilizzato eclipse, allego perciò l'intero progetto che dovrà essere importato su eclipse per poterlo testare.

Per compilare il codice ho utilizzato JDK 11, mentre per il database ho utilizzato postgresql 12, pertanto bisognerà avere installato il dbms in locale.

Oltre la relazione e ai sorgenti allego quindi anche un dump del database.

Per eseguire un dump del database funzionante:

- Va aggiunto un nuovo ruolo: **marco\_hazan** (in grado almeno di fare login)
- Con Password utente: **bikesharing**
- Va aggiunto quindi un database: **marcohazan\_db con owner marco\_hazan**

Questi parametri sono presenti comunque nel sorgente UtilFunc.java che permette di connettersi al database.

Tramite linea di comando con utility PSQL per dumper il database bisognerà digitare:

psql -U marco\_hazan -h localhost marcohazan\_db < "dump.sql". Dovrà quindi crearsi lo schema bikesharing all'interno del database marcohazan\_db.

Sono già presenti dei dati (rastrelliere già riempite di un po' di biciclette) nel database in modo da poter utilizzare da subito il programma.

Clienti tester:

- Codice utente: **ZOOE7QG**, Username: **luca**, password: **luca98**, tipologia abbonamento:**settimanale**, studente: **NO**
- Codice utente: **7HP18R7**, Username: **emanuele**, password: **emanuele**, tipologia abbonamento:**settimanale**, studente: **SI**
- Codice utente: **FS3UPMJ**, Username: **mariano**, password: **mariano**, tipologia abbonamento:**giornaliero**, studente: **NO**
- Codice utente: **O6GYOEV**, Username: **laura**, password: **laura11**, tipologia abbonamento:**giornaliero**, studente: **SI**
- Codice utente: **JGHFE6J**, Username: **tester**, password: **testerpassword**, tipologia abbonamento:**annuale**, studente: **SI**
- Codice utente: **XL7D2OD**, Username: **paolo**, password: **password**, tipologia abbonamento:**annuale**, studente: **NO**

Ci sono 6 tester uno per ogni combinazione di abbonamento (tipo + studente).

Sulle carte dei tester sono addebitati solo i costi degli abbonamenti.

I clienti con abbonamento giornaliero e settimanale hanno gli abbonamenti non ancora cominciati, così che verrà settata la scadenza al loro primo prelievo.

Personale Servizio Tester:

Codice utente: **QWERTY98** Password:**123**

Per la UI ho utilizzato JavaFX (le librerie di javaFX sono già contenute all'interno del progetto).

Dentro il package main sono contenuti i 4 main delle 4 applicazioni diverse:

- MainPersonale fa partire l'applicazione per il personale del servizio
- MainTotem fa partire l'applicazione per i totem
- MainUser fa partire l'applicazione per gli utenti
- MainSistema (che dovrebbe andare in background) fa partire l'applicazione sistema per il settaggio delle multe

Per fare in modo che i main delle applicazioni grafiche funzionino

(MainPersonale,MainUser,MainTotem) bisogna mettere come VM arguments in Run

Configurations: --module-path "libs\lib" --add-modules javafx.controls,javafx.fxml.

MainSistema invece è un main molto semplice e stampa solo su linea di comando.