RELAZIONE PROGETTO INGEGNERIA DEL SOFTWARE

ANDREA UBBIALI 923457
A.A. 2020/21

1 Descrizione del problema

1.1 Analisi e specifica dei requisiti

Il progetto in relazione tratta di un sistema di bike sharing.
Il sistema per essere usufruito da qualsiasi utente richiede un'autenticazione e, precedentemente, una registrazione. È stato scelto di registrare gli utenti in modo tale da poter mantenere in modo persistente i loro dati e permetter loro di sottoscrivere più abbonamenti evitando la registrazione ogni qualvolta.
All'atto della registrazione sono richiesti dei dati all'utente: nome, cognome, email e password. Quest'ultime (email e password) saranno le credenziali utilizzate dall'utente per attuare qualsiasi azione nel sistema essendo unici e quindi identificativi.
L'utente all'atto della registrazione o nella sua area privata potrà dar prova del suo stato di studente. Qualora ciò avvenisse il sistema effettuerà una richiesta al servizio preposto (esterno al sistema) che validerà o meno tale richiesta.
Come citato in precedenza l'utente autenticato ha a disposizione un'area riservata dove ha la possibilità di aggiungere delle carte di credito e sottoscrivere degli abbonamenti utilizzando sempre lo stesso account.
Il sistema dispone di 3 tipi di abbonamenti: giornaliero, settimanale ed annuale. L'abbonamento annuale inizierà contestualmente alla data di sottoscrizione dell'abbonamento mentre, per gli abbonamenti giornalieri e settimanali, la data di inizio sarà al primo prelievo della bicicletta.
A causa di questa differenza di inizio dell'abbonamento è necessario dare un limite entro il quale usufruire del servizio. Il motivo di tale scelta è dettato dal fatto che le carte di credito (unico metodo possibile di pagamento) devono essere valide per tutta la durata dell'abbonamento ed è quindi necessario porre un limite.
Di conseguenza gli abbonamenti giornalieri e settimanali potranno partire entro 90 giorni dalla data di sottoscrizione dell'abbonamento, pena la non possibilità di usufruzione.
Il costo degli abbonamenti è detratto alla sottoscrizione dalla propria carta di credito. Di seguito il tariffario:

- **Giornaliero**: 4.50€
- **Settimanale**: 9€
- **Annuale**: 36€

Il sistema mette a disposizione 3 tipi di biciclette, di seguito vengono presentate anche con il loro rispettivo tariffario:

- **normale**: gratuito per i primi 30 minuti, 0.50€ ogni 30 minuti fino alle 2 ore; Dalle 2 ore in poi 2.00€ ogni ora.

- **elettrica**: gratuito per i primi 3 minuti, 0.25€ per i restanti 27 minuti. Successivamente la tariffa per ogni mezz'ora fino alle 2 ore è rispettivamente 0.50€, 0.50€, 1.00€, 2.00€. Dalle due ore in poi 4.00€ all'ora.
- **elettrica con seggiolino**: tariffario uguale alla bicicletta elettrica.

L'utente registrato e accertato come studente potrà usufruire, diversamente dagli altri utenti, dell'utilizzo gratuito, per tutta la durata del noleggio, della bicicletta di tipo normale.

Il sistema è dotato di totem ai quali sono collegate fisicamente delle rastrelliere dove verranno inserite le biciclette. Ogni morsa può ricevere solo un tipo specifico di bicicletta.
L'utente per iniziare il noleggio di una bicicletta deve inserire nel totem la propria email, la password e scegliere la tipologia di bicicletta; il sistema risponderà con la postazione della morsa aperta dove si trova la bicicletta riservata all'utente.
Per quanto riguarda la restituzione di una bicicletta l'utente dovrà porre la bicicletta in una morsa libera, inserire nel sistema l'email e la password ed il totem provvederà a chiudere la morsa qualora il tipo di bici sia quello accettato.
Contestualmente alla restituzione della bicicletta l'utente potrà comunicare un eventuale danno della bicicletta. Nel caso in cui ciò avvenga, la bicicletta verrà posta automaticamente in manutenzione e non sarà più quindi possibile noleggiarla.

All'interno del sistema esiste inoltre un utente admin con delle funzionalità specifiche per la gestione del servizio. L'utente registrato come admin ha la possibilità di aggiungere nuove rastrelliere, eliminare quelle dismesse, aggiungere delle biciclette oppure eliminarne.
Inoltre tale utente può sistemare le biciclette/morse e resettare lo stato all'interno del servizio. Tutto ciò può avvenire nel pannello di amministrazione previa autenticazione come utente admin.

Il sistema prevede inoltre:
- l'impossibilità di noleggiare una bicicletta se non sono passati 5 minuti dall'ultimo utilizzo
- limite di noleggio di 2 ore, qualora si superasse per 3 volte con lo stesso abbonamento esso verrà annullato.
- Addebito di penale di 150€ più la tariffa di utilizzo per la mancata riconsegna della bicicletta entro le 24 ore.

Il sistema mette inotre a disposizione dei dati statistici:
- numero di abbonamenti attivi;
- numero di noleggi attivi;
- numero di utenti;
- media di utilizzo per noleggio;
- tipologia di bicicletta maggiormente utilizzata.

Ogni qualvolta ci si debba rivolgere all'esterno del sistema, per esempio con l'istituto di credito, rastrelliera(hardware), sistema universitario… nel codice le

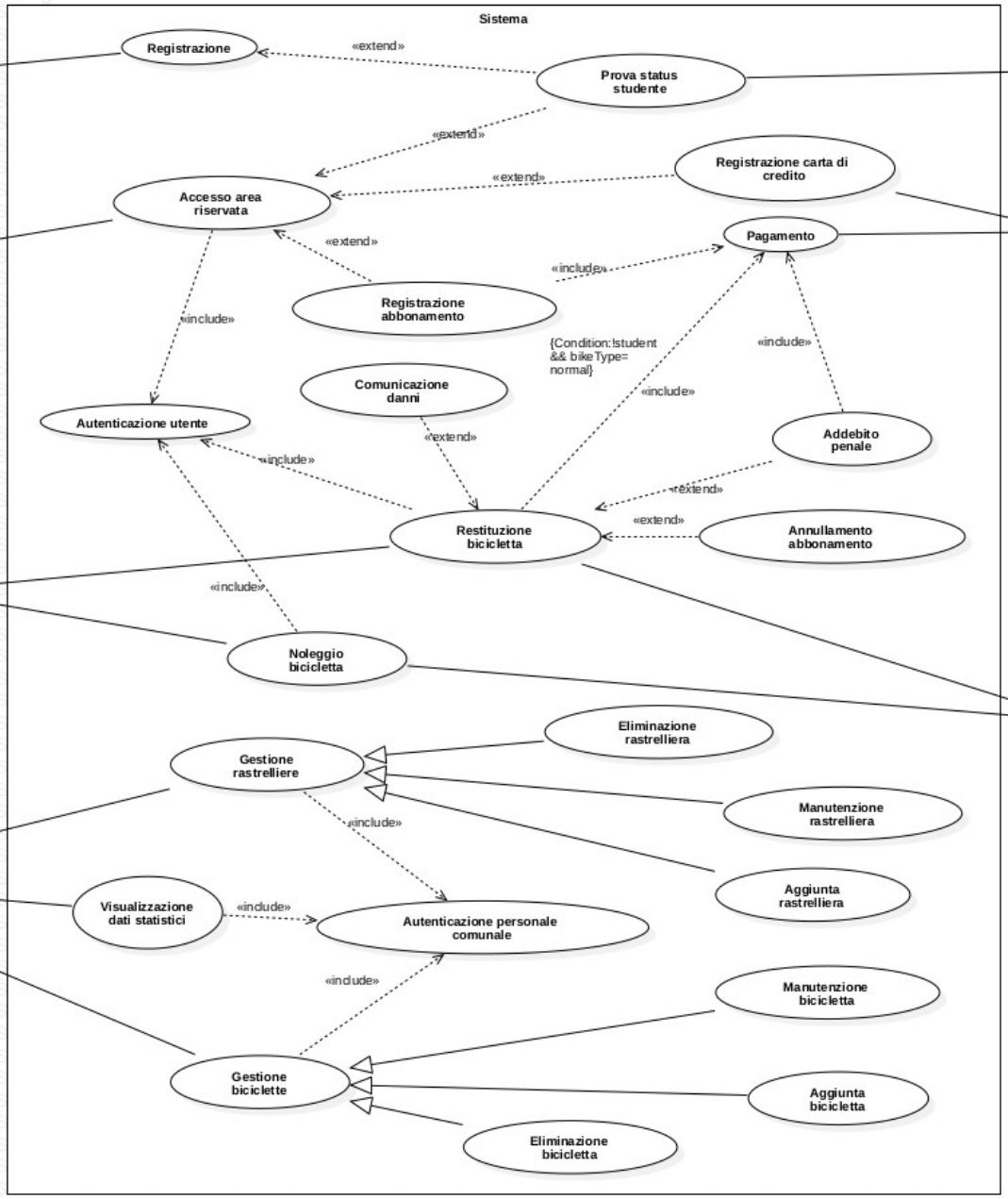
risposte di tali attori esterni sono state definite nel file di configurazione (vedere note per l'installazione e utilizzo per modificare le risposte).

1.2 Glossario

Definisce termini e acronimi utilizzati nel documento.
Sistema = software di bike sharing

2 Progettazione del Sistema

2.1 Diagramma dei casi d'uso



2.1.1 Descrizione testuale dei casi d'uso

REGISTRAZIONE UTENTE
Il caso d'uso richiede all'utente di inserire i propri dati: nome, cognome ed email; viene inoltre richiesto di scegliere una password.
Tale caso d'uso può essere esteso con la richiesta di approvazione di status studente, qualora l'utente lo richieda. Questa azione comporta un dialogo all'esterno del sistema con il sistema universitario.

ACCESSO AREA RISERVATA:
L'accesso all'area riservata dell'utente può avvenire dall'utente registrato tramite l'utilizzo della mail e password con le quali si è iscritto al sistema. Dopo il login possono essere attuate diverse azioni che estendono il caso principale: prova dello status studente, registrazione di un abbonamento o registrazione di una carta di credito. La prova di status studente include un dialogo esterno con il sistema universitario; la registrazione di una carta di credito dialoga con l'istituto di credito per richiedere la validità di tale carta. Infine la registrazione di un abbonamento include direttamente il pagamento per tale servizio, per il pagamento ci si rivolge ancora all'istituto di credito.

NOLEGGIO BICICLETTA:
Per noleggiare una bicicletta l'attore deve essere abbonato e deve autenticarsi al sistema tramite l'utilizzo di email e password scelte durante la registrazione. Le credenziali devono essere inserite sul totem della rastrelliera dalla quale si ha intenzione di noleggiare una bicicletta. Per iniziare il noleggio l'utente deve inoltre scegliere una tipologia di bicicletta tra quelle disponibili. Il totem dopo aver validato la richiesta indica all'attore la posizione sulla rastrelliera della bicicletta riservata all'utente.

RESTITUZIONE BICICLETTA:
L'attore pone la bicicletta in una morsa di una rastrelliera ed inserisce nel totem le proprie credenziali ed il numero di morsa. Contestualmente l'utente avrà la possibilità di comunicare al sistema un danno, ponendo tale bicicletta in manutenzione. Il sistema controllerà che la morsa sia adatta al tipo di bicicletta noleggiato e processa il pagamento tramite la carta di credito inserita in precedenza.
Come estensione di tale caso principale ci può essere l'annullamento dell'abbonamento qualora abbia superato per 3 volte il limite del noleggio e possono essere addebitate delle penali.

GESTIONE RASTRELLIERE:
Generalizzazione dei casi di eliminazione, aggiunta e manutenzione rastrelliere da parte del personale comunale.
Tale gestione include l'autenticazione tramite email e password.

GESTIONE BICICLETTE:
Generalizzazione dei casi di aggiunta, eliminazione e manutenzione delle biciclette da parte del personale comunale.
Tale gestione include l'autenticazione tramite email e password.

2.2 Descrizione degli scenari

Nome	Registrazione
Scopo	Dare la possibilità di registrarsi al sistema
Attore/i	Utente
Pre-condizioni	L'utente non si deve essere mai registrato con l'email e la password che intende usare
Trigger	-
Descrizione sequenza eventi	1. L'attore accede alla maschera di registrazione 2. L'attore compila correttamente tutti i campi richiesti 3. L'attore sceglie una password 4. La password viene accettata dal sistema 5. L'attore è registrato
Alternativa/e	2.a. L'attore non compila correttamente i campi 2.b. L'attore deve ricompilare i campi richiesti.
Post-condizioni	L'attore è registrato al sistema

Nome	Prova status studente
Relations	Extension of: Registrazione Extension of: Accesso area riservata
Scopo	Dar prova dello status di studente
Attore/i	Utente studente e sistema universitario
Pre-condizioni	L'utente deve essere uno studente e quindi possedere e registrarsi con una mail universitaria
Trigger	-
Descrizione sequenza eventi	1. L'attore si registra come utente studente 2. L'attore entra nella maschera per provare il suo status di studente 2. L'attore inserisce l'email universitaria 3. Il sistema richiede la conferma all'istituto universitario 4. Il sistema universitario approva l'email
Alternativa/e	1.a. L'attore accede all'area riservata continua dal punto 2 4.a. Il sistema universitario non approva l'email
Post-condizioni	L'utente è un utente studente

Nome	Registrazione carta di credito
Relations	Extension of: Accesso area riservata
Scopo	Registrare una carta di credito dalla quale verranno stornati i costi dell'abbonamento, noleggio delle biciclette ed eventuali penali
Attore/i	Utente registrato e Istituto di credito
Pre-condizioni	L'utente deve essere registrato e possedere una carta di credito valida
Trigger	-

Descrizione sequenza eventi	1. L’utente accede all’area riservata 2. L’utente inserisce correttamente la carta di credito con tutti i campi richiesti. 3. I dati vengono inviati dal sistema all’istituto di credito 4. L’istituto di credito accerta la validità della carta di credito
Alternativa/e	2.a. L’utente non inserisce correttamente tutti i campi richiesti della carta di credito 2.b. Il sistema richiede all’utente di reinserire correttamente i dati della carta di credito 3.a. L’istituto di credito non accerta la validità della carta di credito 3.b. L’utente torna al punto 1
Post-condizioni	La carta di credito dell’utente viene registrata nel sistema

Nome	Registrazione abbonamento
Relations	Extension of: Accesso area riservata Include: Pagamento
Scopo	Registrare un nuovo abbonamento.
Attore/i	Utente registrato
Pre-condizioni	L’utente deve essere registrato ed aver registrato correttamente una carta di credito. Inoltre deve essere disponibile su quest’ultima il denaro pari all’ammontare del costo dell’abbonamento scelto.
Trigger	Pagamento
Descrizione sequenza eventi	1. L’attore accede all’area riservata 2. L’attore sceglie un’abbonamento 3. Il sistema invia la richiesta di pagamento 4. Il pagamento viene effettuato
Alternativa/e	4.a. Il pagamento non viene effettuato
Post-condizioni	L’utente registrato ha un abbonamento valido

Nome	Accesso area riservata
Relations	Include: Autenticazione
Scopo	Accedere all’area riservata per gestire gli abbonamenti, inserire una carta di credito oppure per provare lo status di studente
Attore/i	Utente registrato
Pre-condizioni	L’utente deve essere registrato.
Trigger	-
Descrizione sequenza eventi	1. L’utente effettua l’autenticazione 2. L’utente visualizza la propria area riservata con gli abbonamenti e le carte di credito
Alternativa/e	-
Post-condizioni	-

Nome	Autenticazione utente
Scopo	Permettere all’utente registrato di autenticarsi ed entrare nella propria area riservata
Attore/i	Utente registrato
Pre-condizioni	L’utente deve essere registrato
Trigger	-
Descrizione sequenza eventi	1. L’utente inserisce nella maschera di autenticazione email e password 2. Il sistema controlla la validità dell’account 3. La validità è accertata
Alternativa/e	1.a. L’utente non inserisce in modo corretto username e email 1.b. Il sistema richiede di inserire username ed email 3.a. Il sistema non accerta la validità dell’account 3.b. rinizia dal punto 1.
Post-condizioni	L’utente è autenticato

Nome	Pagamento
Scopo	Interfacciarsi con l’istituto di credito per attuare una richiesta di pagamento
Attore/i	Istituto di credito
Pre-condizioni	L’utente deve aver registrato una carta di credito valida e deve avere abbastanza denaro per la richiesta fatta
Trigger	-
Descrizione sequenza eventi	1. Il sistema invia all’istituto di credito le informazioni della carta di credito dell’utente ed l’ammontare del pagamento. 2. L’stituto di credito accerta la correttezza della richiesta e processa il pagamento 3. Il sistema riceve il pagamento
Alternativa/e	2.a. L’istituto di credito non accerta la correttezza della carta di credito 2.b. Fallimento dell’istanza di pagamento
Post-condizioni	Il sistema riceve il pagamento dall’istituto di credito

Nome	Noleggio bicicletta
Relations	Include: Autenticazione
Scopo	Permettere all’utente di noleggiare una bicicletta
Attore/i	Utente abbonato Totem rastrelliera
Pre-condizioni	L’utente deve avere un abbonamento attivo/attivabile nel momento in cui vuole noleggiare una bicicletta
Trigger	-
Descrizione sequenza	1. L’attore arriva alla rastrelliera 2. L’attore si autentica nel totem tramite email e password

eventi	e sceglie la tipologia di bicicletta desiderata 3. Il totem controlla la validità 4. Il totem richiede alla rastrelliera di sbloccare la bicicletta prescelta 5. La rastrelliera indica che ha sbloccato la bicicletta correttamente 6. Il totem indica il numero di posteggio della bicicletta assegnata all'attore 7. L'attore preleva la bicicletta
Alternativa/e	3.a. Il totem non valida email e password 3.b. Il totem richiede nuovamente l'autenticazione 3.c. rinizia dal punto 2 3.a. Il totem non trova biciclette del tipo richiesto. 3.b. rinizia dal punto 2
Post-condizioni	Inizio del noleggio

Nome	Restituzione bicicletta
Realtions	Include: Pagamento
Scopo	Permettere all'attore di restituire una bicicletta noleggiata
Attore/i	Utente abbonato Totem rastrelliera
Pre-condizioni	L'attore deve avere noleggiato una bicicletta e non deve ancora averla restituita
Trigger	Pagamento {Condition:!Utente studente && bikeType = normal}
Descrizione sequenza eventi	1. L'attore ripone la bicicletta in una rastrelliera 2. L'attore inserisce nel totem email e password, la posizione nella quale ha posto la bicicletta ed eventuali danni 3. Il sistema controlla che la posizione sia adatta al tipo di bicicletta 4. Il sistema conclude il noleggio e chiude la morsa.
Alternativa/e	3.a. la posizione non è adatta 3.b. ritorna al punto 1
Post-condizioni	Conclusione del noleggio

Nome	Addebito penale
Relations	Include: Pagamento Extension of: Restituzione bicicletta
Scopo	Addebitare una penale all'attore abbonato che non restituisce la bicicletta entro le 24 ore dall'ultimo noleggio
Attore/i	Utente abbonato
Pre-condizioni	L'utente deve avere un abbonamento valido, aver noleggiato la bicicletta ed avere registrato una carta di credito valida
Trigger	Pagamento

Descrizione sequenza eventi	1. L’attore non restituisce la bicicletta entro le 24 ore successive 2. Il sistema gli addebita la penale
Alternativa/e	-
Post-condizioni	Addebito penale sulla carta di credito dell’attore

Nome	Annullamento abbonamento
Relations	Extension of: Restituzione bicicletta
Scopo	Annullare l’abbonamento all’attore che supera il limite massimo di 2 ore per tre volte
Attore/i	Utente abbonato
Pre-condizioni	L’utente deve essere in possesso di un abbonamento valido, aver noleggiato una bicicletta per almeno 3 volte.
Trigger	-
Descrizione sequenza eventi	1. L’attore restituisce la bicicletta passate 2 ore dal noleggio 2. Il sistema calcola il numero di volte nel quale non ha rispettato tale vincolo 3. Il sistema trova che l’attore è recidivo (>2) 4. Il sistema revoca l’abbonamento all’attore
Alternativa/e	-
Post-condizioni	Il sistema revoca l’abbonamento all’attore

Nome	Comunicazione danni
Relations	Extension of: Restituzione bicicletta
Scopo	Dare la possibilità all’attore di comunicare danni alla bicicletta noleggiata, in modo da permettere una manutenzione
Attore/i	Utente abbonato
Pre-condizioni	L’attore deve possedere un abbonamento valido ed aver noleggiato una bicicletta
Trigger	-
Descrizione sequenza eventi	1. L’attore noleggia una bicicletta 2. L’attore si accorge di danni alla bicicletta 3. L’attore restituisce la bicicletta e comunica il danno 4. La rastrelliera rende indisponibile tale bicicletta
Alternativa/e	-
Post-condizioni	La bicicletta viene resa indisponibile

Nome	Gestione biciclette
Relations	Generalization of: - Aggiunta bicicletta - Eliminazione bicicletta - Manutenzione bicicletta Include: Autenticazione personale comunale
Scopo	Dare la possibilità all’attore di gestire le biciclette del

	sistema di bike sharing
Attore/i	Personale comunale
Pre-condizioni	L’attore deve essere autenticato come personale comunale
Trigger	-
Descrizione sequenza eventi	1. L’attore si autentica come personale comunale 2. L’attore entra nel pannello di gestione e attuerà le azioni opportune.
Alternativa/e	-
Post-condizioni	L’attore entra nel pannello di gestione delle biciclette

Nome	Manutenzione bicicletta
Relations	Specialization of: Gestione biciclette
Scopo	Permettere all’attore di attuare una manutenzione ad una bicicletta
Attore/i	Personale comunale
Pre-condizioni	L’attore deve essere autenticato come personale comunale e deve esistere una bicicletta in manutenzione
Trigger	-
Descrizione sequenza eventi	1. L’attore si autentica come personale comunale 2. L’attore entra nel pannello di gestione delle biciclette di una rastrelliera 3. L’attore entra nel pannello di manutenzione 4. L’attore seleziona una bicicletta 5. L’attore sistema la bicicletta 6. L’attore indica al sistema la conclusione della manutenzione 7. Il sistema rende di nuovo disponibile la bicicletta
Alternativa/e	-
Post-condizioni	La bicicletta torna disponibile

Nome	Eliminazione bicicletta
Relations	Specialization of: Gestione biciclette
Scopo	Permettere all’attore di eliminare una bicicletta dal sistema
Attore/i	Personale comunale
Pre-condizioni	L’attore deve essere autenticato come personale comunale
Trigger	-
Descrizione sequenza eventi	1. L’attore si autentica come personale comunale 2. L’attore entra nel pannello di gestione delle biciclette di una rastrelliera 3. L’attore seleziona una bicicletta da eliminare 4. Il sistema provvede ad eliminare la bicicletta 5. Il sistema conferma la corretta eliminazione della bicicletta
Alternativa/e	-
Post-	La bicicletta viene eliminata dal sistema

condizioni	

Nome	Aggiunta bicicletta
Relations	Specialization of: Gestione biciclette
Scopo	Permettere all’attore di aggiungere una nuova bicicletta al sistema
Attore/i	Personale comunale
Pre-condizioni	L’attore deve essere autenticato come personale comunale
Trigger	-
Descrizione sequenza eventi	1. L’attore si autentica come personale comunale 2. L’attore entra nel pannello di gestione delle biciclette di una rastrelliera 3. L’attore inserisce i dati relativi la bicicletta richiesti dal sistema 4. Il sistema convalida i dati 5. Il sistema conferma la corretta aggiunta della bicicletta
Alternativa/e	4.a. Il sistema non conferma i dati 4.b. Il sistema richiede all’attore di reinserire i dati in modo corretto 4.c. continua da punto 4
Post-condizioni	La bicicletta viene aggiunta al sistema

Nome	Gestione rastrelliere
Relations	Generalization of: - Eliminazione rastrelliera - Manutenzione rastrelliera - Aggiunta rastrelliera Include: Autenticazione personale comunale
Scopo	Permettere all’attore di gestire le rastrelliere del sistema
Attore/i	Personale comunale
Pre-condizioni	L’attore deve essere autenticato come personale comunale
Trigger	-
Descrizione sequenza eventi	1. L’attore si autentica come personale comunale 2. L’attore entra nel pannello di gestione delle rastrelliere dal quale potrà scegliere le azioni opportune.
Alternativa/e	-
Post-condizioni	L’attore entra nel pannello di gestione delle rastrelliere

Nome	Eliminazione rastrelliera
Relations	Specialization of: Gestione rastrelliere
Scopo	Permettere all’attore di eliminare le rastrelliere dal sistema
Attore/i	Personale comunale
Pre-condizioni	L’attore deve essere autenticato come personale comunale e la rastrelliera deve essere vuota
Trigger	-

Descrizione sequenza eventi	1. L’attore si autentica come personale comunale 2. L’attore entra nel pannello di gestione e seleziona una rastrelliera 3. L’attore elimina la rastrelliera 4. Il sistema controlla che la rastrelliera sia vuota 5. Il sistema conferma che la rastrelliera selezionata può essere eliminata 6. Il sistema elimina la rastrelliera 7. Il sistema invia un messaggio all’attore di corretta eliminazione della rastrelliera
Alternativa/e	6.a. Il sistema trova delle biciclette legate alla rastrelliera 6.b. Il sistema indica all’attore che è necessario che la rastrelliera sia prima vuota per essere eliminata
Post-condizioni	La rastrelliera viene eliminata dal sistema

Nome	Aggiunta rastrelliera
Relations	Specialization of: Gestione rastrelliere
Scopo	Permettere all’attore di aggiungere una rastrelliera al sistema
Attore/i	Personale comunale
Pre-condizioni	L’attore deve essere autenticato come personale comunale
Trigger	-
Descrizione sequenza eventi	1. L’attore si autentica come personale comunale 2. L’attore entra nel pannello di gestione 3. L’attore entra nel pannello di aggiunta rastrelliera 4. L’attore inserisce i dati richiesti dal sistema riguardanti la nuova rastrelliera 5. Il sistema controlla la correttezza dei dati 6. Il sistema conferma la correttezza dei dati immessi 7. Il sistema conferma l’aggiunta della rastrelliera al sistema
Alternativa/e	6.a. Il sistema indica che i dati non sono stati immessi correttamente 6.b. Il sistema chiede il reinserimento dei dati all’attore 6.c. continua da punto 4
Post-condizioni	La rastrelliere viene aggiunta al sistema

Nome	Manutenzione rastrelliera
Relations	Specialization of: Gestione rastrelliere
Scopo	Permettere all’attore di fare manutenzione ad una rastrelliera
Attore/i	Personale comunale
Pre-condizioni	L’attore deve essere autenticato come personale comunale
Trigger	-

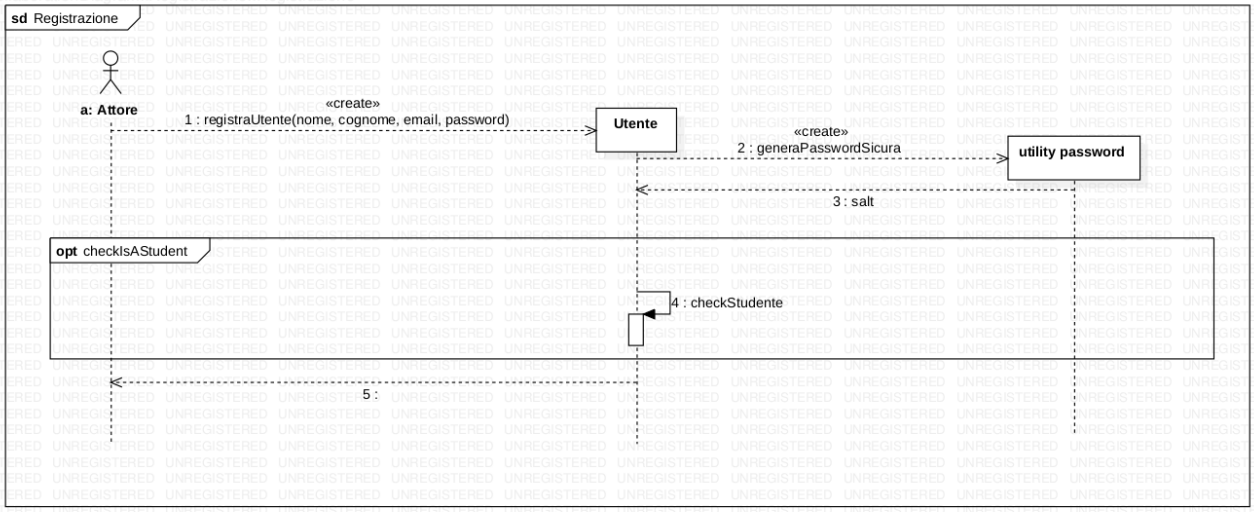
Descrizione sequenza eventi	1. L’attore si autentica come personale comunale 2. L’attore entra nel pannello di gestione e sceglie la rastrellirea 3. L’attore sceglie la morsa che ha sistemato 4. Il sistema aggiorna lo stato della morsa
Alternativa/e	-
Post-condizioni	La morsa viene sistemata

Nome	Produzione dati statistici
Relations	Include: Autenticazione personale comunale
Scopo	Permettere all’attore di avere dei dati statistici da parte del sistema
Attore/i	Personale comunale
Pre-condizioni	L’attore deve essere autenticato come personale comunale
Trigger	-
Descrizione sequenza eventi	1. L’attore si autentica come personale comunale 2. L’attore entra nel pannello dei dati statistici
Alternativa/e	-
Post-condizioni	L’attore visualizza diverse viste di dati statistici

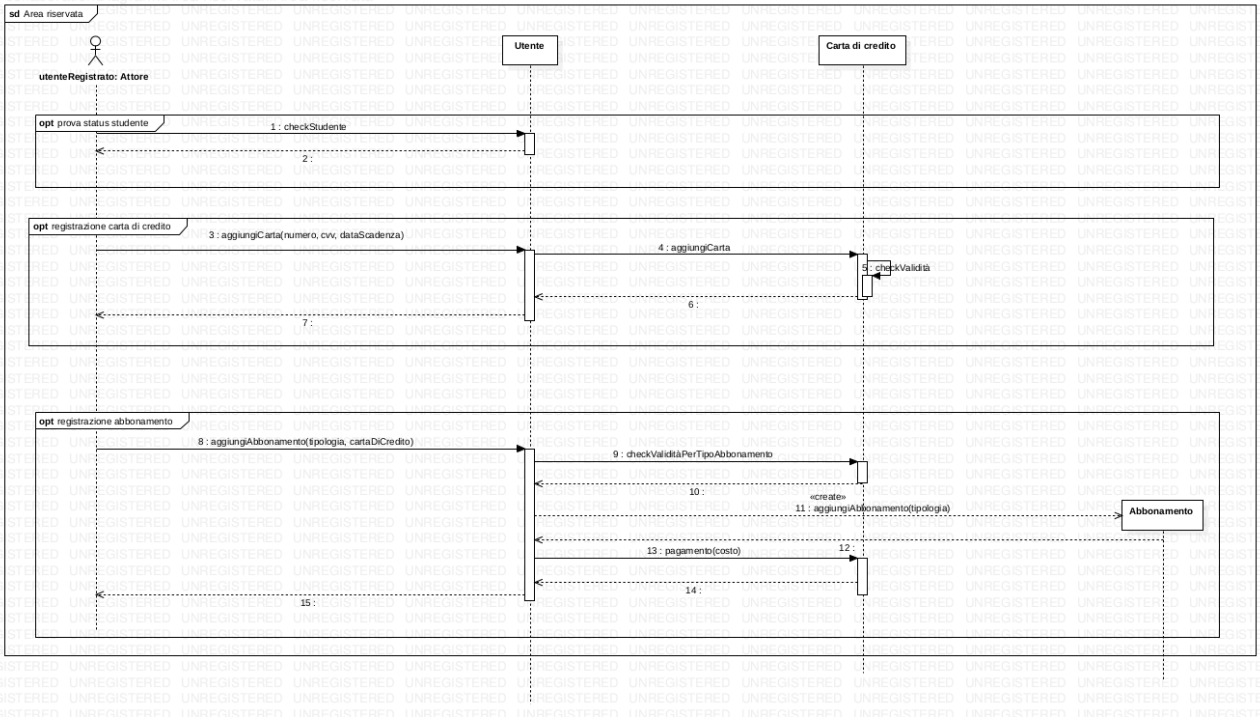
Nome	Autenticazione personale comunale
Scopo	Permettere all’attore di autenticarsi come personale comunale per avere dei permessi privilegiati e specifici sul sistema
Attore/i	Personale comunale
Pre-condizioni	L’attore che si vuole autenticare come personale comunale deve registrato e deve avere il campo is_admin in database settato a true
Trigger	-
Descrizione sequenza eventi	1. L’attore entra nella maschera di autenticazione come personale comunale 2. L’attore inserisce i dati richiesti per l’autenticazione correttamente 3. Il sistema controlla la validità dell’account 4. La validità viene accertata
Alternativa/e	2.a. L’attore non inserisce correttamente i dati richiesti per l’autenticazione 2.b. Il sistema richiede all’attore di reinserire i dati 2.c. continua dal punto 2 4.a. Il sistema non accerta la validità dell’account 4.b. rinizia dal punto 1.
Post-condizioni	L’utente è autenticato

2.3 Diagrammi di sequenza

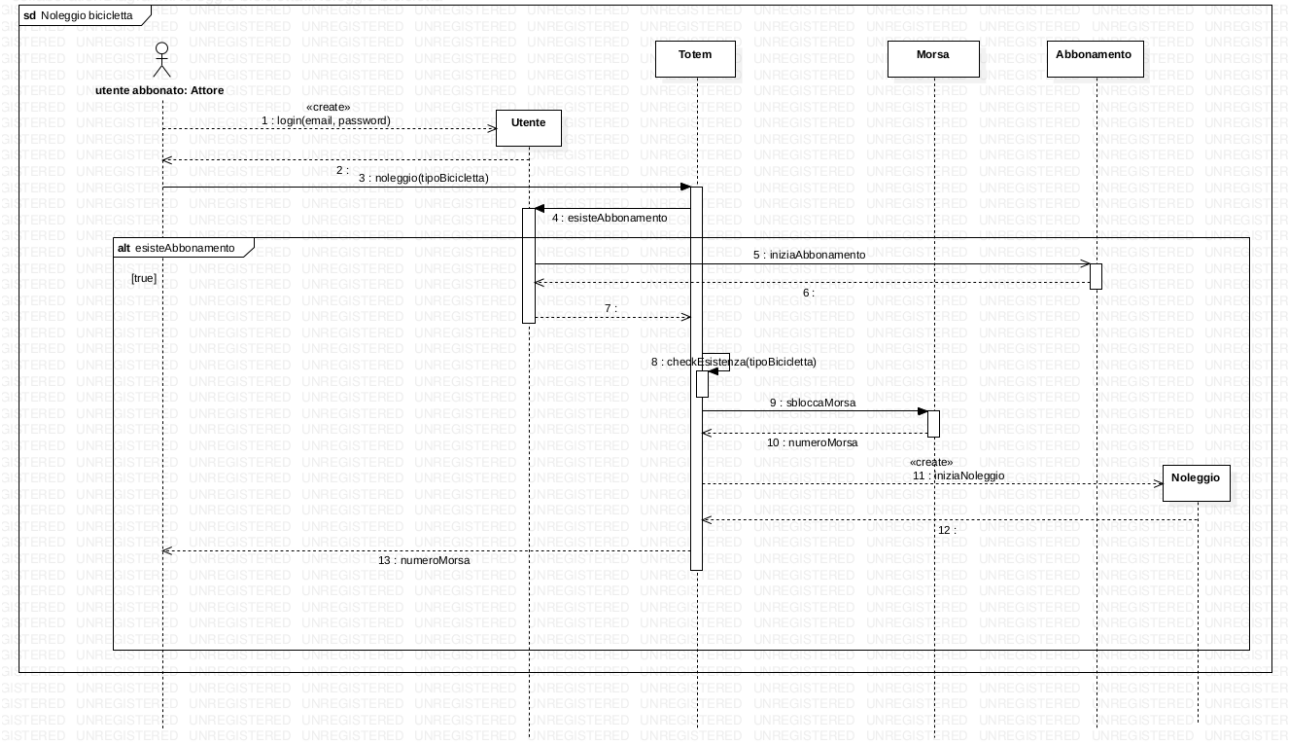
REGISTRAZIONE UTENTE



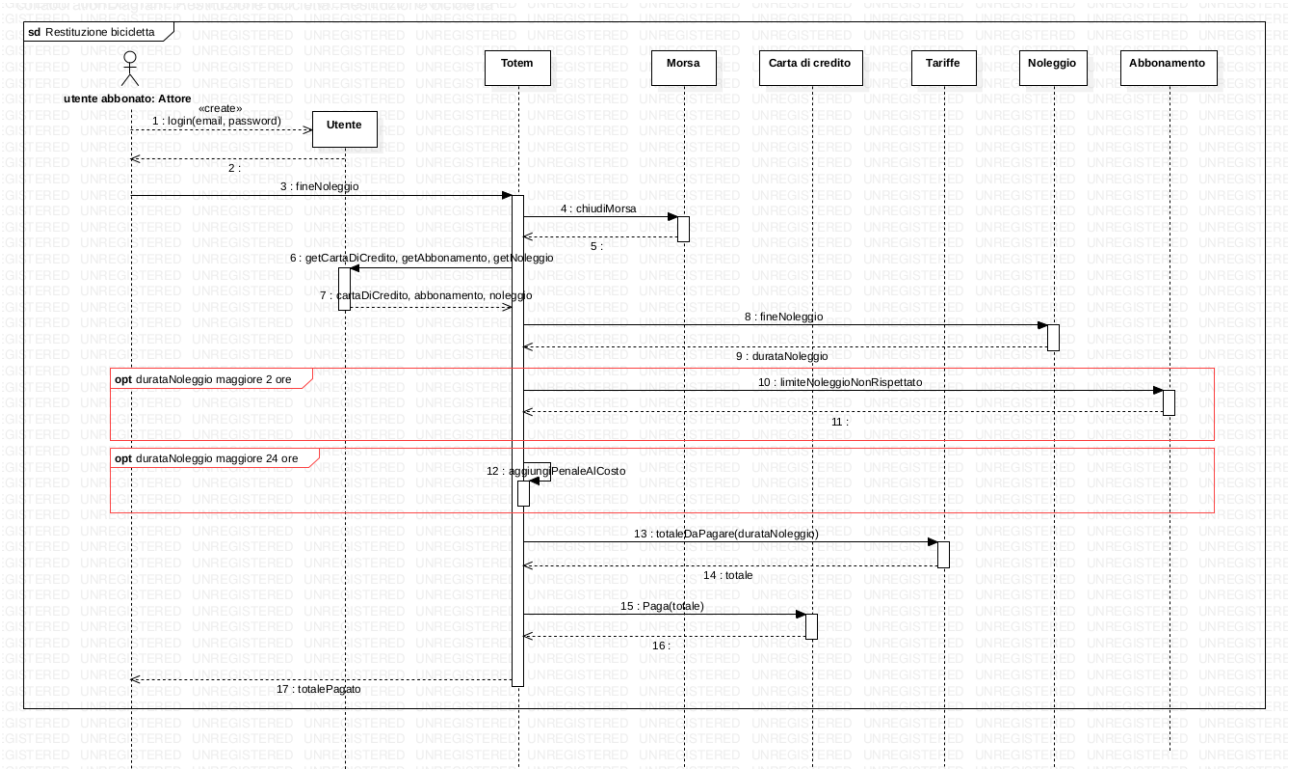
AREA RISERVATA



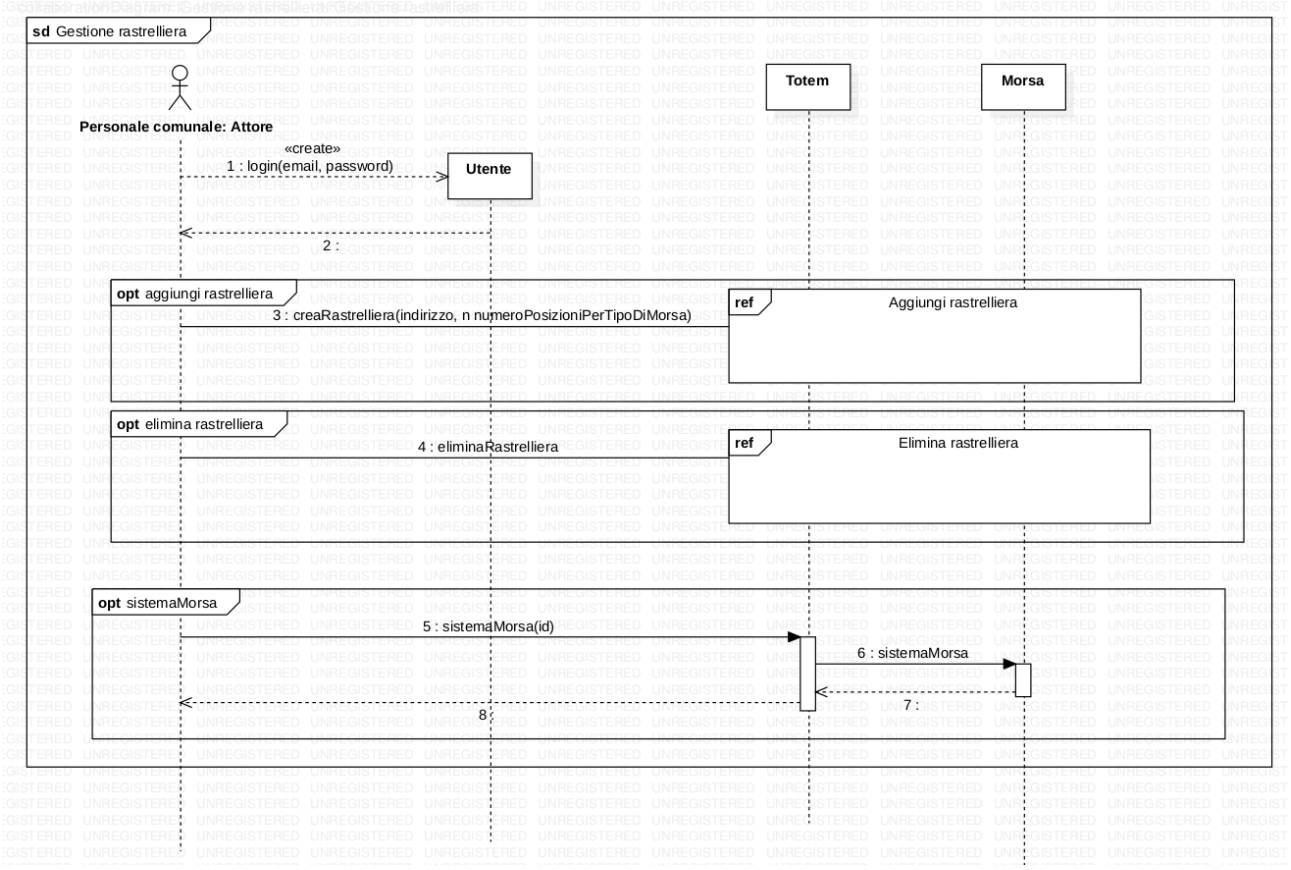
NOLEGGIO BICICLETTA

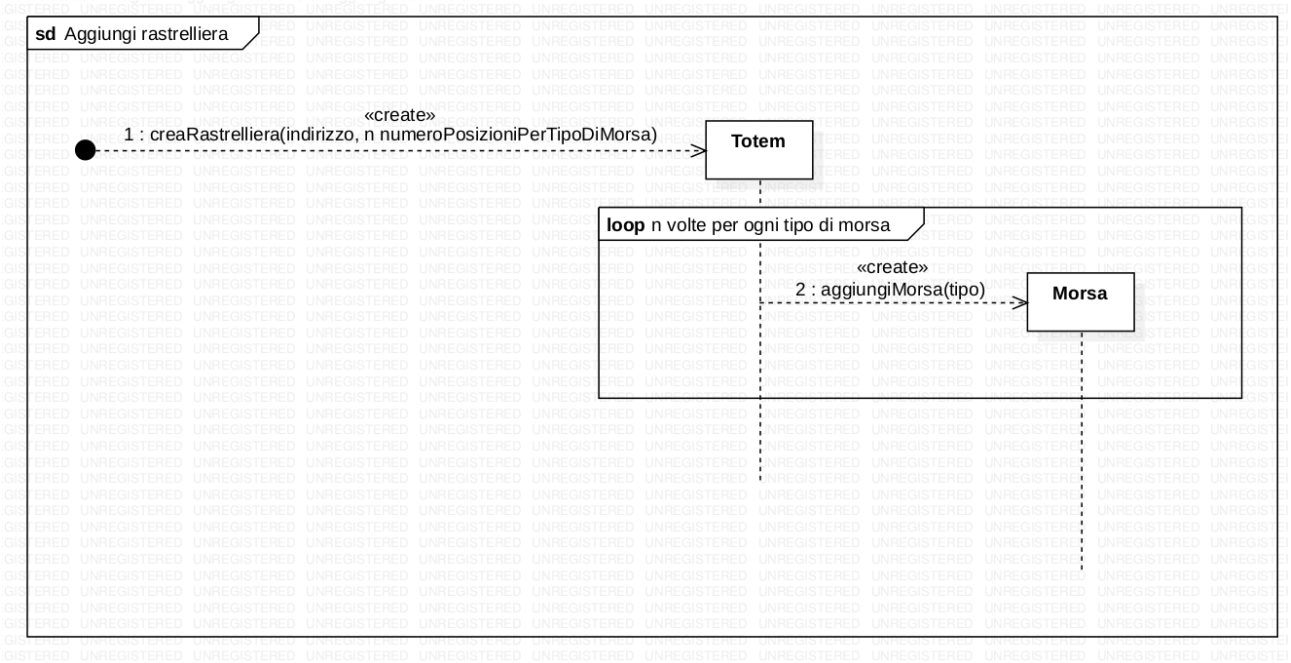


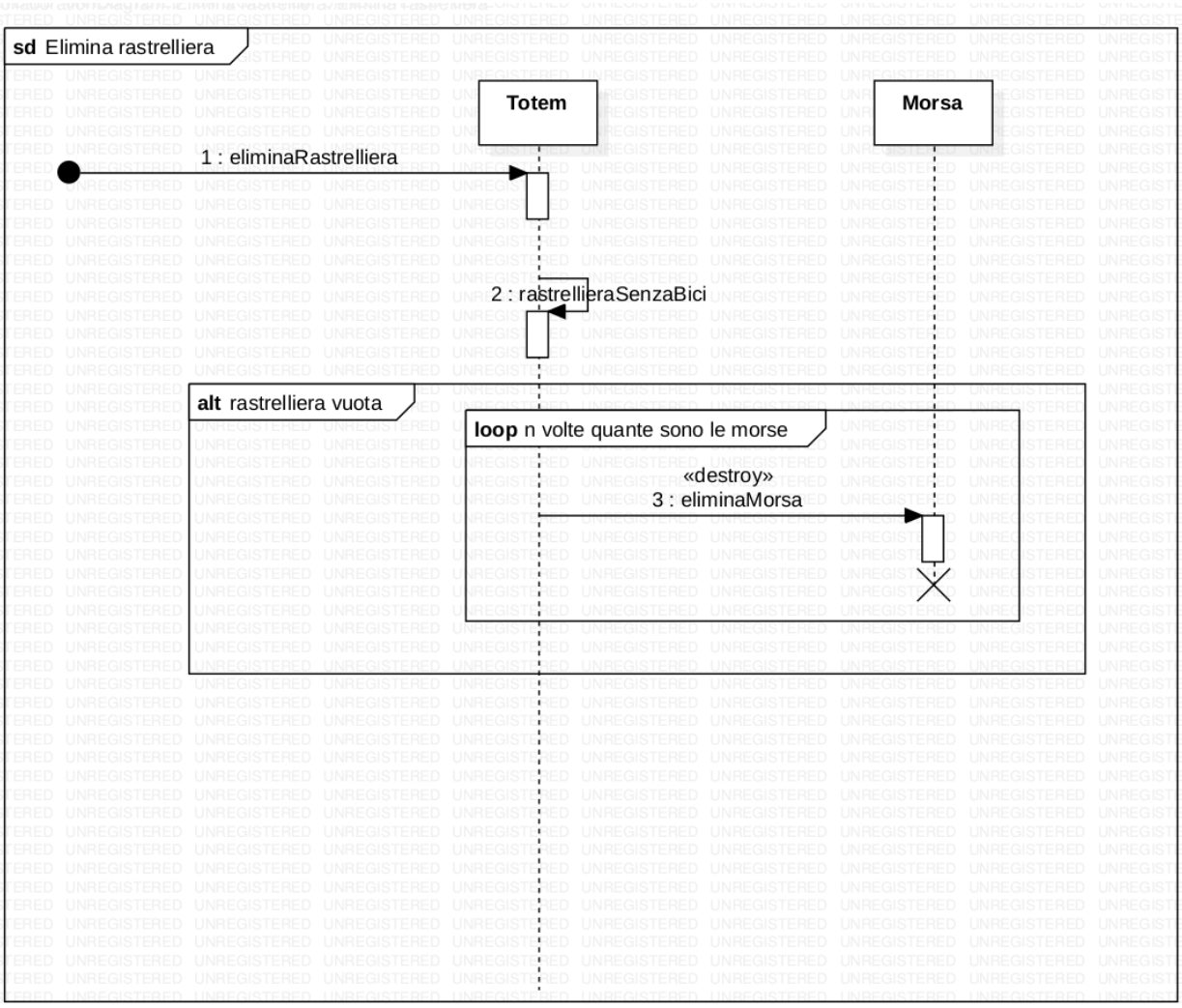
RESTITUZIONE BICICLETTA



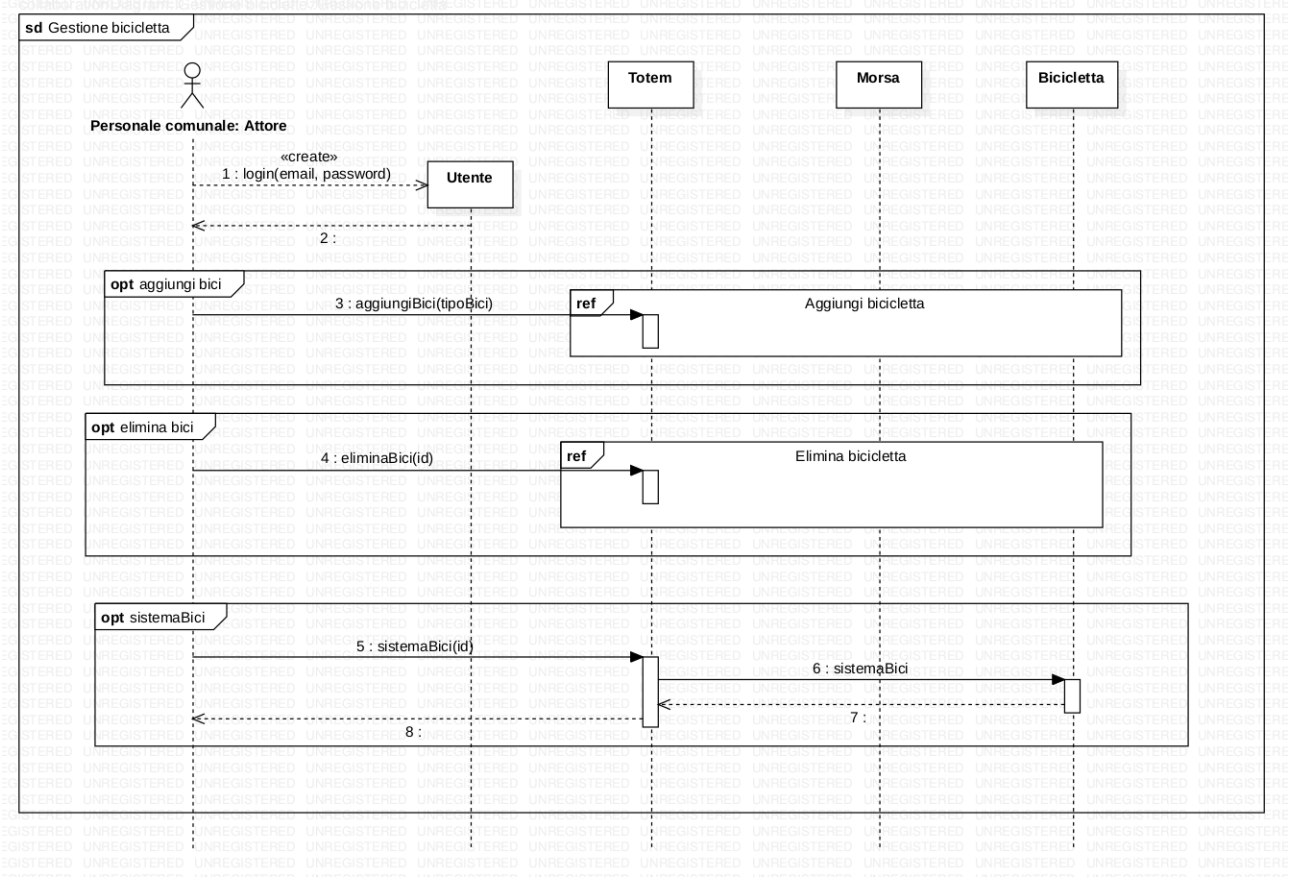
GESTIONE RASTRELLIERE

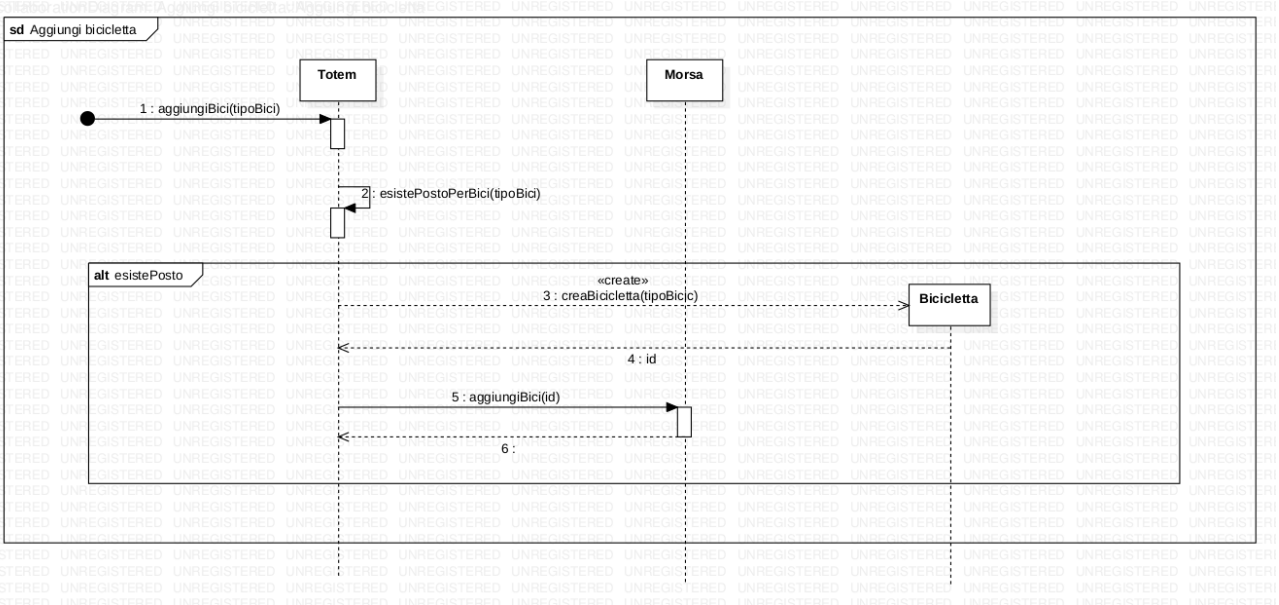


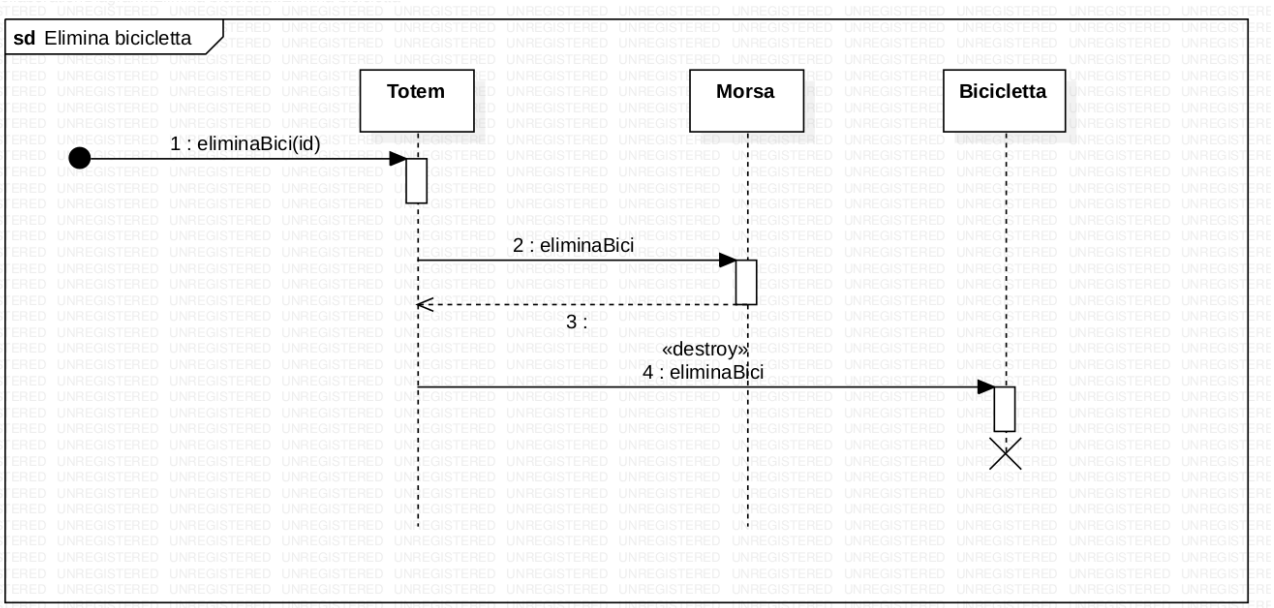



sd Elimina rastrelliera
Totem
Morsa
1 : eliminaRastrelliera
2 : rastrellieraSenzaBici
alt rastrelliera vuota
loop n volte quante sono le morse
«destroy»
3 : eliminaMorsa

GESTIONE BICICLETTE



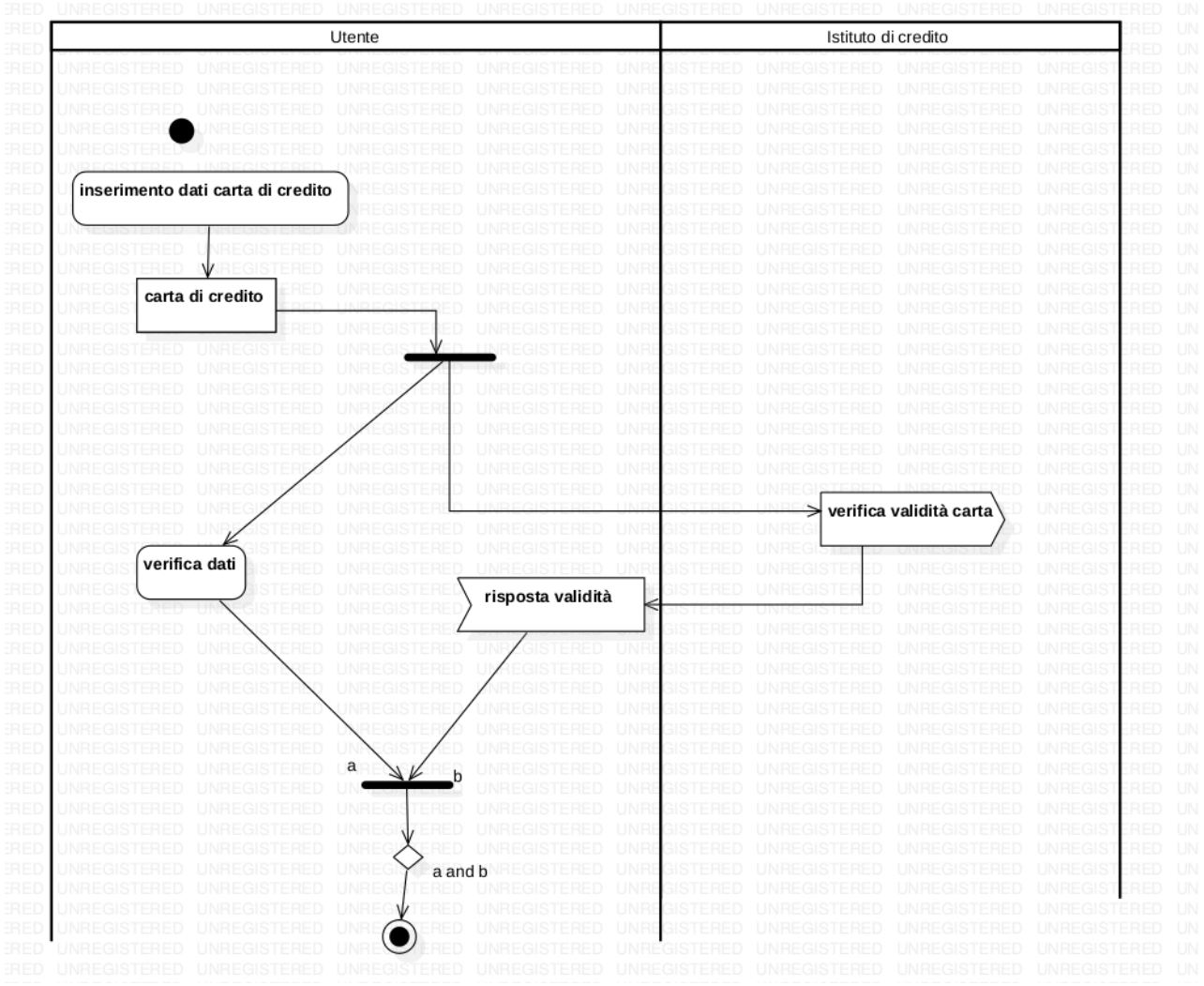



sd Elimina bicicletta
Totem
Morsa
Bicicletta
1 : eliminaBici(id)
2 : eliminaBici
3 :
«destroy»
4 : eliminaBici

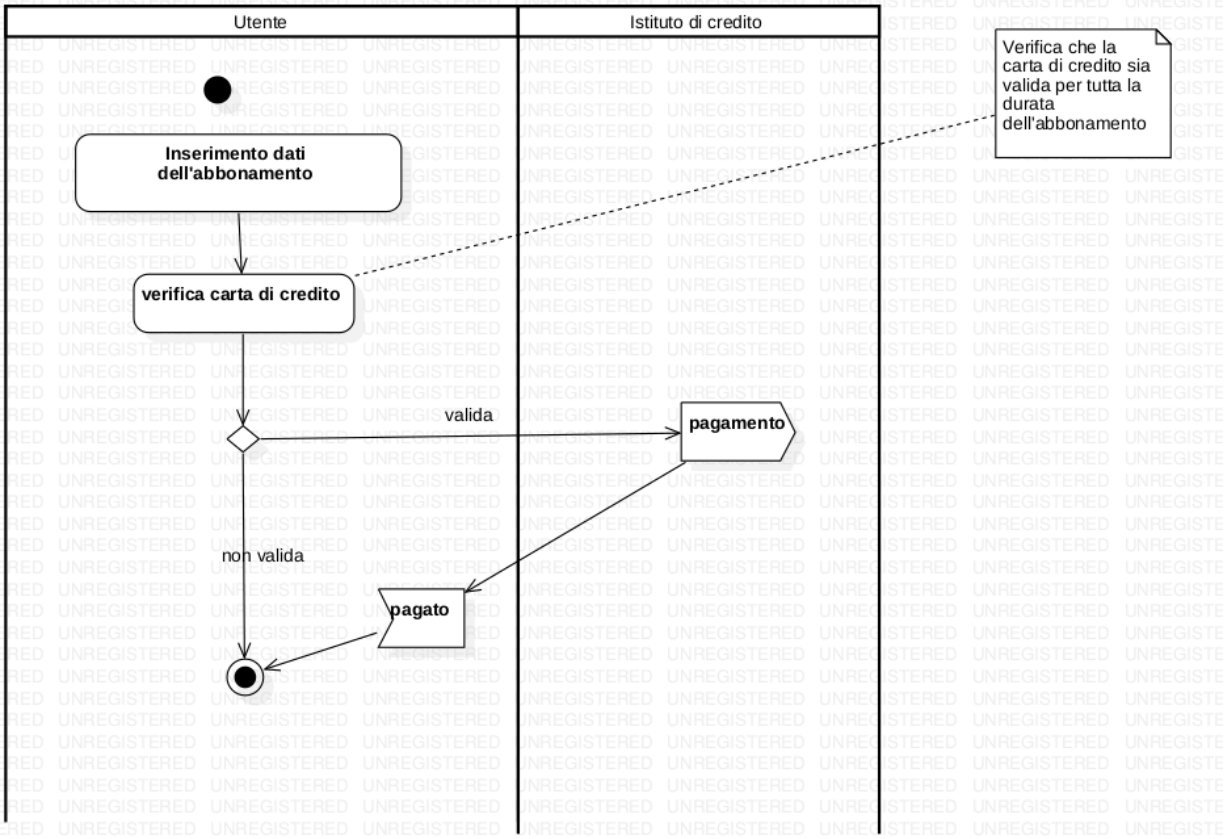
2.4 Diagrammi delle attività

Vengono riportati i diagrammi di attività in riferimento ai casi d'uso:

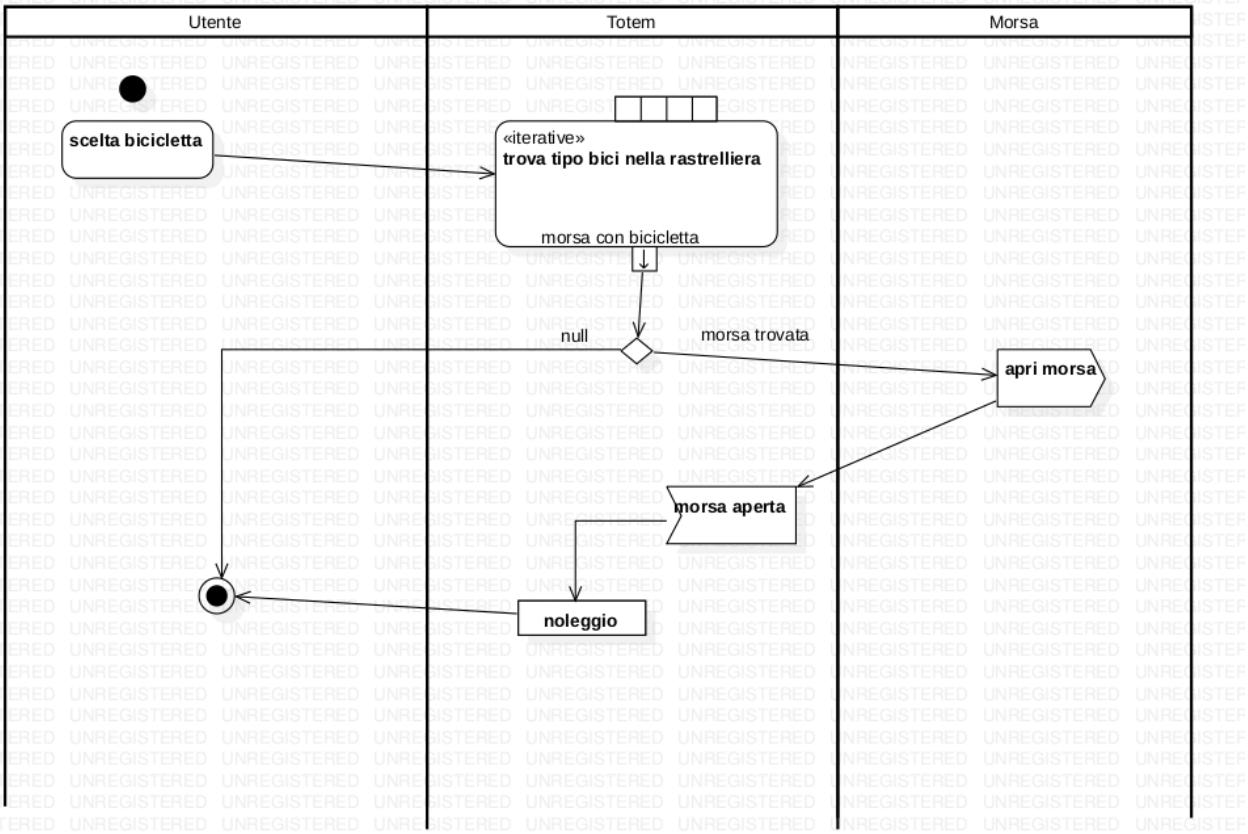
AGGIUNTA CARTA DI CREDITO



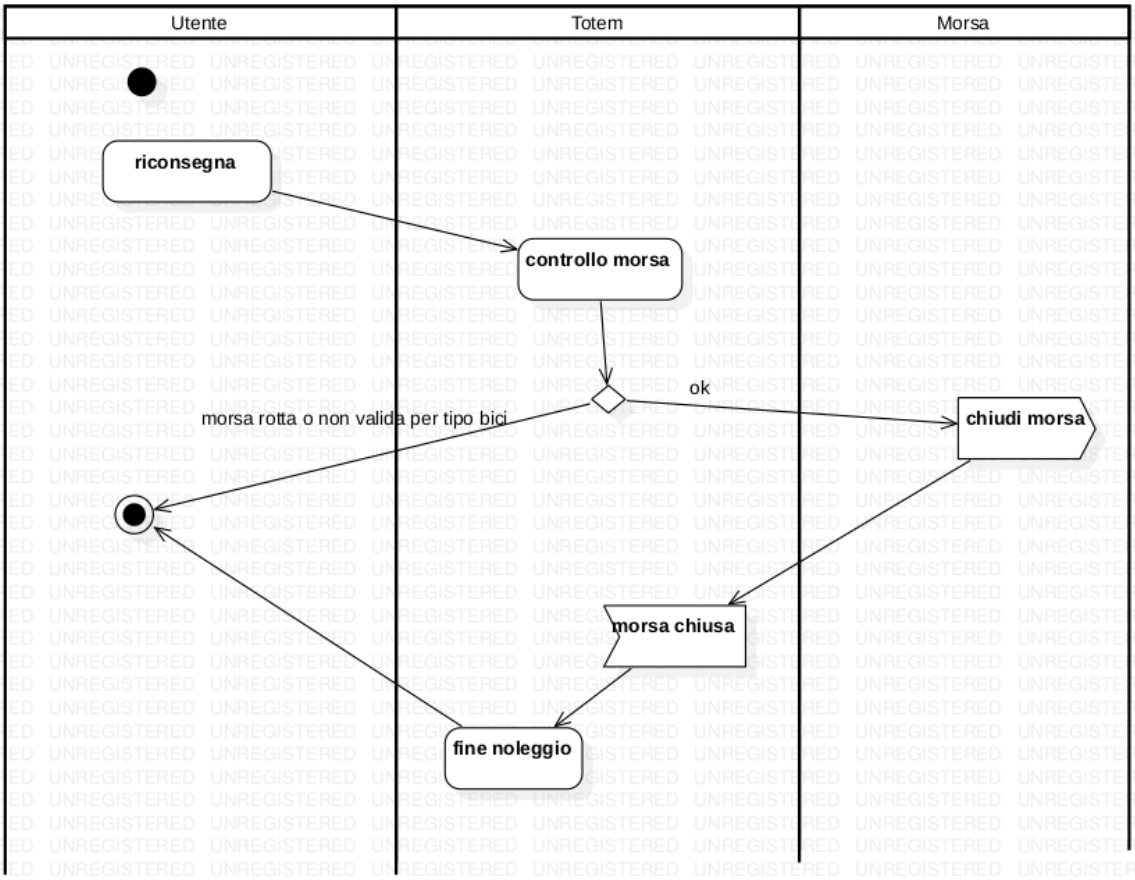
REGISTRAZIONE DI UN ABBONAMENTO



NOLEGGIO BICICLETTA

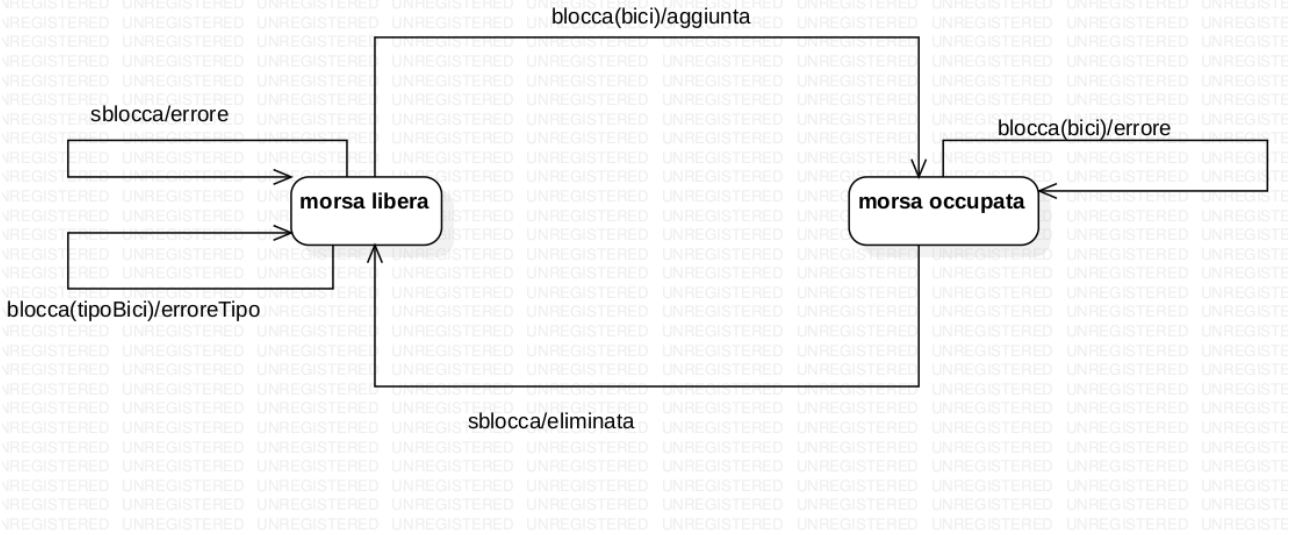


RICONSEGNA BICICLETTA



2.5 Macchine di stato

Macchina di stato morsa.
<S={Morsa libera, morsa occupata},
I = {blocca(tipoBici), sblocca()},
O = {aggiunta, eliminata, errore, erroreTipo}
T = {(morsa libera, blocca, aggiunta, morsa occupata),(morsa occupata, sblocca, eliminata, morsa libera)}>

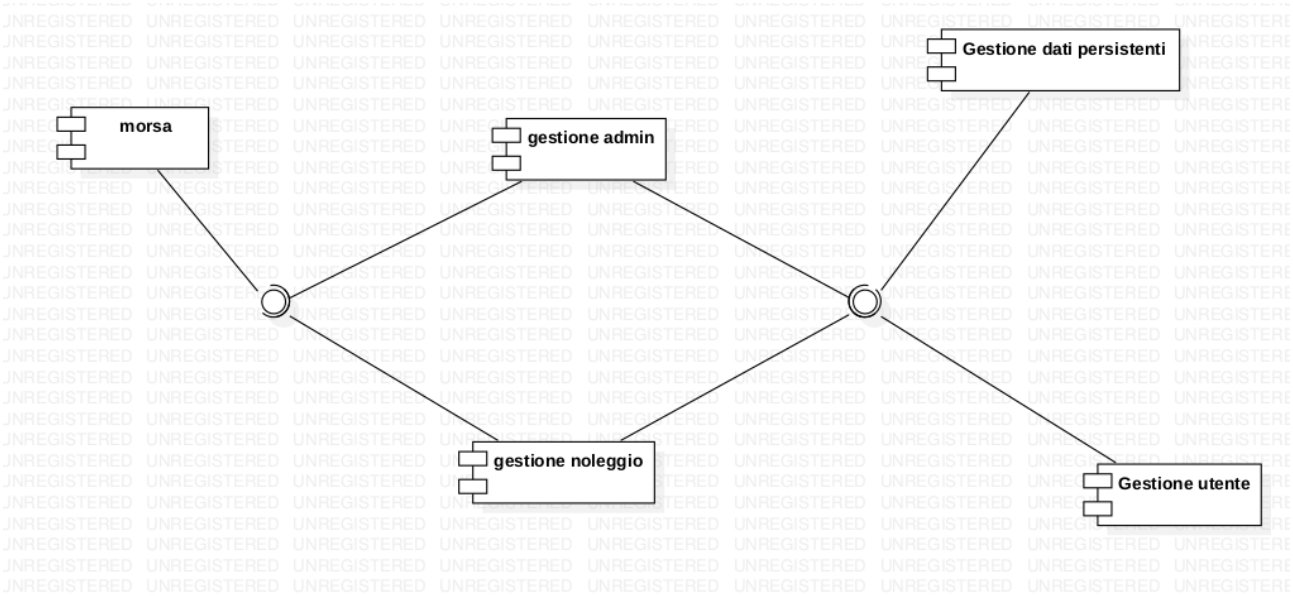


Il funzionamento di tale macchina è il seguente:
Se la morsa è libera e viene fatta un'operazione di blocco(tipoBici adatto alla morsa) si passa allo stato di morsa occupata e la bici viene aggiunta.
Al contrario se la morsa è occupata e viene fatta un'operazione di sblocca() si passa allo stato di morsa libera e viene sbloccata la morsa eliminando la bici dalla morsa.
Se ci trovassimo nello stato di morsa libera e venisse fatta un'operazione di sblocca ci sarebbe un messaggio d'errore, stesso comportamento nello stato di morsa occupata con operazione di blocca(tipoBici).
Un'altro messaggio d'errore specifico sul tipo di bicicletta sbagliato lo si riceve quando si fa un operazione di blocca(tipo bici errato per morsa) nello stato di morsa libera.

Per l'implementazione di tale macchina di stato è stato usato in modo massivo (come in tutto il sistema) il database. In questo modo diversi servizi potrebbero collegarsi allo stesso database e sapere lo stato della morsa in un dato momento.
Tale macchina di stato è stata implementata all'interno della classe RackPosition. Morsa libera è lo stato quando il campo bike = null. Morsa occupata invece è quando il campo bike != null. Quando viene fatta la chiamata a lock(tipoBici) si passa allo stato di morsa occupata (se il tipo della bicicletta è adatto) . Il controllo sul tipo della bicicletta è attuato dal metodo

isFreeAndAccessibleForBikeType. Da morsa occupata si passa a morsa libera con la chiamata al metodo unlock().
Con i metodi lock e unlock ci si riferisce anche all'hardware e gli si da il compito di attuare un'azione.
Le risposte dell'hardware, non avendo la possibilità di essere testate realmente, sono state create nel config.json. In questo modo possiamo testare nel codice sia l'errore che il conseguimento dell'azione richiesta.

2.6 Diagramma dei componenti

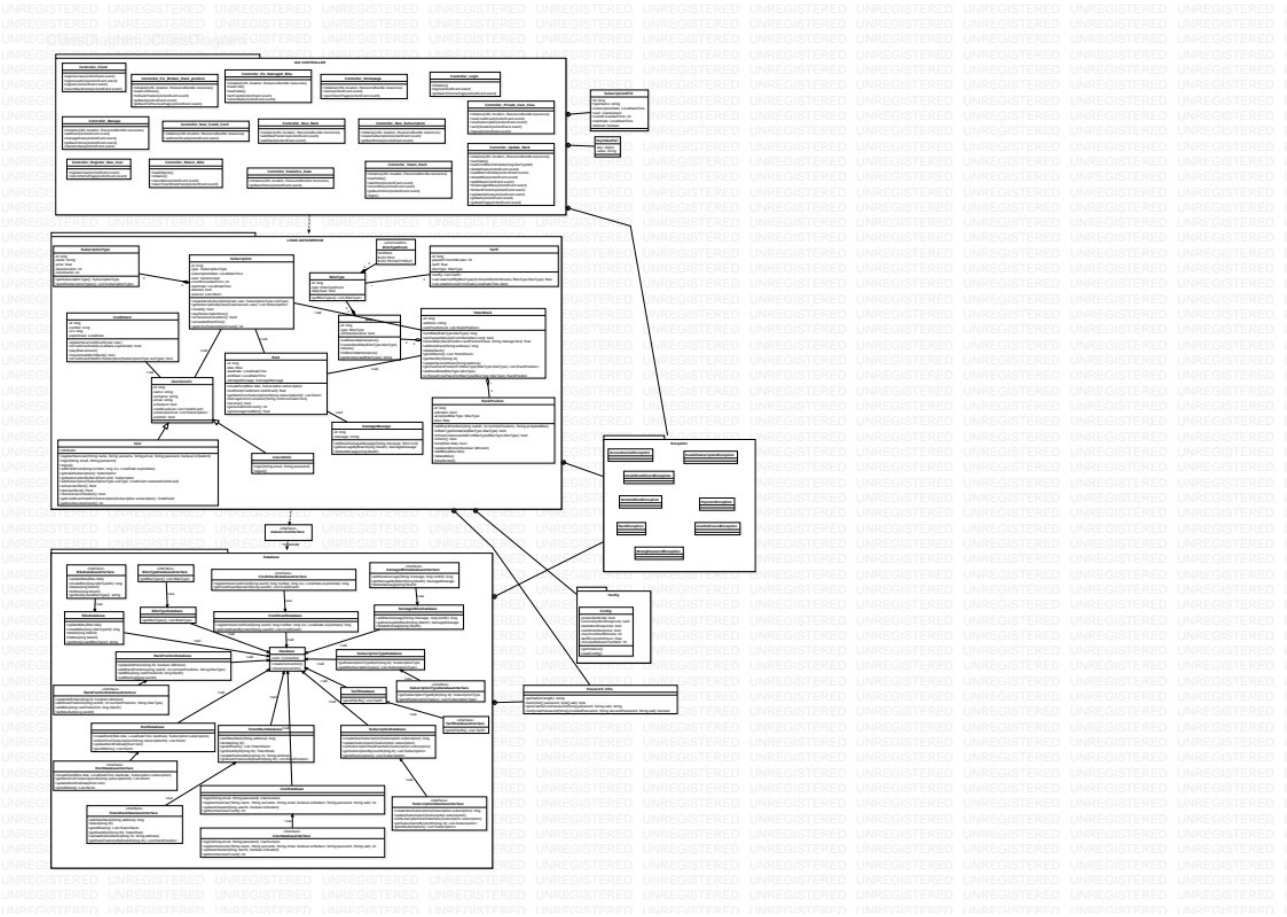


Dallo schema si può notare che il sistema è composto da diversi componenti: morsa, gestione noleggio, gestione admin, gestione dati persistenti e gestione utente.
Il componente gestione dati persistenti contiene tutte le classi necessarie all'interazione con il database per tenere aggiornati i dati.
Il componente gestione utente comprende le classi necessarie per le interazioni riguardanti l'utente(creazione abbonamento, aggiunta carta di credito..).
Il componente gestione noleggio comprende le classi inerenti alle azioni necessarie per un noleggio completo.
Il componente morsa raggruppa invece tutte le funzionalità legate alla gestione delle rastrelliere.
L'ultimo componente, gestione admin, comprende le classi per attuare le azioni da parte dell'utente admin.

3 Implementazione del sistema

3.1 Diagramma delle classi (modello di programma)

- Nota1: Il codice è stato sviluppato in inglese per abitudine personale nel programmare, di conseguenza le classi avranno il corrispettivo nome in inglese. La classe morsa è rappresentata da rackPosition.
- Nota2: i metodi get e set non sono stati riportati perché considerati impliciti.

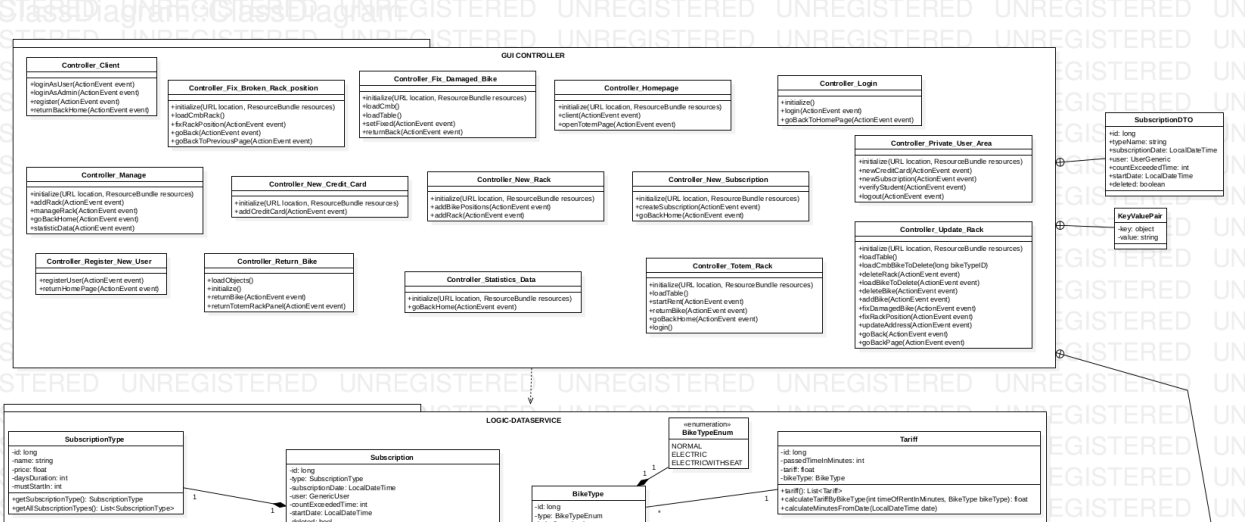


Le classi sono organizzate in diversi package:
-Controller: rappresenta la gui e comprende una parte di logica di verifica degli input inseriti dall'utente
-Dataservice: rappresenta la logica principale del sistema con tutte le classi necessarie.
-Database: contiene tutte le classi necessarie al salvataggio dei dati in modo persistente all'interno del database.
- Config: contiene la classe che legge il file di configurazione. Il file di configurazione è un file json e i parametri al suo interno sono alcuni mock di risposte da parte di agenti esterni e altre configurazioni come minutesBetweenTwoRent, tariffExceeded24Hours... È stato creato tale file per permettere in un secondo momento di riprogrammare tali parametri, qualora ne fosse necessario, senza dover modificare tutto il codice.
-Exception: eccezioni utilizzate nel codice

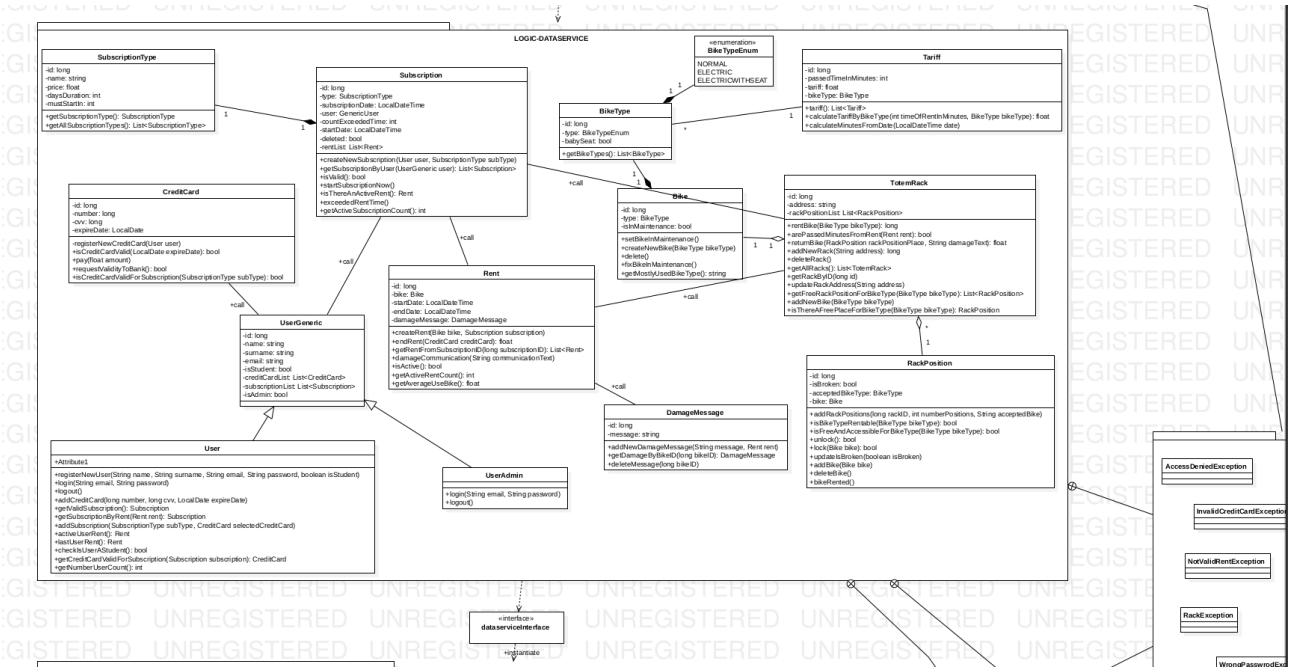
Il package dataservice dialoga strettamente con il package database tramite l’interfaccia dataserviceInterface che istanzia le classi di database.

Qui sotto vengono ripresentati i package:

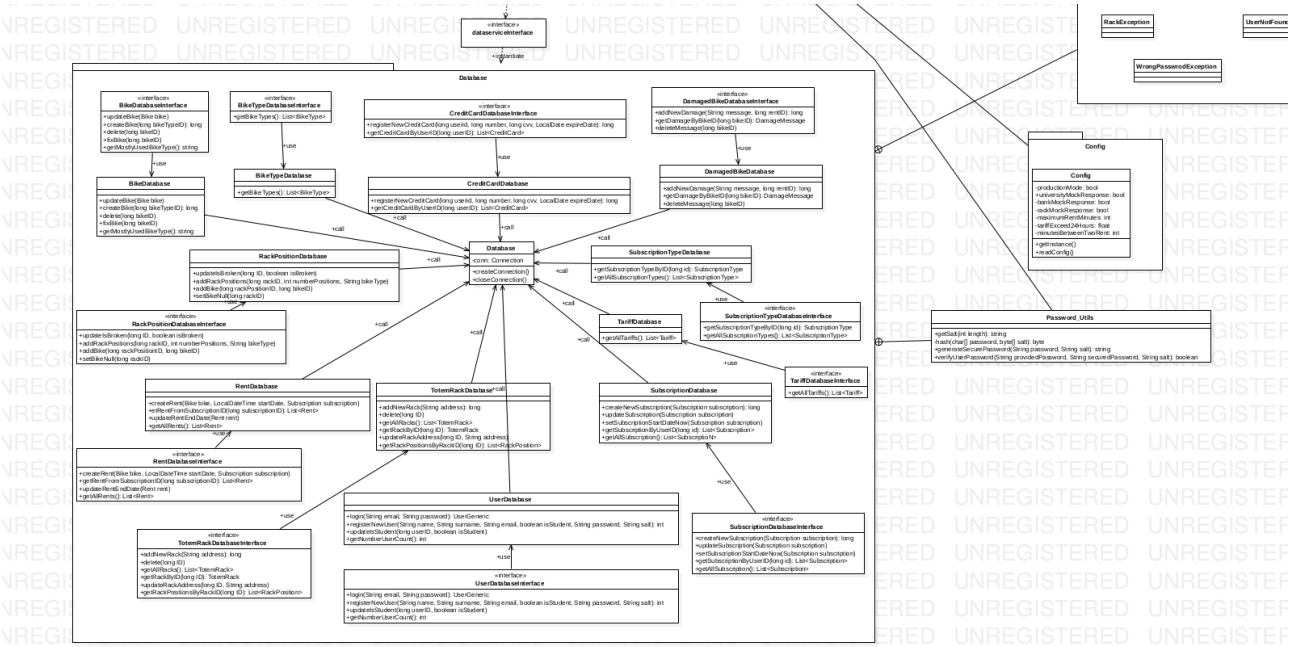
Controller:
Le classi di controller non contengono relazioni al loro interno, si interfacciano direttamente con le classi all’interno del package dataservice. Per cercare di avere uno schema più pulito non sono state rappresentate le relazioni perché sono tutte relazioni di tipo ‘instantiate’ e ‘call’ di classi all’interno del package dataservice.



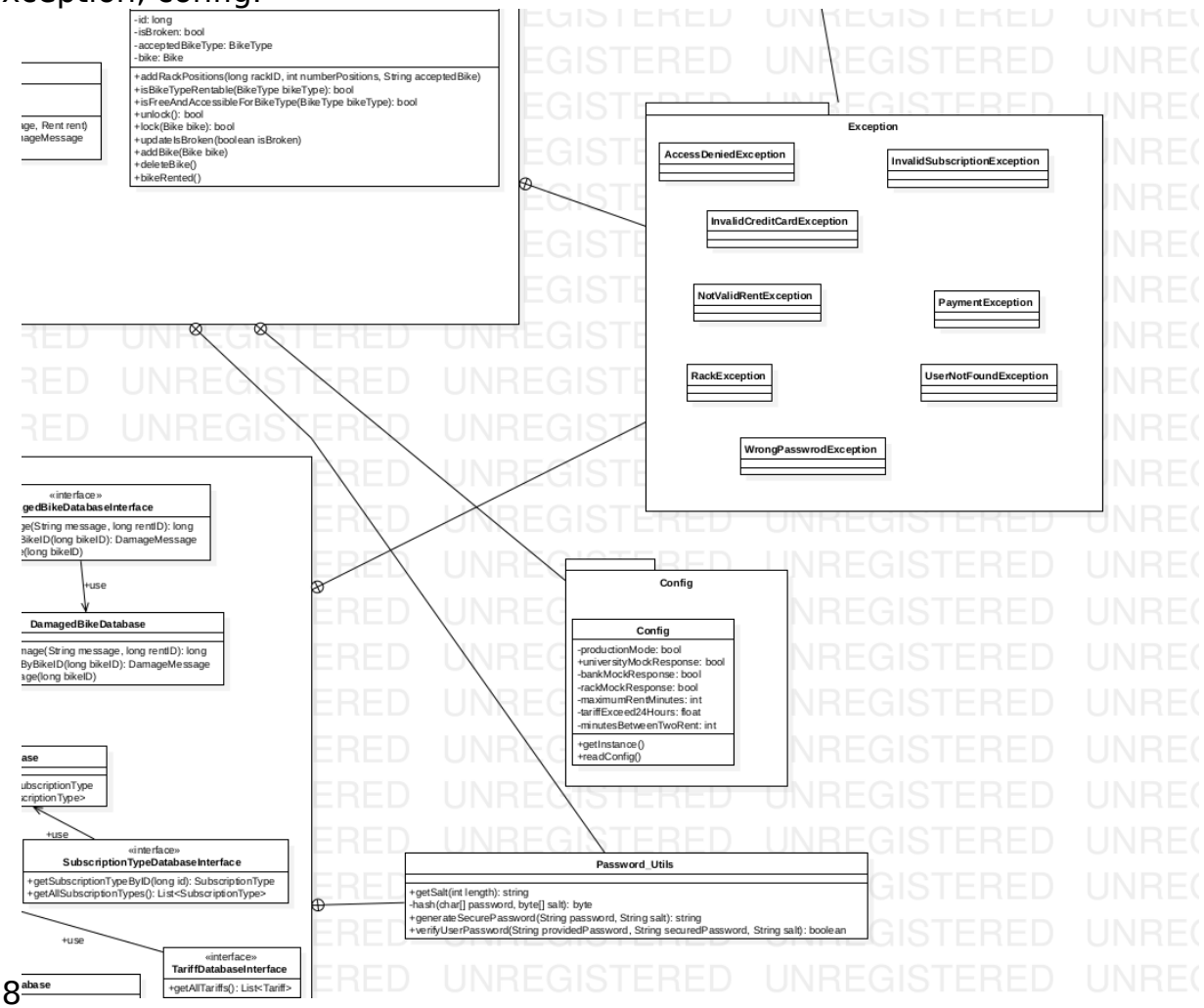
Dataservice:



Database:



Exception, config:



3.1.1 Discussione dei Design Pattern utilizzati

Singleton:

Il singleton pattern consiste nel gestire delle classi che prevedono l'instanziazione di un unico oggetto, uguale per tutte le classi e che l'istanza sia visibile globalmente.
Un'esempio di utilizzo di tale pattern è la classe database all'interno del package database. Questa classe viene istanziata direttamente dal main tramite la chiamata al metodo getInstance il quale se non esiste alcuna istanza ne crea una nuova, altrimenti ritorna quella esistente.
In questo modo viene creata la connessione al database che poi verrà utilizzata da tutte le classi del package database.
Un'altro esempio di tale pattern è la classe User, può esistere una sola istanza di user in un dato momento. Tale classe viene istanziata tramite il login/ registrazione e viene posta a null tramite la chiamata a logout.

MVC:
Il pattern model-view-controller è ampiamente utilizzato all'interno del progetto per separare al meglio la logica di funzionamento del programma dalla logica di gestione dell'interfaccia con l'utente.

Model: le classi che si trovano nel package dataservice forniscono i metodi per accedere ai dati utili all'applicativo
View: la visualizzazione dei dati avviene grazie ai file all'interno della directory resources. Sono file ".fxml" e permettono la visualizzazione dei dati e l'interazione con l'utente.
Controller: le classi all'interno del package controller fanno da tramite tra view e model passando i dati inseriti/richiesti dall'utente.

Ogni file '.fxml' ha un proprio controller. Il controller si occupa poi di interagire con le classi nel package dataservice le quali si lavoreranno con i dati.

DAO

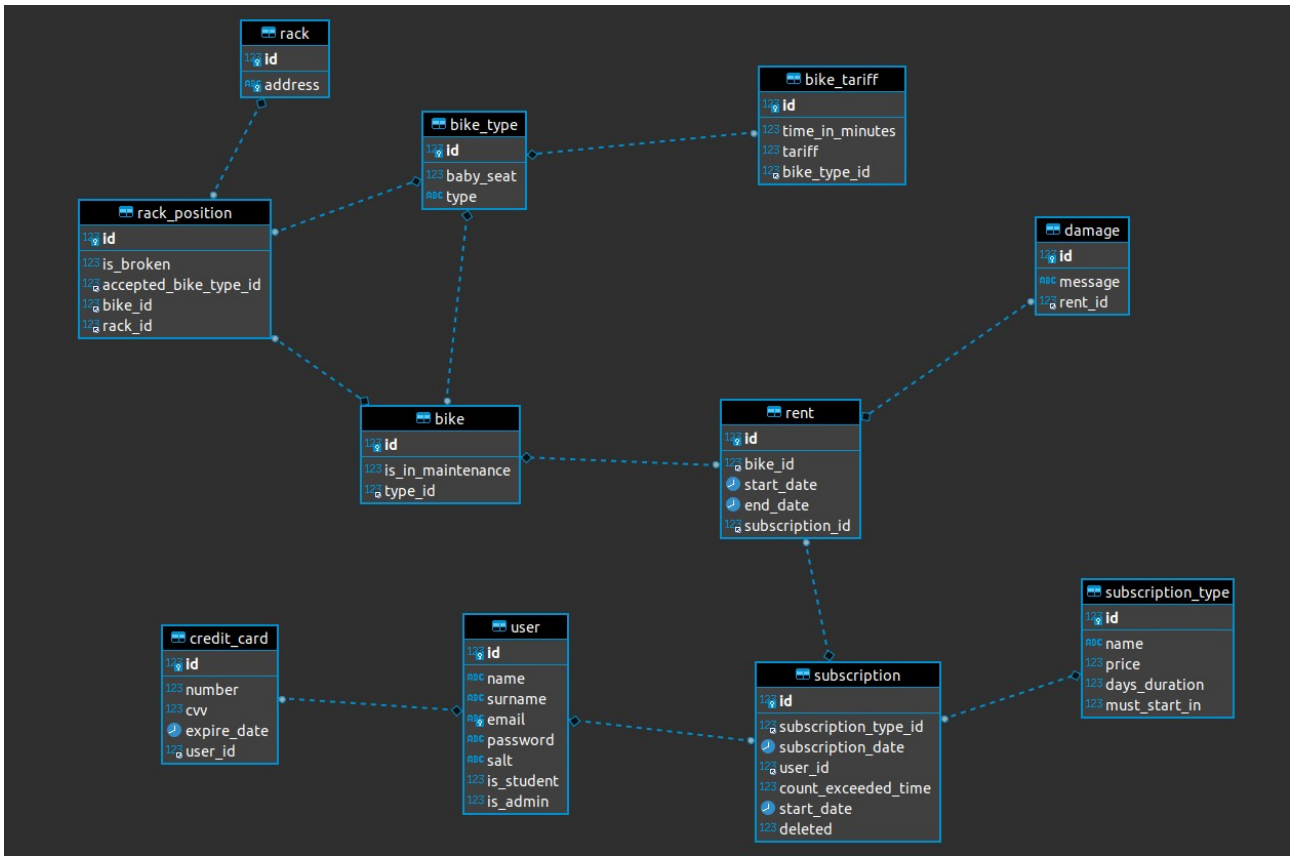
Il pattern DAO è stato utilizzato per implementare le classi all'interno del package database per la memorizzazione dei dati persistenti. Tale pattern è molto importante perché ci permette in un momento futuro, qualora dovesse cambiare tipo di memoria persistente o un differente dbms, basterà implementare delle nuove classi che rispettino le interfacce create.
Tale pattern lo possiamo notare nel diagramma delle classi all'interno del package database, le interfacce hanno il nome "nomeClasseInterface" mentre le classi che implementano tale interfaccia saranno nominate: "nomeClasse".

DTO

Il dto pattern è stato necessario implementarlo all'interno del package controller per permettere la visualizzazione di alcuni dati. Tramite questo pattern abbiamo la possibilità di creare diverse viste di un modello. Le classi dto non contengono alcuna logica di business.
Un'esempio è la classe SubscriptionDTO la quale prende in pasto una subscription e la rielabora per far si che vengano visualizzati solo i dati richiesti.
Altre classi dto si possono ritrovare nelle classi di controller per la visualizzazione di alcune tabelle di javafx.

3.2 Gestione dei dati persistenti

Per la gestione dei dati persistenti è stato utilizzato un un database relazionale MySQL. Qui viene presentato lo schema della base di dati:



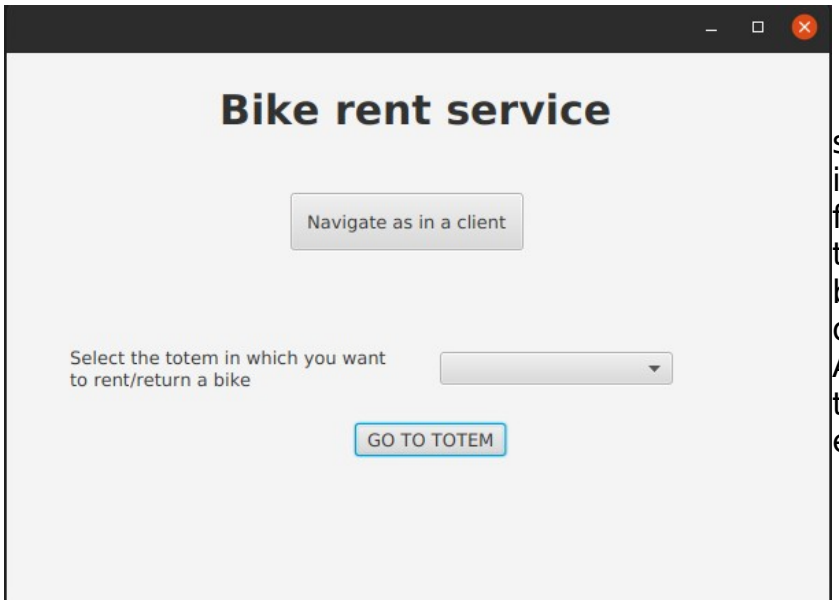
Di seguito una specifica per ogni tabella:
- **Credit_card**: tale tabella si riferisce alle carte di credito di un'utente
- **User**: dati essenziali di un utente. Un'utente con privilegi admin è l'utente che ha all'interno del suo record il campo is_admin = true.
- **Subscription**: abbonamenti creati da un utente. Importante da specificare il campo 'delete' posto a true dopo che count_exceeded_time = 3. Quest'ultimo

campo serve per tener conto di quante volte l'utente ha superato il limite di noleggio.
- **Subscription_type**: tipologie di abbonamenti. È stato scelto di utilizzare una tabella per mantenere le tipologie di abbonamento perché se in un secondo momento si volessero aggiungere di nuove/eliminarne basta modificare il dabatase e il codice del sistema continua a funzionare correttamente. Questo permette un'alta mantenibilità e modificabilità del sistema.
La tabella subscription-type contiene due campi che è meglio specificare il loro significato:
  - days-duration: specifica la durata dell'abbonamento. Tale campo è importante per permettere modifiche future agli abbonamenti
  - must_start_in: specifica entro quando l'abbonamento deve iniziare. Tale campo è fondamentale per permettere di capire se una carta di credito è valida o meno. Dalle specifiche, la carta di credito, deve essere valida per tutta la durata dell'abbonamento. Esistono tuttavia degli abbonamenti che iniziano al primo prelievo della bicicletta e quindi sostanzialmente sarebbero di durata infinita. Questo non permetterebbe però il controllo sulla carta di credito. È stato quindi deciso di mettere un limite di giorni entro il quale l'abbonamento deve iniziare (impostato a 90 gg).
- **Rent**: tabella per il salvataggio di ogni noleggio.
- **Damage**: tabella per salvare un eventuale messaggio di danno relativo ad un noleggio di una bicicletta.
- **Bike**: tabella per il salvataggio di tutte le biciclette.
- **Bike_Type**: tabella per mantenere salvate le tipologie delle biciclette. Anche qua è stato utilizzato il database in modo tale che se dovessero essere aggiunte delle biciclette nuove bisogna toccare in modo lieve il codice e aggiornare la base di dati.
- **Bike_tariff**: è la tabella utilizzata per mantenere il tariffario del costo delle biciclette. IL calcolo delle tariffe avviene secondo un criterio a scatto. Nella colonna time_in_minutes sono specificati i minuti, se il tempo del noleggio è minore di tale valore si paga tale somma. Contrariamente se fosse superiore si somma il valore della tariffa a quello dello slot successivo(rifacendo lo stesso controllo). In questo modo se dovesse essere deciso di modificare delle tariffe o slot orari è necessaria solo una modifica di questa tabella.
- **Rack_Position**: tabella che indica ogni morsa di una rastrelliera. Per ogni morsa abbiamo un tipo di bici accettata e un campo che indica se è utilizzabile o meno (is_broken).
- **Rack**: tabella che indica la rastrelliera.

All'interno del sistema abbiamo un dato importante che non può essere salvato in chiaro, la password dell'utente. Per questo viene utilizzata la classe PasswordUtils come libreria per generare partendo dalla password inserita una nuova password crittografata, che sarà poi leggibile grazie all'utilizzo di una chiave. In questo modo la password, per chiunque la legga da database, non è decifrabile.

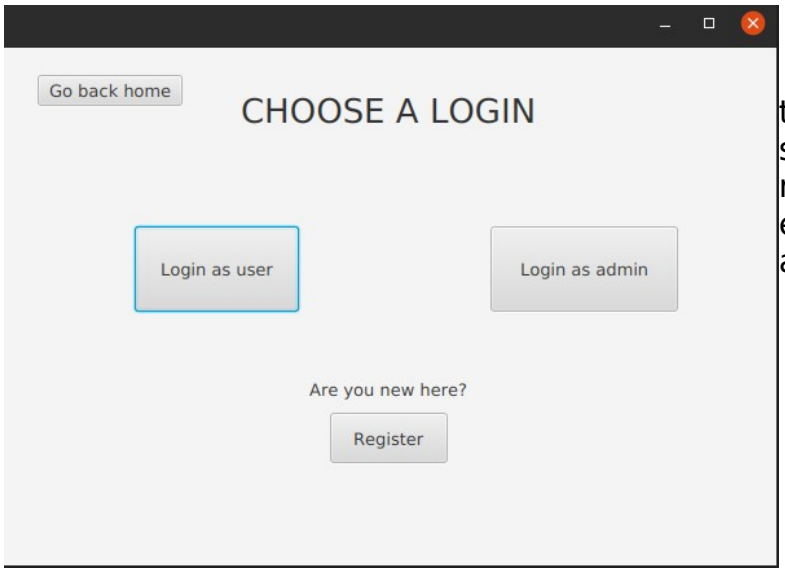
Inoltre per una sicurezza nell'esecuzione delle query viene utilizzata l'istruzione preparedStatement per proteggersi da eventuali attacchi esterni con sql injection o altri possibili attacchi.

3.3 Descrizione dell'Interfaccia Grafica

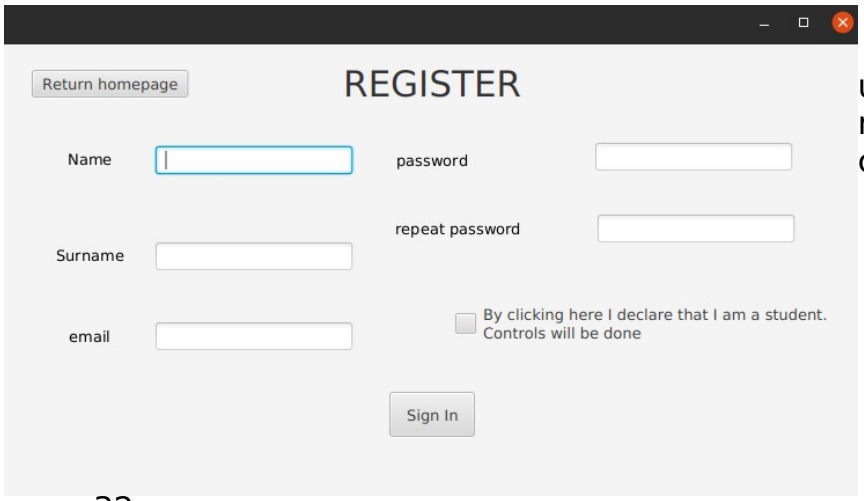


La prima pagina che si incontra è la seguente. In realtà vuole indicare ipoteticamente dove si trova fisicamente l'utente. Qualora si trovasse davanti ad un totem allora bisogna scegliere dalla casella a discesa e cliccare su 'go to totem'. Al contrario se l'utente non si trovasse davanti ad un totem può entrare nel sito attraverso un client.

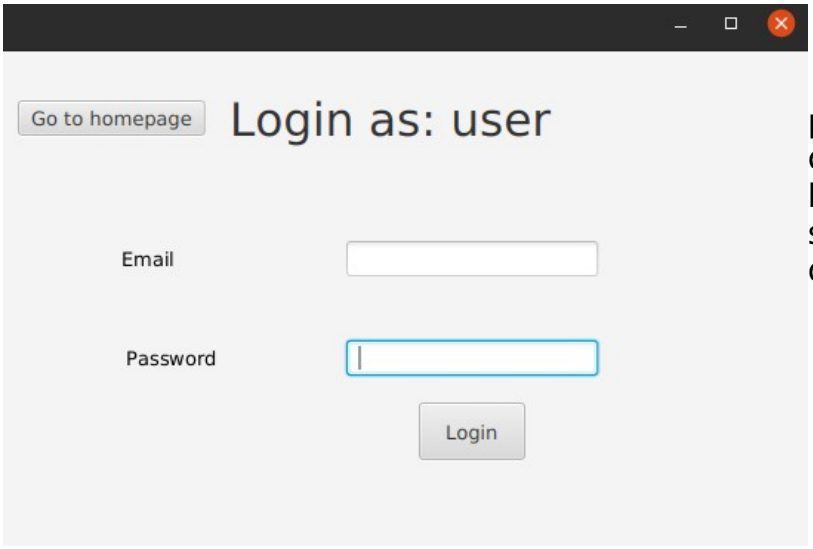
NAVIGATE AS A CLIENT



cliccando su navigate as a client ci troviamo davanti a questa schermata. In questo caso viene richiesto all'utente l'azione che vuole eseguire tra login come utente, come admin oppure registrarsi al sistema.

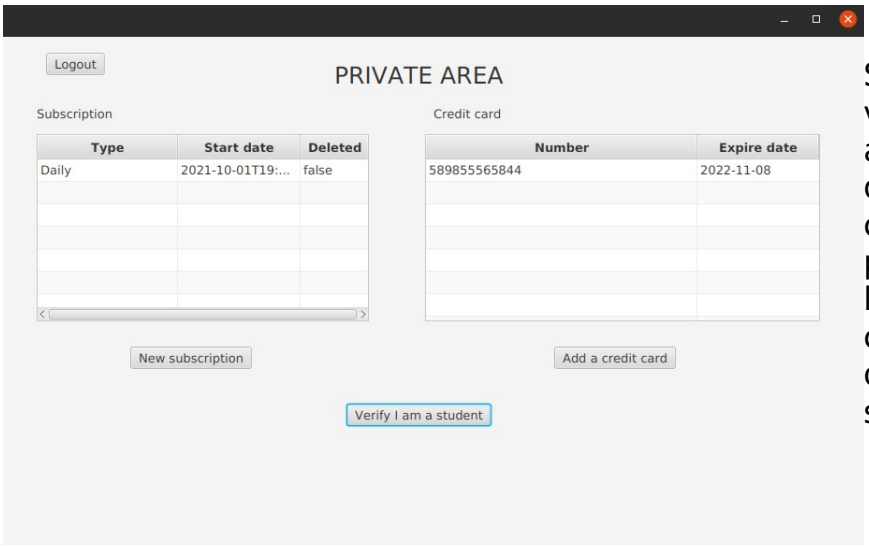


Pannello di registrazione di un utente. Tutti i campi sono richiesti e vengono controllati al click sul bottone Sign in

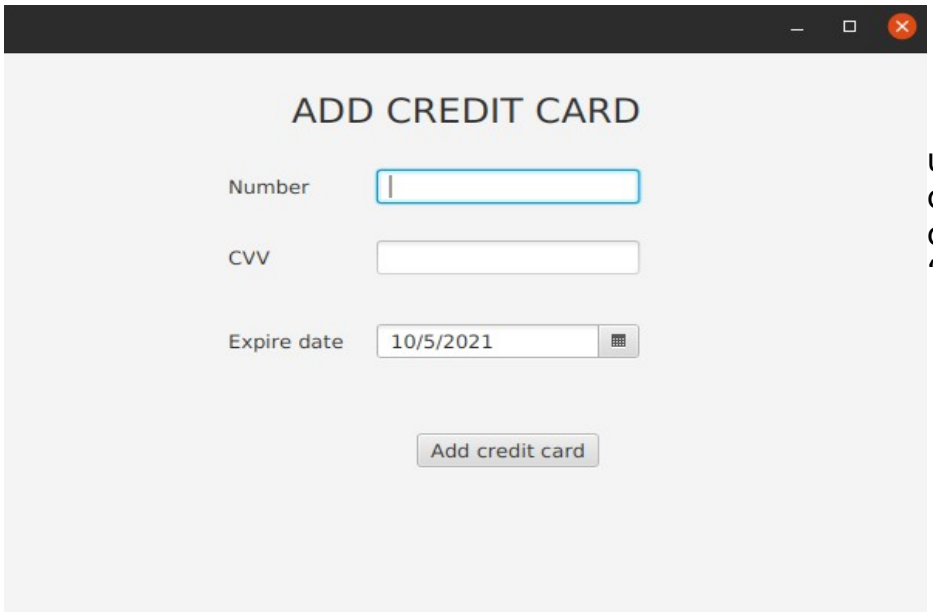


pannello di login, tutti i campi sono obbligatori e controllati al click sul bottone ‘login’. Qualora fosse stato scelto il login as admin la label diventerebbe ‘Login as: admin’

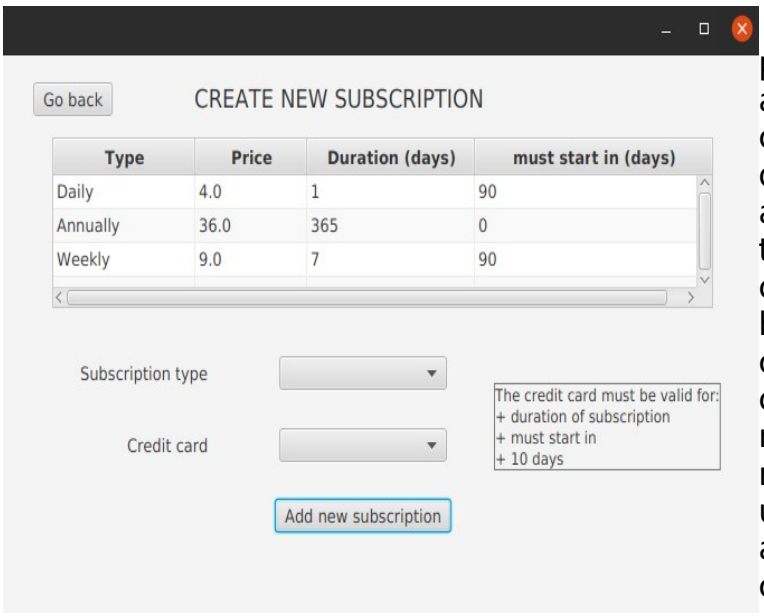
LOGIN AS USER



Sulla sinistra di questo pannello vediamo una lista degli abbonamenti dell’utente, a destra invece una lista delle sue carte di credito. In questo pannello, tramite i 3 bottoni, l’utente può aggiungere una carta di credito, un’abbonamento oppure verificare il suo status di studente

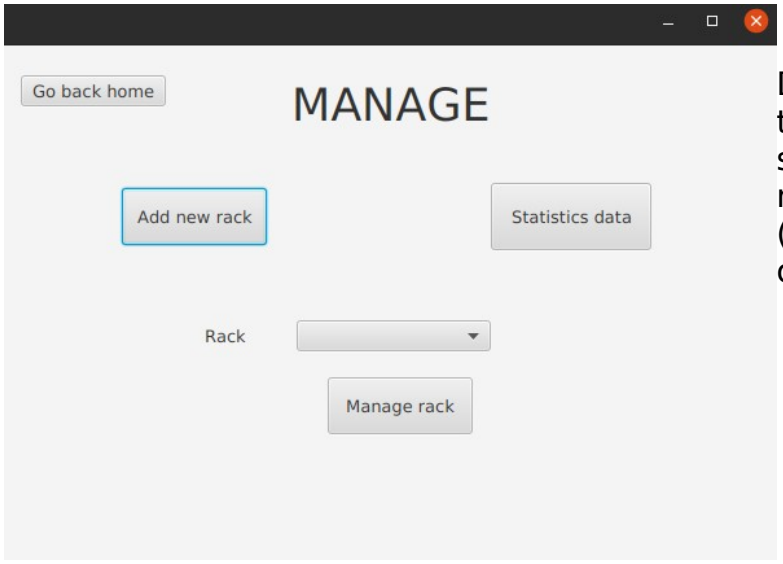


pannello per l’aggiunta di una carta di credito, tutti i campi sono obbligatori e controllati al click del bottone ‘add credit card’

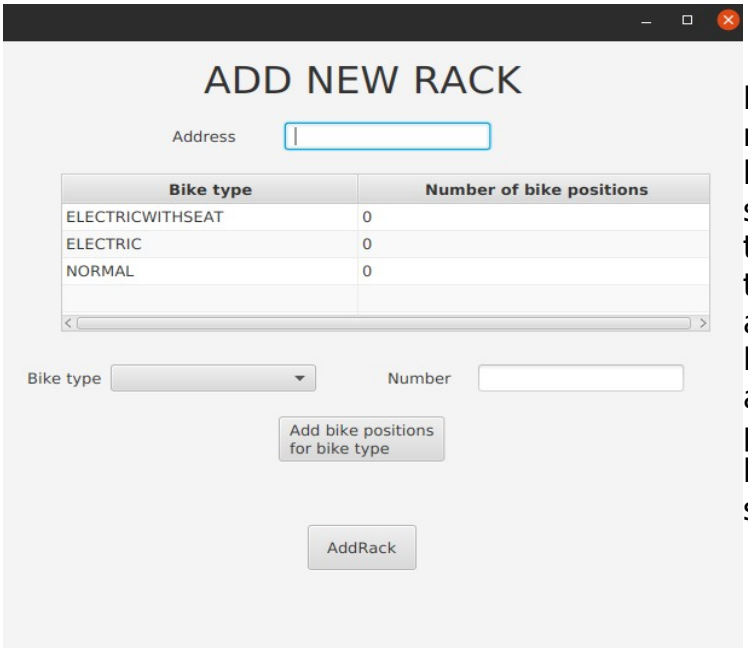


pannello per l'aggiunta di un nuovo abbonamento. In alto c'è una tabella che mostra i tipi di abbonamento disponibili. Per aggiungere un nuovo abbonamento bisogna scegliere un tipo di abbonamento e una carta di credito da utilizzare. Al click sul bottone di aggiunta vengono fatti i controlli necessari sulla carta di credito. Non viene ritornato in questo momento alcun codice univoco. In realtà l'abbonamento ha un codice univoco (id)che però rimane implicito all'utente. Tale codice viene ritrovato con l'autenticazione dell'utente.

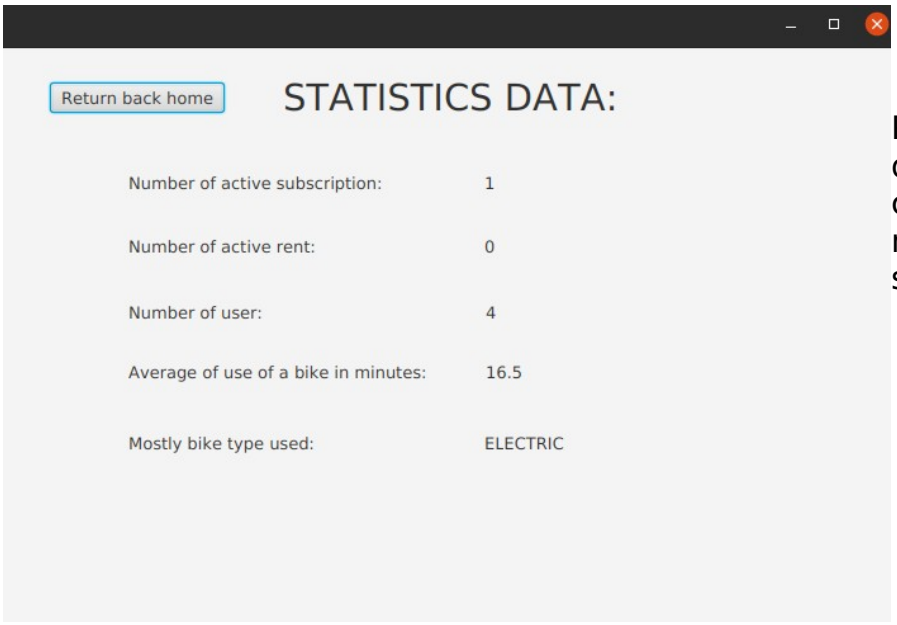
LOGIN AS ADMIN



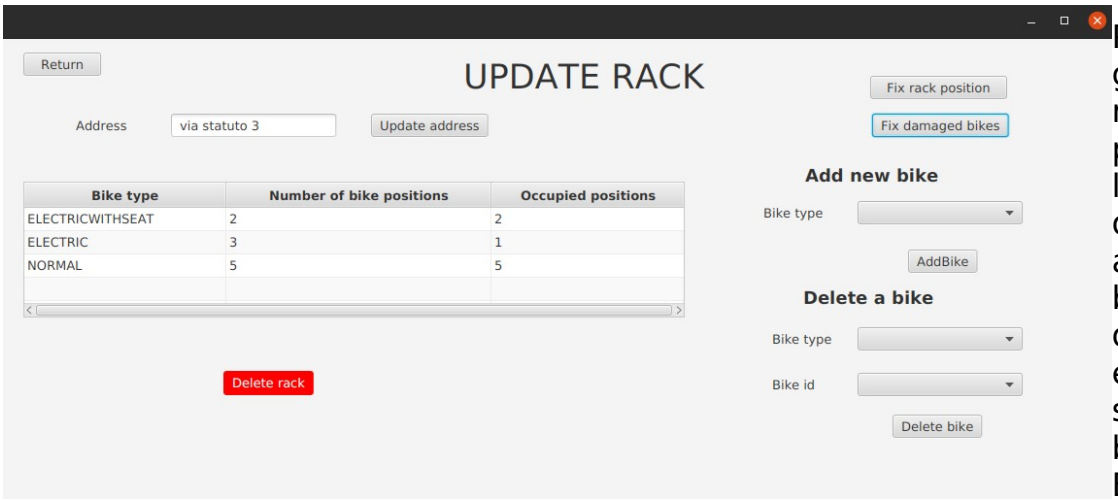
Dopo esserci loggati come admin troviamo il pannello di gestione del sistema. L'admin può aggiungere una nuova rastrelliera, gestire una esistente (scegliendola dalla casella a discesa) oppure consultare i dati statistici.



Pannello di aggiunta di una nuova rastrelliera. Bisogna aggiungere l'indirizzo di tale rastrelliera e successivamente indicare per ogni tipo di morsa quante ce ne sono in tale rastrelliera. Infine cliccare su addrack per crearla.
La rastrelliera viene creata senza aggiungere le bici perché è stato pensato come l'operatore che registra la nuova rastrelliera e poi successivamente aggiunge le bici.

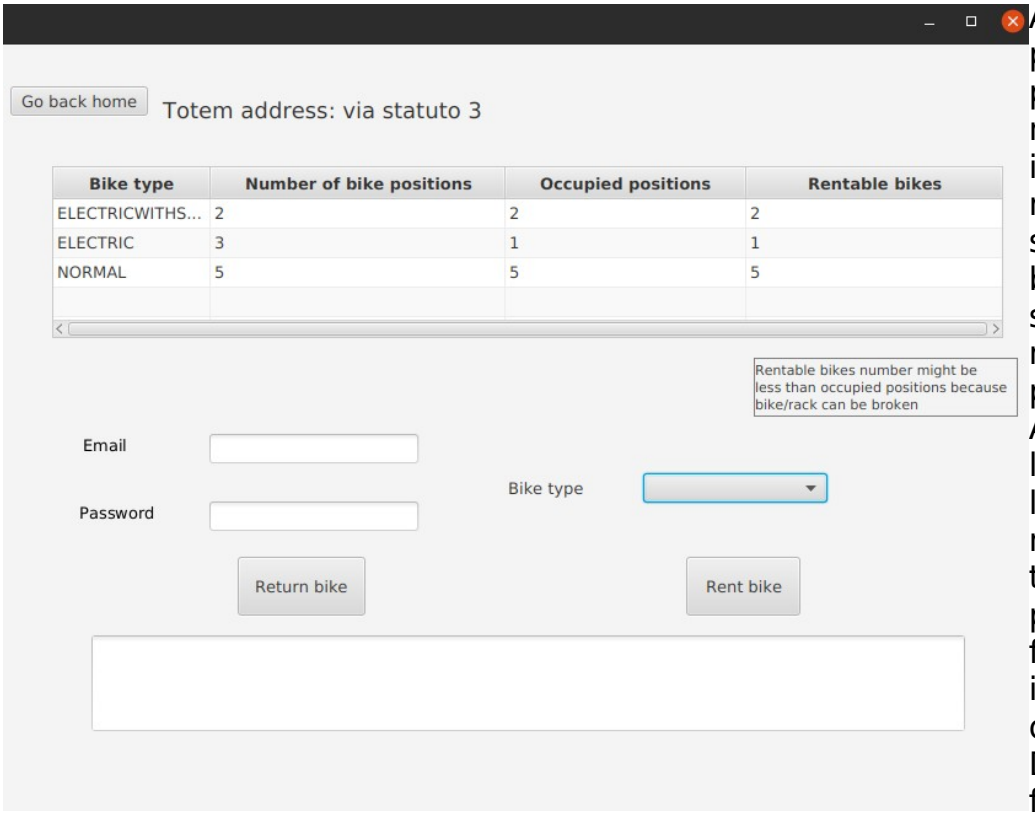


Pannello di amministrazione dove possono essere consultati tutti i dati statistici messi a disposizione dal sistema.



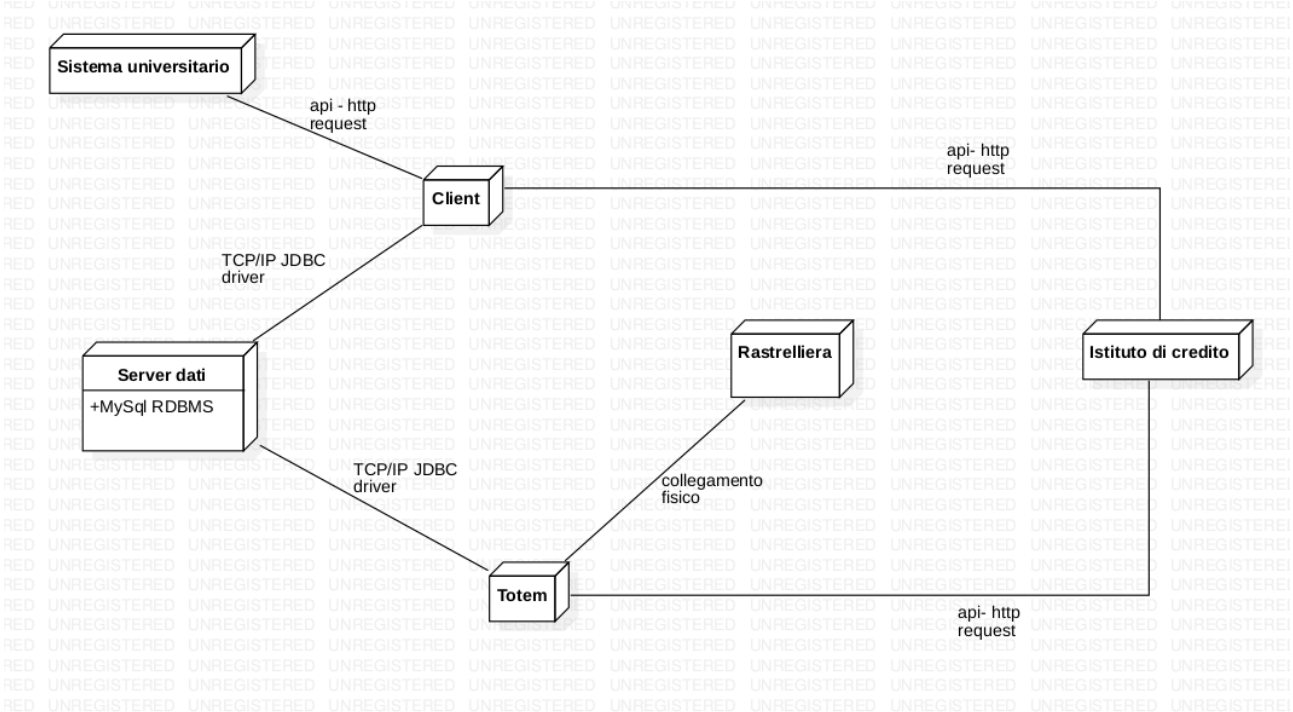
Pannello di gestione di una rastrelliera. Si può modificare l'indirizzo alla quale si trova, aggiungere biciclette di un dato tipo oppure elimiare una specifica bicicletta.
È inoltre possibile eliminare una rastrelliera dal sistema, al click del bottone viene controllato che la rastrelliera non contenga alcuna bici. È inoltre possibile sistemare eventuali biciclette o morse rotte.

GO TO TOTEM



Attraverso questo pannello l'utente ha la possibilità di iniziare un nuovo noleggio inserendo la propria mail e password e scegliendo il tipo di bicicletta desiderato. Il sistema indicherà poi la morsa dalla quale prendere la bicicletta. Alla fine del noleggio l'utente dovrà lasciare la bicicletta in una morsa, avvicinarsi al totem, inserire email e password e il sistema, fatti i dovuti controlli, informa il costo totale dell'abbonamento. Durante la procedura di fine noleggio l'utente può anche comunicare dei danni alla bicicletta, tale form si apre al click sul bottone di return bike.

3.4 Diagramma di deployment



Possiamo notare diversi nodi computazionali:

- sistema universitario: rappresenta un nodo esterno con il quale il sistema si interfaccia tramite api necessario per validare la richiesta di un utente di autenticarsi come studente.
- Istituto di credito: rappresenta un nodo esterno con il quale il sistema si interfaccia tramite api, necessario per validare le carte di credito e processare i pagamenti.
- Client: nodo con il quale l'utente interagisce per creare abbonamenti, aggiungere carte di credito o per la gestione generale da parte dell'admin.
- Rastrelliera: nodo hardware utilizzato tramite collegamento fisico che si occupa di gestire le rastrelliere e le morse
- Totem: nodo che rappresenta il totem collegato fisicamente alla rastrelliera necessario per il noleggio
- Server dati: nodo centrale al quale sono collegati tutti i totem e i client contente tutti i dati persistenti necessari per il funzionamento corretto del sistema.

3.5 Specifica e verifica dei vincoli

La documentazione dei setter e dei getter è stata omessa per brevità.
Data l'incopatibilità di JML con le versioni recenti di Java nel codice non è presente la documentazione.

```
Context Bike
inv: self.id > 0
inv: self.type → not null
inv: self.isInMaintenance → not null

/*
    invariant id > 0 && type != null && isInMaintenance != null
*/

Context Bike:: setBikeInMaintenance
pre: self → not null
post: self.isInMaintenance = true

/*
    requires: this != null
    ensures: isInMaintenance == true
*/

Context Bike:: fixBikeInMaintenance
pre:  self → not null
post: self.isInMaintenance = false

/*
    requires: this != null
    ensures: isInMaintenance == false
*/
```

Context BikeType
inv: self.id > 0
inv: self.type = BikeTypeEnum::NORMAL || self.type =
BikeTypeEnum::ELECTRIC ||self.type = BikeTypeEnum::ELECTRICWITHSEAT
inv: self.babySeat → not null

Context CreditCard
inv: self.id > 0
inv: self.number > 0
inv: self.cvv > 0
inv: self.expireDate → not null

/*
    invariant id >0 && number >0 && cvv >0 && expireDate != null
*/

Context DamageMessage
inv: self.id > 0
inv: self.message → not null

/*
   invariant id >0 && message != null
*/

Context RackPosition
inv: self.id > 0
inv: isBroken → not null
inv: acceptedBikeType → not null

/*
  invariant id > 0 && isBroken != null && acceptedBike != null
*/

Context RackPosition::unlock
pre: self.bike → not null
post: self.bike = null

/*
   requires: bike != null
   ensures: bike = null
*/

Context RackPosition::lock(bike)
pre: self.bike = null
pre: bike.type = self.acceptedBikeType
post: self.bike → not null

/*

```
    requires: this.bike != null && bike.getType() == this.acceptedBikeType
    ensures: this.bike = null
*/

Context Rent
inv:self.id > 0
inv: self.bike → not null
inv: self.startDate → not null
inv: self.startDate < self.endDate

/*
   invariant id > 0 && bike != null && startDate != null && startDate<endDate
*/

Context Rent::endRent
pre: self → not null
post: self.endDate = today

/*
   requires: this != null
   ensures: this.endDate = now()
*/

Context Subscription
inv: self.id > 0
inv: self.type → not null
inv: self.subscriptionDate → not null
inv: self.user → not null
inv: 0<= self.countExceededTime <=3
inv: deleted → not null
inv: self.type = “Annually” implies self.startDate = self.subscriptionDate
inv: self.countExceededTime = 3 implies deleted = true

/*
  invariant id >0 && type != null && subscriptionDate != null && user != null
        && countExceededTime >= 0 && countExceededTime <= 3 && deleted !
        = null
  invariant type = “Annually” ==> startDate = subscriptionDate
  invariant countExceededTime = 3 ==> deleted = true
*/

Context Subscription::startSubscriptionNow()
pre: self → not null
post: self.startDate = today

/*
   requires: this != null
   ensures: this.startDate = now()
*/
```

```
Context SubscriptionType
inv: self.id > 0
inv: self.name → not null
inv: self.price >= 0
inv: self.daysDuration > 0
inv: self.mustStartIn >= 0

/*
  invariant id > 0 && name != null && price >= 0 && daysDuration > 0 &&
       mustStartIn >= 0
*/

Context Tariff
inv: self.id > 0
inv: self.passedTimeInMinutes >= 0
inv: self.tariff >= 0
inv: self.bikeType → not null

/*
  invariant id > 0 && passedTimeInMinutes >= 0 && tariff >0 && bikeType !=
       null
*/

Context TotemRack
inv: id > 0
inv: address → not null
inv: rackList → not null

/*
  invariant id>0 && address != null && rackList != null
*/

Context UserGeneric
inv: self.id > 0
inv: self.name → not null
inv: self.surname → not null
inv: Self.email → not null
inv: self.isStudent → not null

/*
  invariant id> 0&& name != null && surname != null && email != null &&
       isStudent != null
*/
```

3.6 Descrizione del testing

Il sistema è dotato di unit test su unità funzionali testabili in quanto staccate totalmente da tutte gli altri metodi.
I test attuati controllano che la logica implementata sia adatta e servono noltre per future modifiche, per controllare che una modifica non vada a impattare negativamente sul resto del sistema.

I metodi testati sono:
- isCreditCardValid della classe CreditCard
- isCreditCardValidForSubscription della classe CreditCard
- isBikeTypeRentable della classe RackPosition
- isFreeAndAccessibleForBikeType della classe RackPosition
- isThereAnActiveRent della classe Subscription
- isValid della classe Subscription
- calculateTariffByBikeType della classe Tariff
- arePassedMinutesFromRent della classe Rent
- lastUserRent della classe User

Sono stati utilizzati dei criteri di copertura di branch coverage, copertura delle decisioni e delle condizioni.

3.7 Note per l'installazione e l'utilizzo

Per la compilazione del codice è consigliato l'utilizzo di maven dato che ci sono delle librerie esterne utilizzate. Per eseguire il software bisogna:

- importare il database e nel caso cambiasse nome o altre configurazioni (per esempio la porta) modificare la stringa nella classe database nel package database.
- Da riga di comando spostarsi nella directory del progetto e inserire il comando: "mvn clean javafx:run".

Degli account di test sono:

- utente con privilegi admin
  - **email**:admin@gmail.com
  - **password**: admin
- utente senza privilegi admin
  - **email**:uba99a@gmail.com
  -**password**:password

Le credenziali per il database sono:
- **nome**: bikeProject
- **password**: 8bikeProject8*
- **user**: bikeProject

Per compiere dei test è possibile modificare dei dati all'interno del file config.json. Per esempio se si volesse che il sistema universitario risponda negativo alla richiesta di una validazione come status studente basta porre il valore di universityMockResponse = false e il sistema prenderà in automatico tale risposta.