

TEMPLATE STRUTTURA RELAZIONE

1 Descrizione del problema

1.1 Analisi e specifica dei requisiti

L'obiettivo è quello di realizzare un'applicazione per bike sharing.

Andiamo a vedere chi compone il nostro sistema:

Ovviamente avremo le **biciclette** che possono distinguersi in tre categorie: elettrice, elettriche con seggiolino e normali. Esse sono posizionate dal personale (della quale tratteremo più avanti) nelle **rastrelliere**. Ogni stazione è costituita da una rastrelliera (composta da **morse** (di tre tipi come le biciclette). Ad' ogni stazione è presente un **totem** con la quale gli **utenti** possono interagire per prelevare/depositare una bicicletta.

Riassumendo quindi, nel mio progetto il concetto di stazione è rappresentato come l'unione di una rastrelliera composta da N morse e un totem associato ad essi.

Gli **utenti** sono dotati di una applicazione mobile dalla quale è possibile registrarsi e ottenere un **abbonamento** che può essere di tre tipi: giornaliero, settimanale o annuale. Una volta ottenuto un abbonamento (l'utente inserisce i propri dati personali, una password, tipo di abb. da acquistare e i dati della sua carta di credito e riceve un codice utente da usare per i futuri login), un utente può prelevare una bicicletta (fino alla scadenza del suo abbonamento). Se un utente restituisce una bicicletta dopo 2h dal ritiro viene aggiunta un'ammonizione al suo abbonamento. Alla terza ammonizione l'abbonamento viene annullato. Inoltre non è possibile ritirare una nuova bicicletta per 5 minuti dopo l'ultimo prelievo. Se un utente restituisce una bicicletta dopo 24h viene addebitata una penale di 150€ più i normali costi di utilizzo sulla sua carta.

Gli abbonamenti sono associati ad una **carta di credito** dalla quale vengo scalati i soldi per l'attivazione dell'abbonamento e dopo ogni **prelievo** seguendo un opportuno listino prezzi basato sul tipo di bicicletta e sul minutaggio di utilizzo. Un prelievo viene creato quando un utente ritira una bicicletta e viene chiuso quando esso la deposita. Mantenere la traccia dei prelievi e della loro durata consente di calcolare la spesa da attribuire all'utente per l'utilizzo della bicicletta.

I **totem** sono i punti chiave con la quale un utente interagisce per la "gestione delle biciclette". L'utente per il prelievo/deposito inserirà le credenziali nel totem. Nel caso del ritiro dovrà poi anche indicare quale tipo di bicicletta esso vuole prelevare e se è presente essa verrà tolta dalla rastrelliera associata al totem in questione.

Il **personale** ha accesso ad un terminale, dal quale possono visualizzare la lista di stazioni e di biciclette presenti nel db. Possono aggiungere, togliere o spostare biciclette, aggiungere o togliere intere stazioni dal sistema

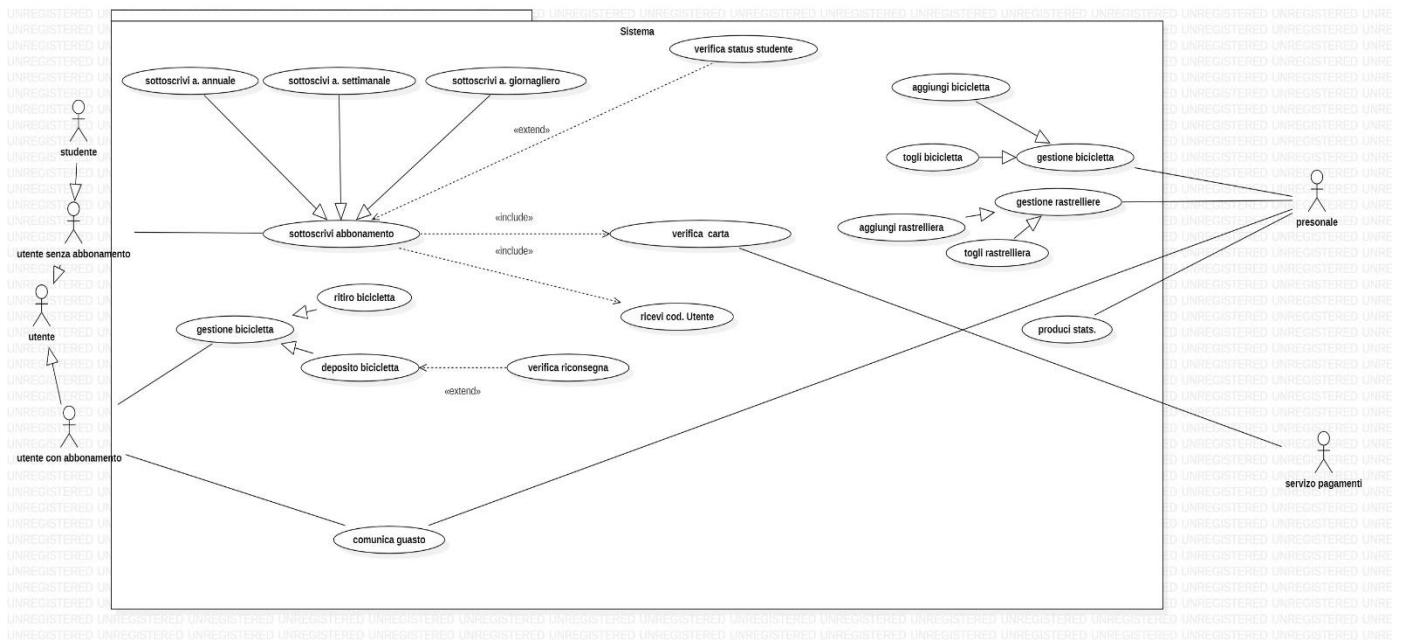
Il personale ha anche accesso a dei dati statistici sull'utilizzo delle biciclette e altro ancora.

Per completezza riporto il listino prezzi delle biciclette.

	Primi 30 min (o studenti)	Secondi 30 min	Terzi 30 min	Quarti 30 min	Ore successive
NORMALE	0.00€	0.50€	0.50€	0.50€	2.00€
ELETTRICA	0.25€	0.50€	1,00€	2.00€	4.00€
E. CON SEGGIOLINO	0.25€	0.50€	1.00€	2.00€	4.00€

2 Progettazione del Sistema

2.1 Diagramma dei casi d'uso



Registrazione utente: l'utente sottoscrive un abbonamento inserendo nome, cognome, username, dati della carta di credito e password.

Registrazione studenti: l'utente può indicare di essere uno studente. Nel mio progetto non verranno fatte delle verifiche sulla veridicità dall'informazione: verrà presa per buona.

Ritiro bicicletta: l'utente inserisce sul totem il proprio codice abbonamento fornito dal sistema durante la registrazione e la propria password, specificando quale tipologia di bici vuole utilizzare. Il sistema indica il numero di posteggio della bicicletta da prelevare. Se non ci sono biciclette di quel tipo disponibili sulla rastrelliera, lo comunica all'utente.

Pagamento: quando la bici viene restituita, viene calcolata la spesa totale dato il tempo di utilizzo e il tipo di bicicletta utilizzata.

Restituzione bicicletta: l'utente inserisce il codice utente e la password sul totem. Se non vi sono morse libere per quel tipo di bicicletta sulla rastrelliera il deposito non viene effettuato. Altrimenti il sistema conferma la restituzione della bicicletta e indica all'utente quanto ha speso per il prelievo appena concluso.

Segnalazione malfunzionamento: l'utente, tramite il totem può indicare se ha riscontrato un danno alla bicicletta. Il danno viene salvato nel db così che un manutentore in un secondo momento potrà verificare l'entità del danno e aggiustare la bici (parte non richiesta dalla specifica e non implementata)

Aggiunta/rimozione rastrelliere: il personale del comune può aggiungere o rimuovere una o più rastrelliere.

Aggiunta/rimozione biciclette: il personale del comune può aggiungere o rimuovere una o più bici.

Visualizzazione dati statistici: il sistema produce delle stats su bici, stazioni ecc...

2.2 Descrizione degli scenari

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

Nome	Registrazione abbonamento
Scopo	Far ottenere ad un nuovo utente un abbonamento
Attore/i	Utente senza abbonamento, servizio pagamenti
Pre-condizioni	
Trigger	
Descrizione sequenza eventi	<ol style="list-style-type: none">1. L'utente inserisce nome e cognome2. L'utente seleziona la password del suo nuovo abbonamento3. L'utente sceglie il tipo di abbonamento da acquistare4. L'utente inserisce i dati della propria carta di credito che verrà associata all'abbonamento5. Il servizio pagamenti controlla la validità della carta (controlla semplicemente che il numero sia di 16 cifre, che il codice di sicurezza sia di 3 e che la scadenza sia valida)6. Viene fornito un codice utente allo user e le informazioni riguardanti nuovo utente, abbonamento associato e carta associata vengono salvati
Alternativa/e	1b. Password non valida (deve esserci almeno una cifra e una lettera maiuscola)

	<p>2b. Scelta abbonamento non valida</p> <p>4b. Il servizio pagamenti informa che la carta non è valida</p>
Post-condizioni	L'utente, nel caso operazione riuscita, potrà utilizzare il suo nuovo abbonamento e di conseguenza utilizzare le biciclette.

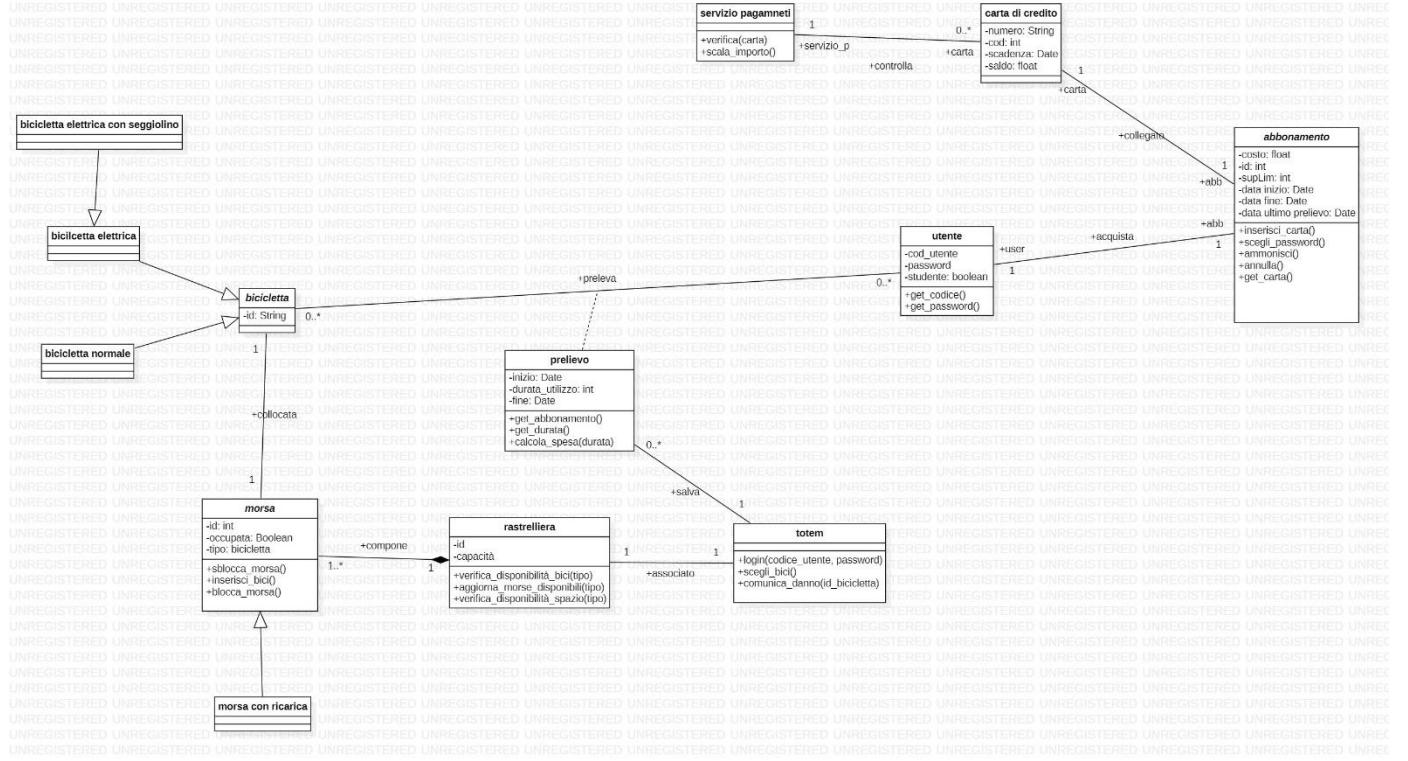
Nome	Ritiro Bici
Scopo	Un utente registrato preleva una bici
Attore/i	Utente con abbonamento
Pre-condizioni	L'utente deve avere un abbonamento
Trigger	
Descrizione sequenza eventi	<ol style="list-style-type: none"> 1. L'utente si avvicina al totem e inserisce le proprie credenziali 2. L'utente sceglie la tipologia di bici (e a che stazione si trova) 3. L'utente preleva la bici 4. Se l'abbonamento non era ancora attivo, esso viene attivato automaticamente
Alternativa/e	<p>1b. L'utente inserisce credenziali errate</p> <p>1c. Il totem invita l'utente registrato a re-inserire le credenziali</p> <p>2b. La tipologia di bici scelta dall'utente con abbonamento non è disponibile a quella stazione</p> <p>2c. Il totem invita l'utente registrato a scegliere un altro tipo di bicicletta</p>
Post-condizioni	L'utente ottiene la bicicletta desiderata

Nome	Deposito Bici
Scopo	Un utente registrato deposita una bici
Attore/i	Utente con abbonamento
Pre-condizioni	L'utente deve avere ritirato precedentemente la bici che sta depositando
Trigger	

Descrizione sequenza eventi	<ol style="list-style-type: none"> 1. L'utente inserisce le credenziali 2. Seleziona la stazione in cui si trova 3. Deposita la bicicletta 4. Il prelievo viene concluso 5. Viene addebitata la spesa alla carta del cliente
Alternativa/e	<ol style="list-style-type: none"> 1b. Credenziali errate 3b. Non ci sono slot disponibili per la bici selezionata 5b. Se la carta va “in rosso” l’abbonamento viene annullato
Post-condizioni	E’ disponibile una bici in più nella rastrelliera in questione

Nome	Aggiunta bicicletta a rasstelliera
Scopo	Aggiungere una bici ad una rastrelliera
Attore/i	Personale
Pre-condizioni	Deve esserci almeno un posto libero nella rastrelliera nella quale verrà inserita la nuova bicicletta
Trigger	
Descrizione sequenza eventi	<ol style="list-style-type: none"> 1. Un membro del personale seleziona la bici da spostare 2. Seleziona la stazione target
Alternativa/e	<ol style="list-style-type: none"> 2b. Nella stazione target non ci sono spazi disponibili per quel tipo di bicicletta
Post-condizioni	E’ <u>disponibile</u> una bici in più nella rastrelliera in questione

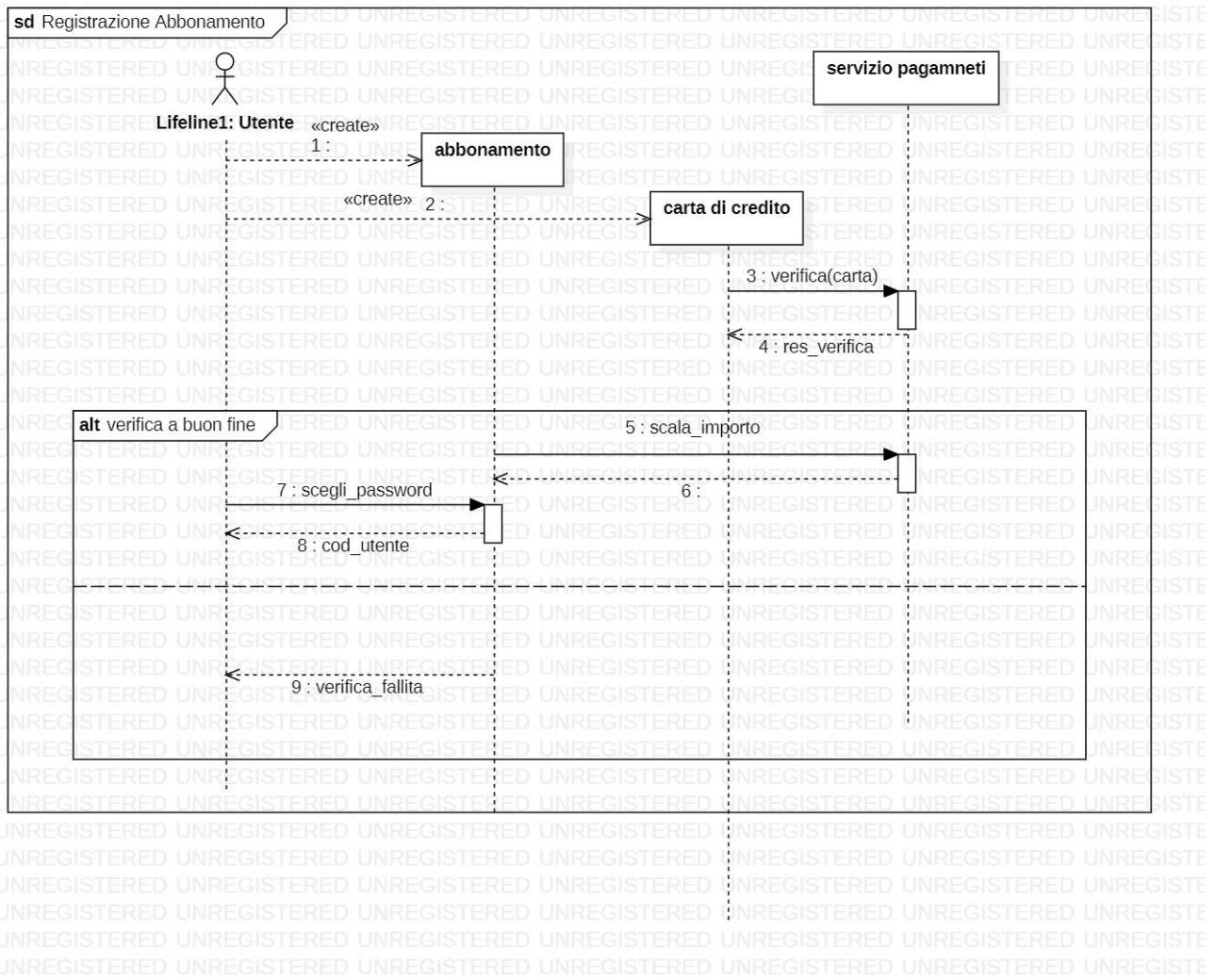
2.3 Diagramma delle classi (modello di progetto)

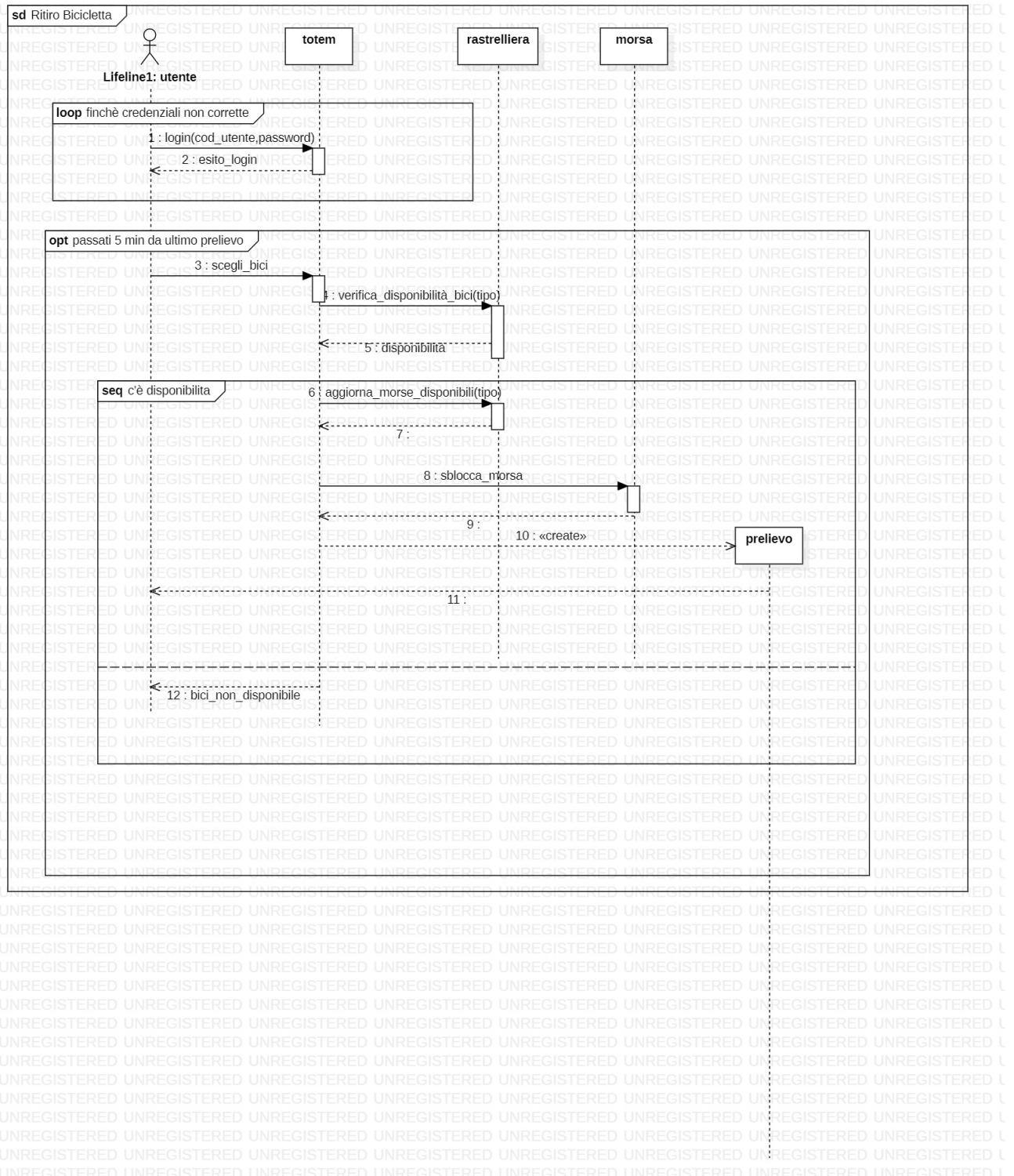


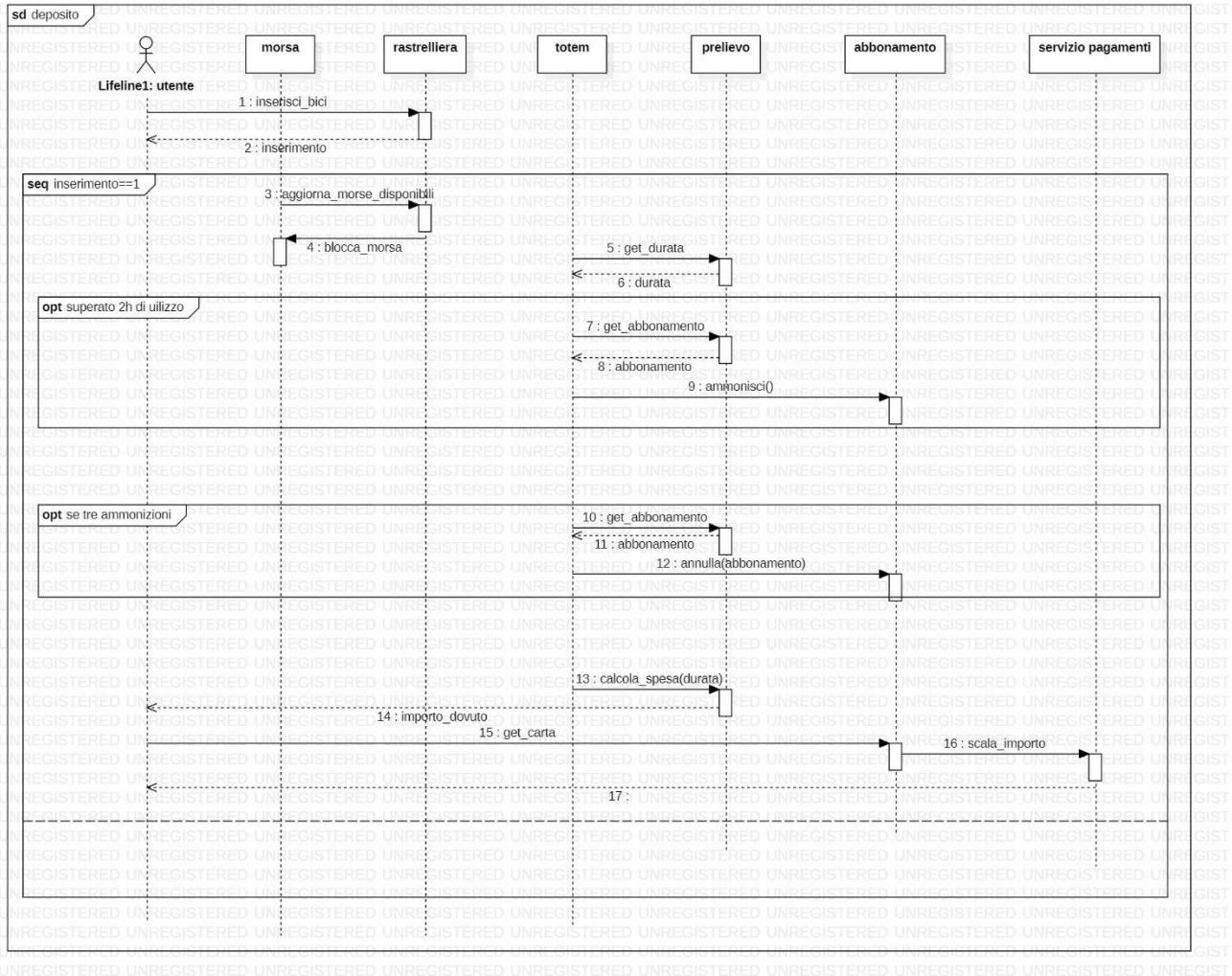
NB: Questo **NON** è il diagramma di programma, perciò non sarà completamente tracciabile con il codice in quanto nel diagramma di programma sono state incluse più classi/metodi.

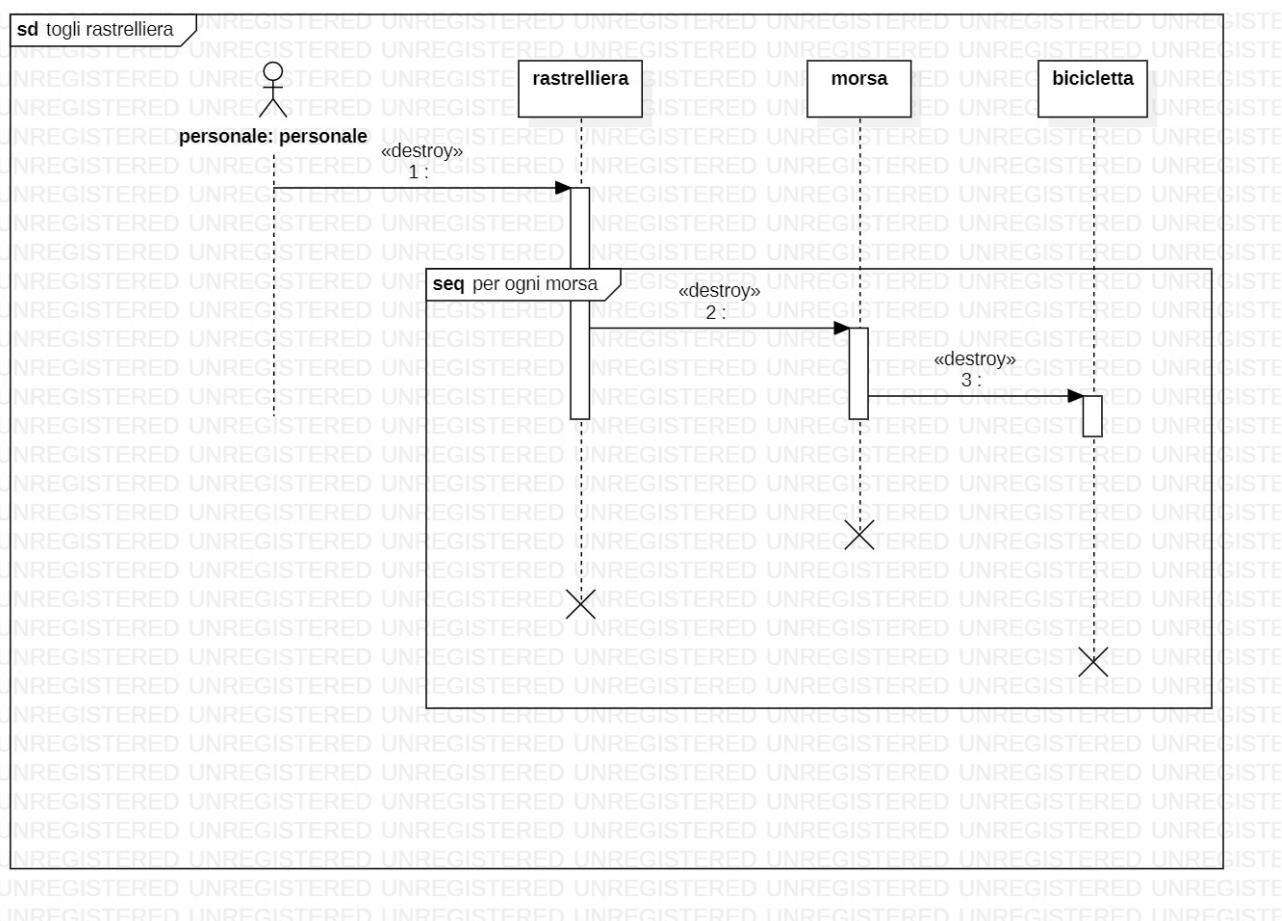
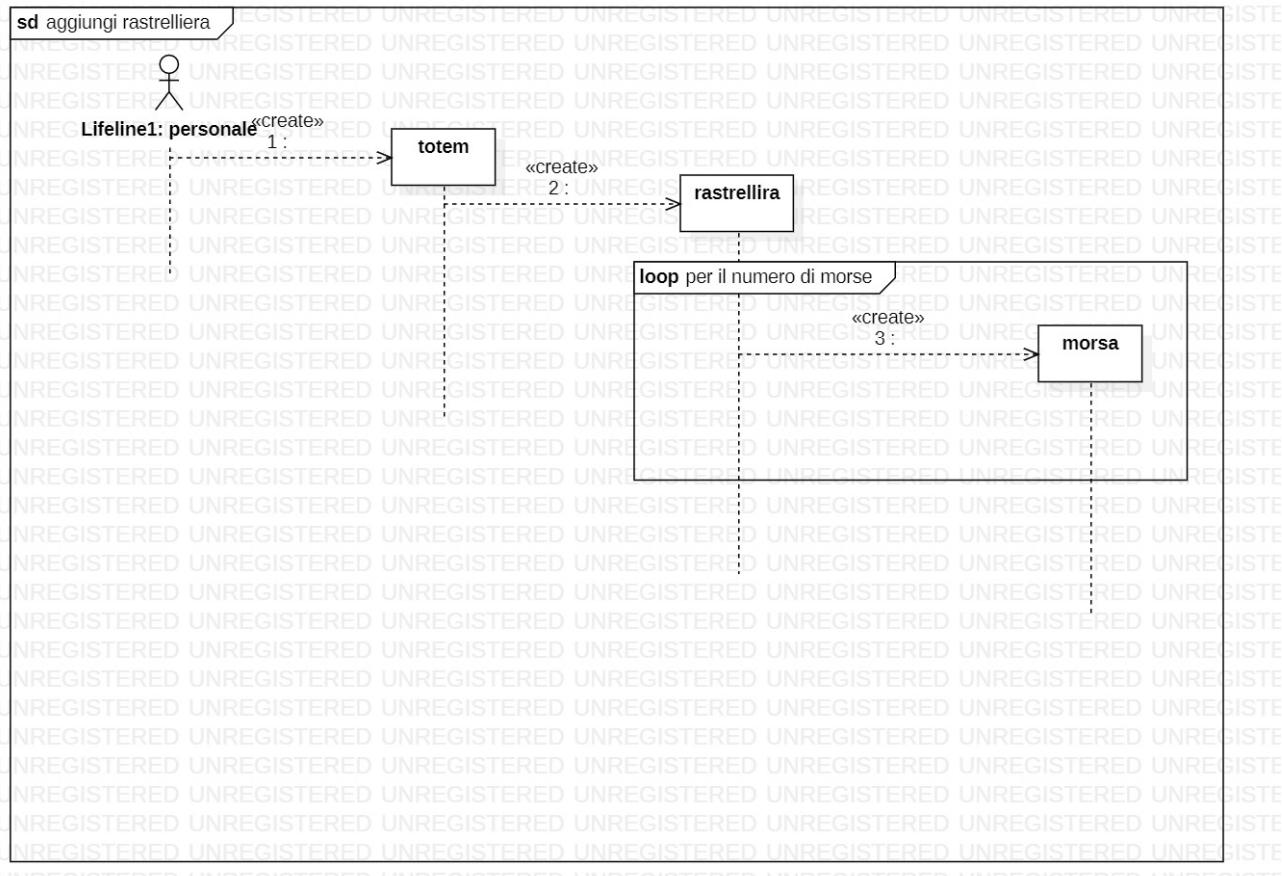
Inoltre i diagrammi di sequenza ed attività fanno riferimento al diagramma di progetto sopra riportato.

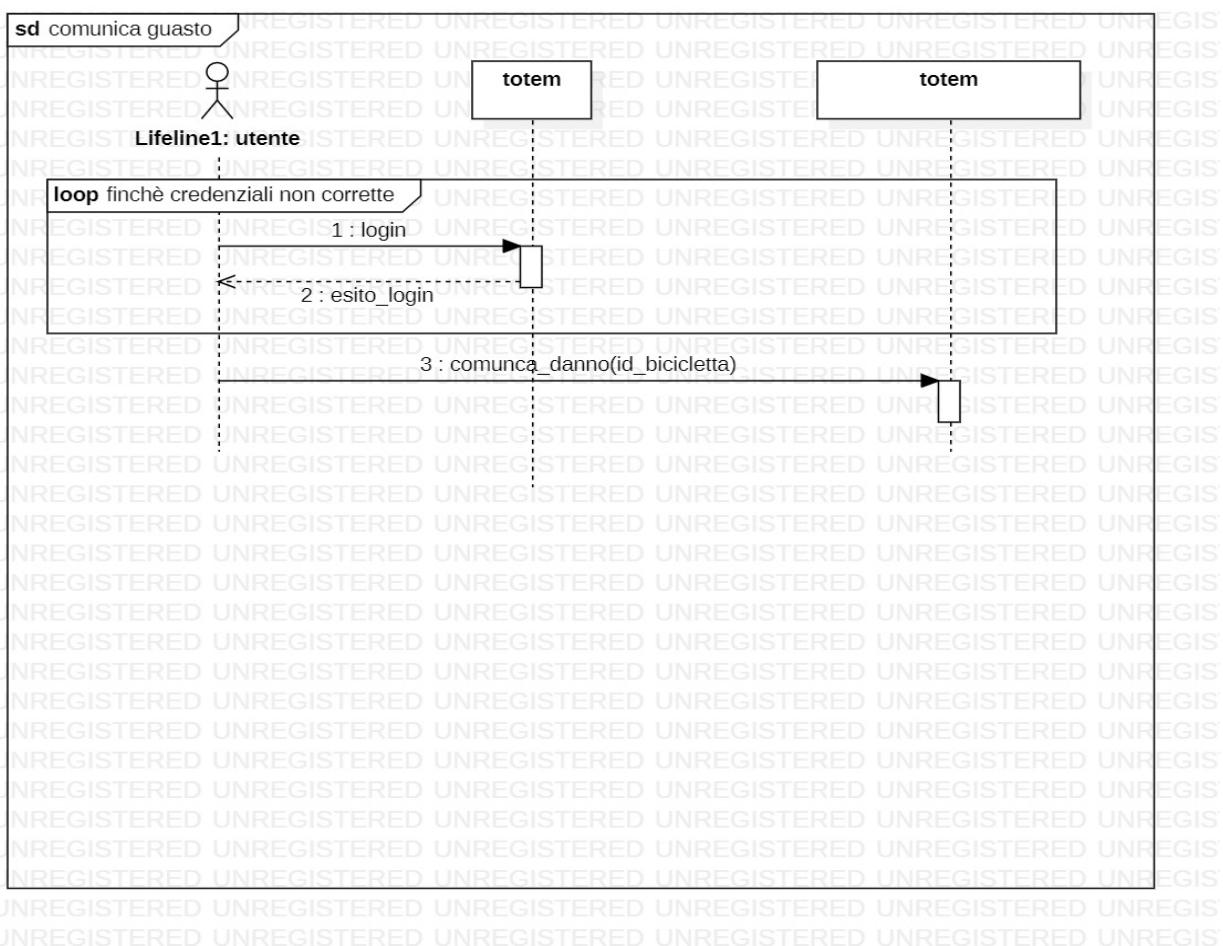
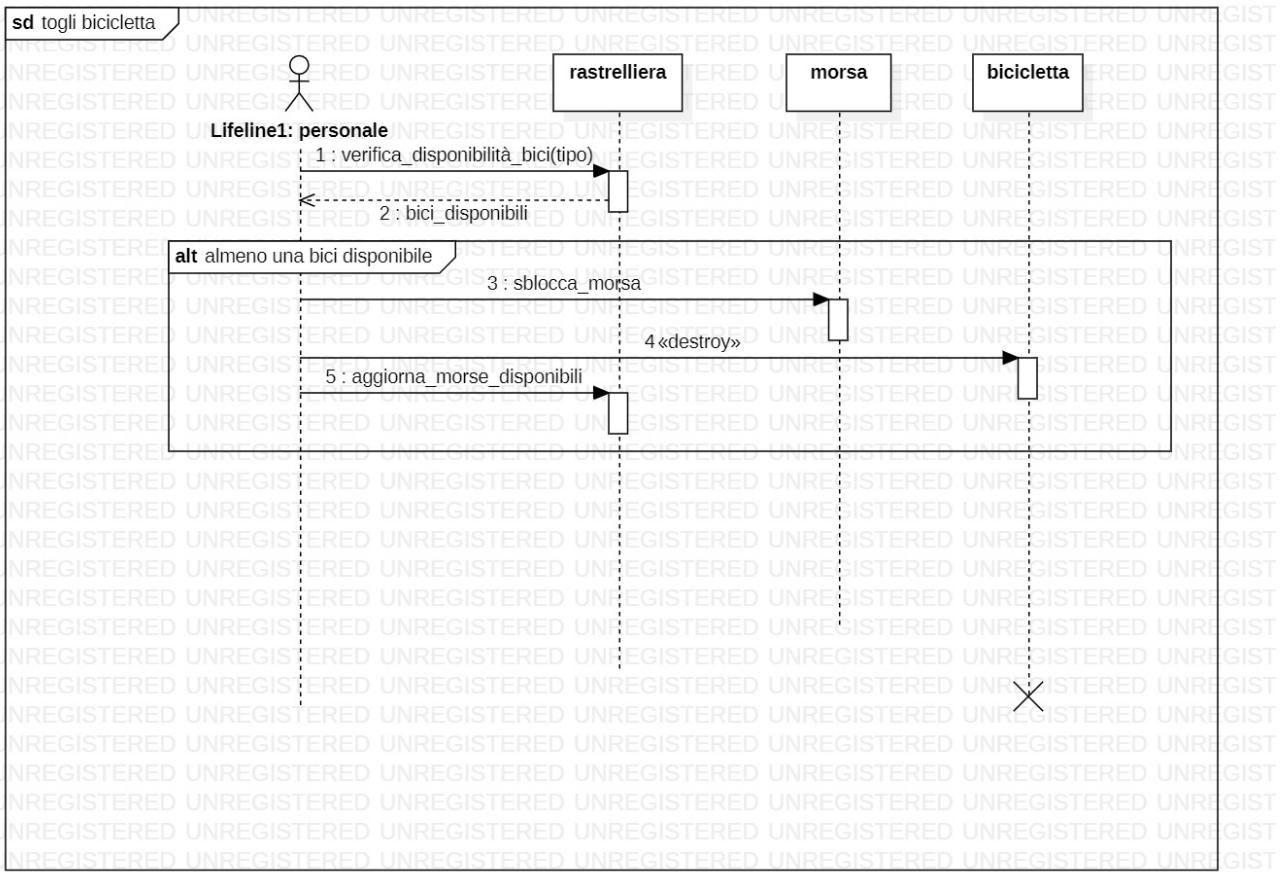
2.4 Diagrammi di sequenza





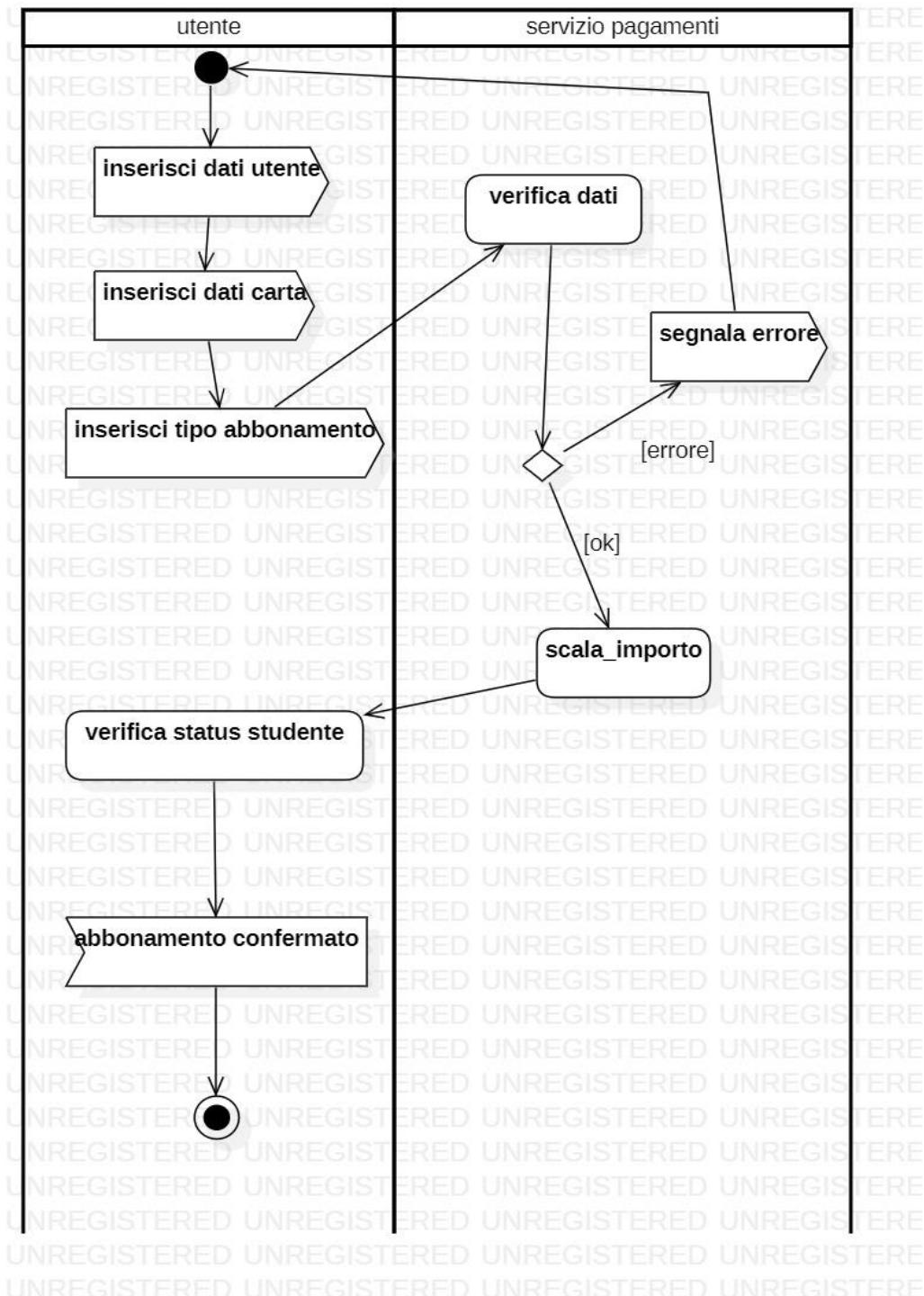




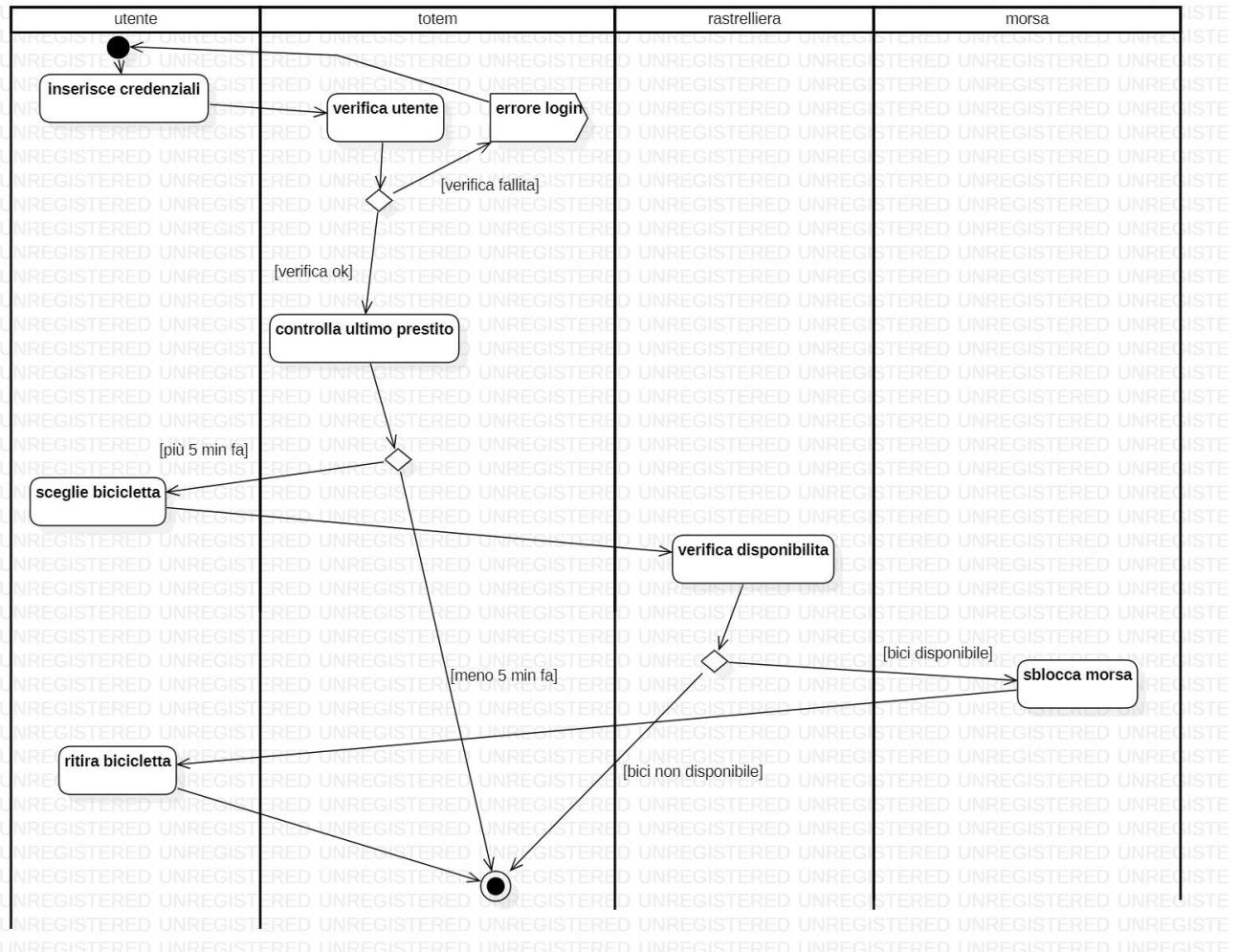


2.5 Diagrammi delle attività

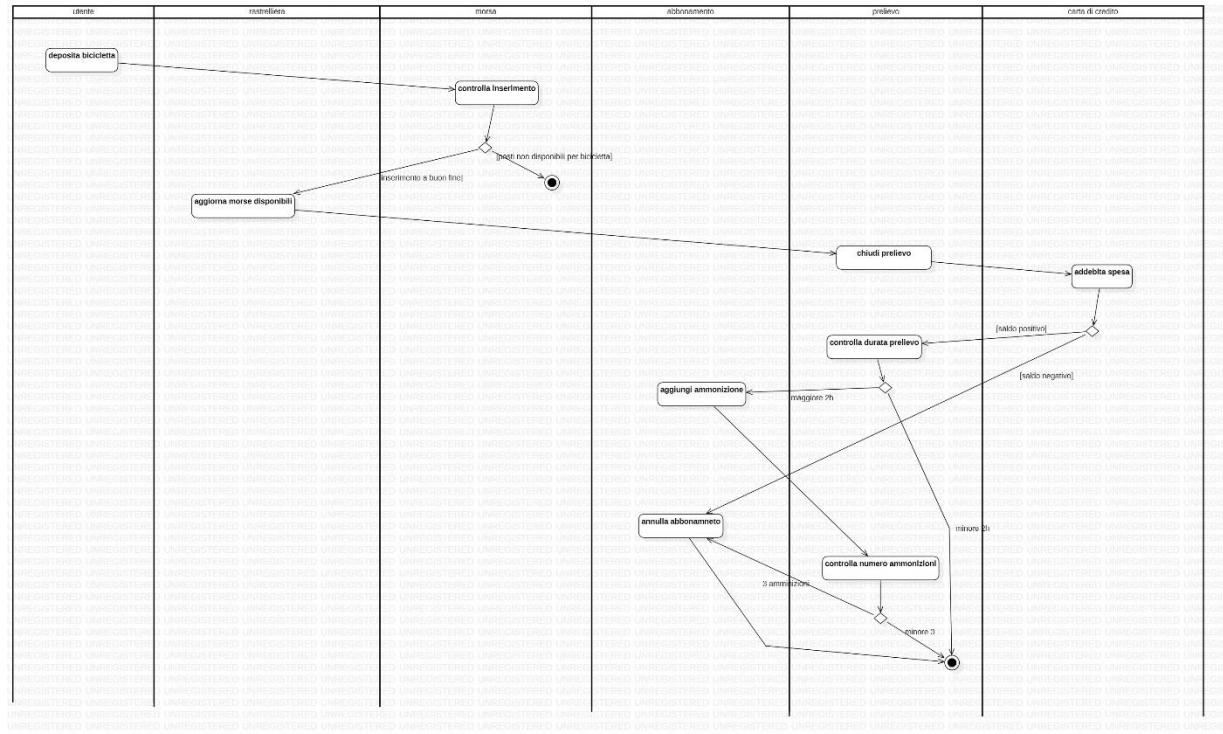
REGISTRAZIONE



RITIRO BICICLETTA

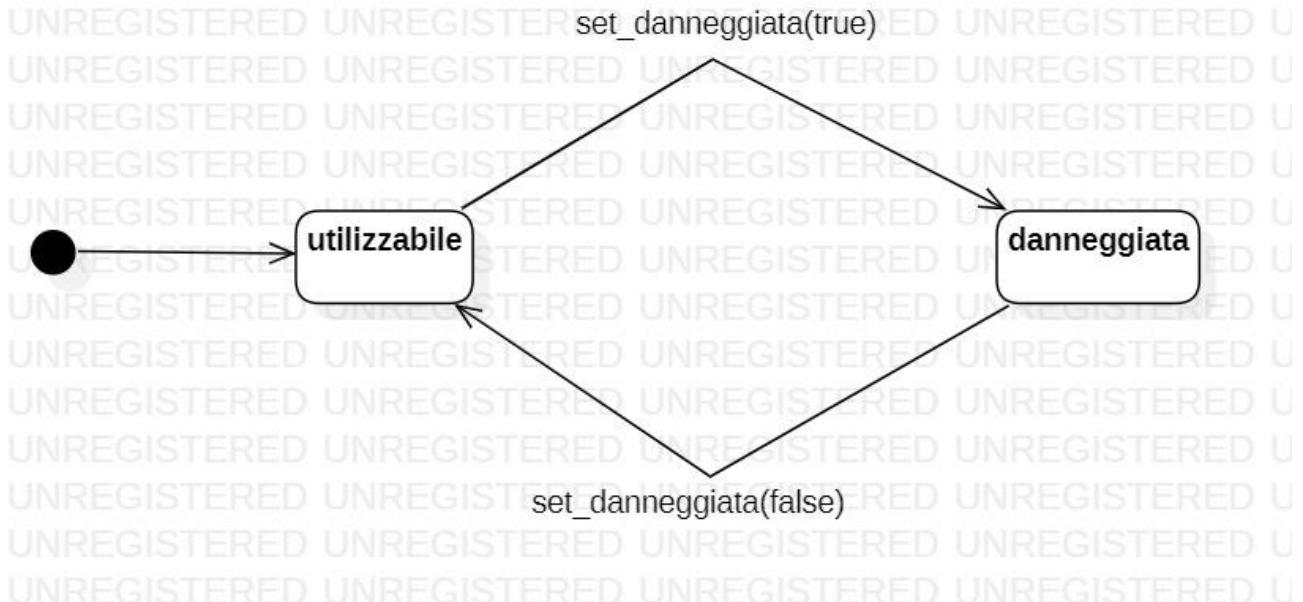


DEPOSITO BICICLETTA

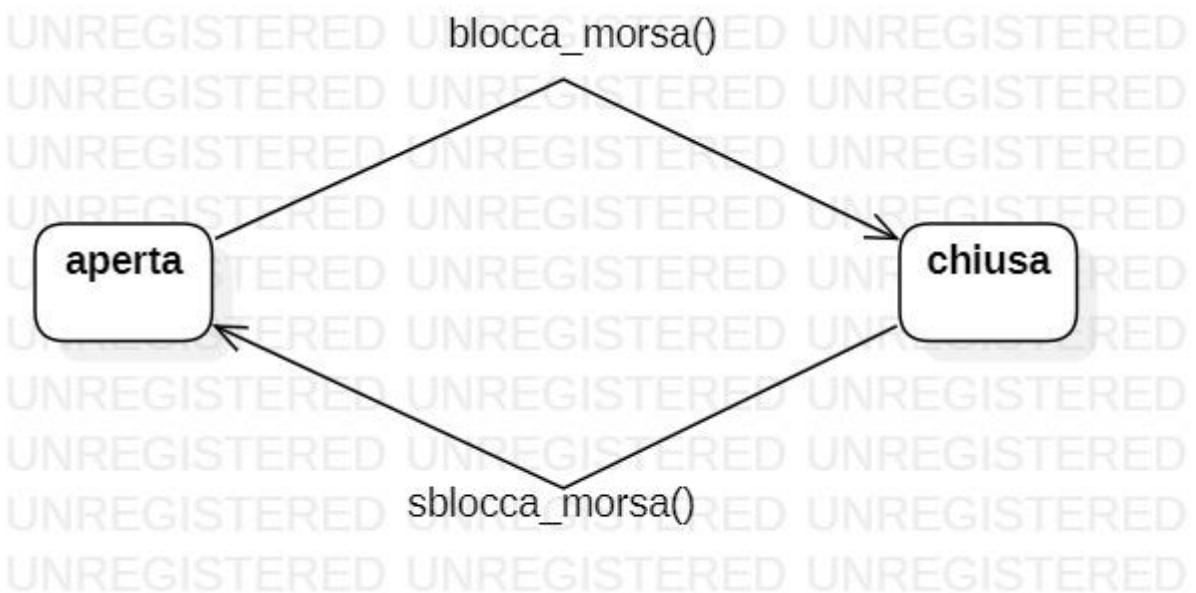


2.6 Macchine di stato

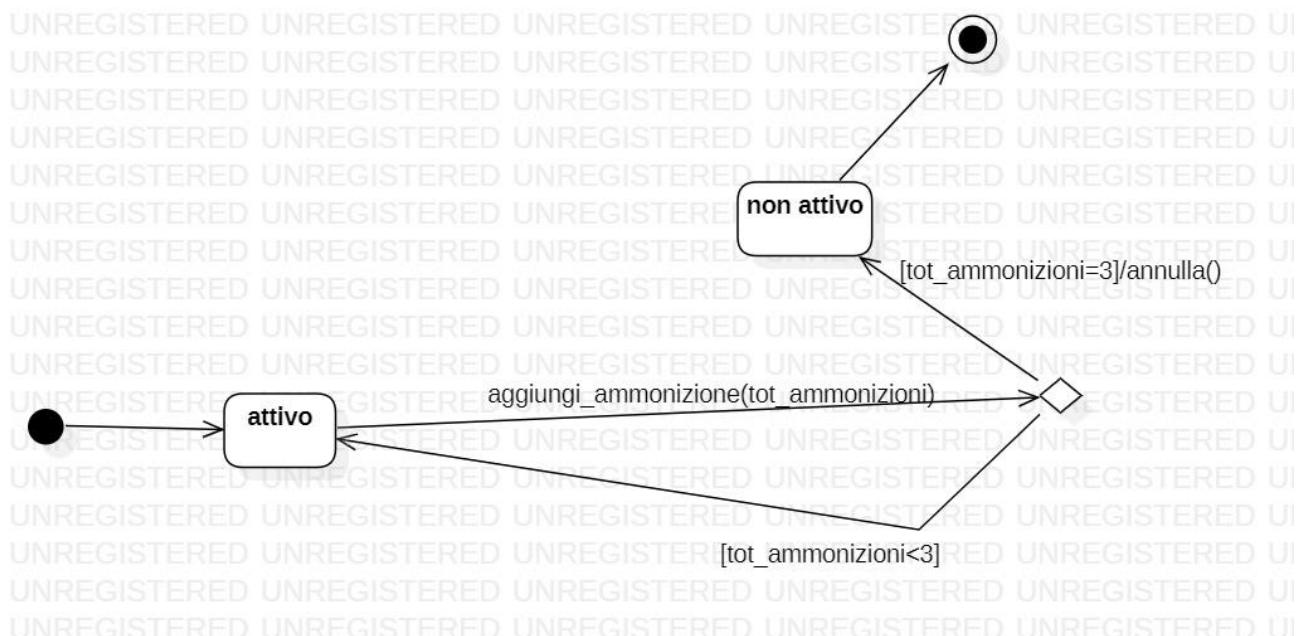
BICICLETTA



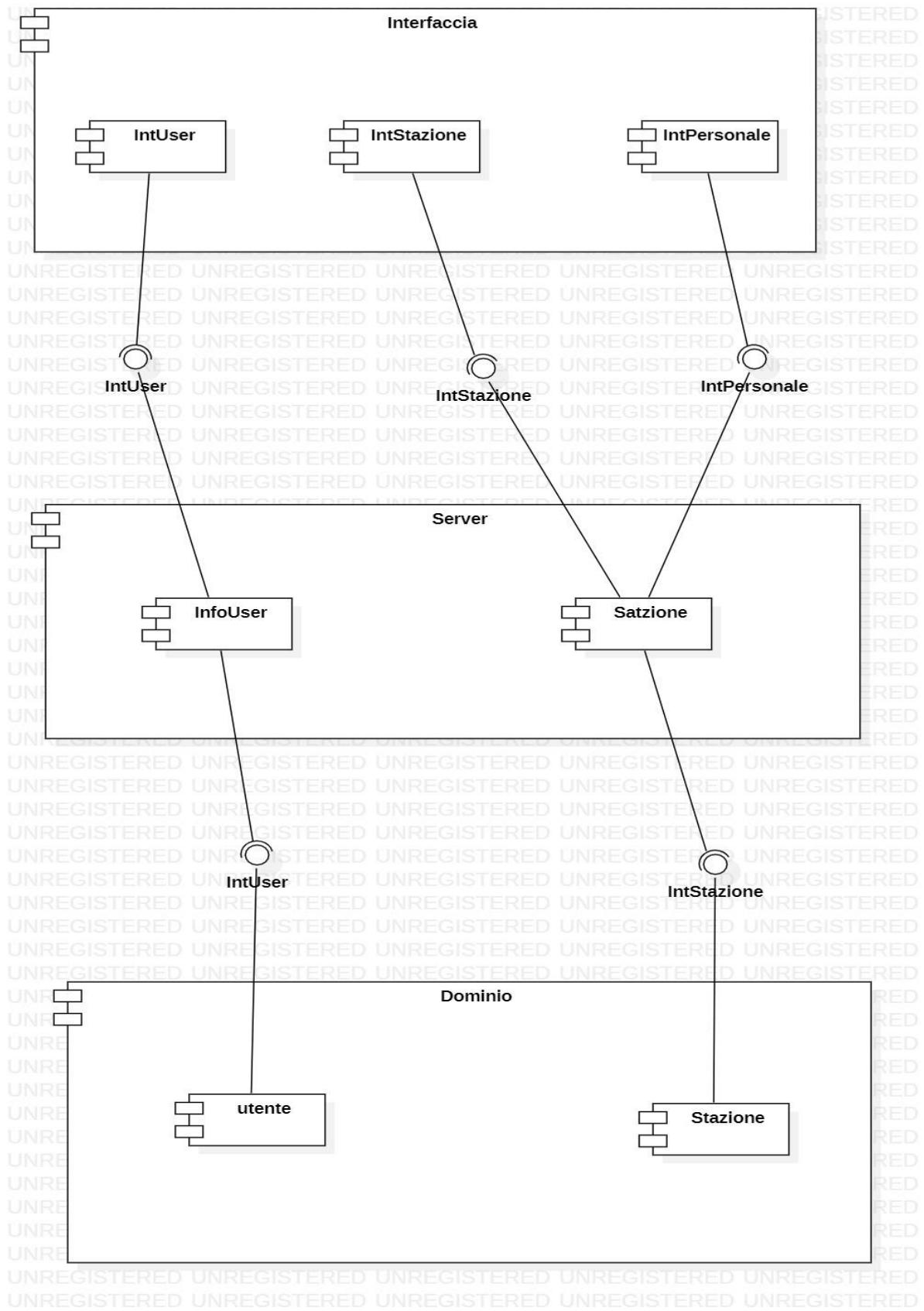
MORSA



ABBONAMENTO

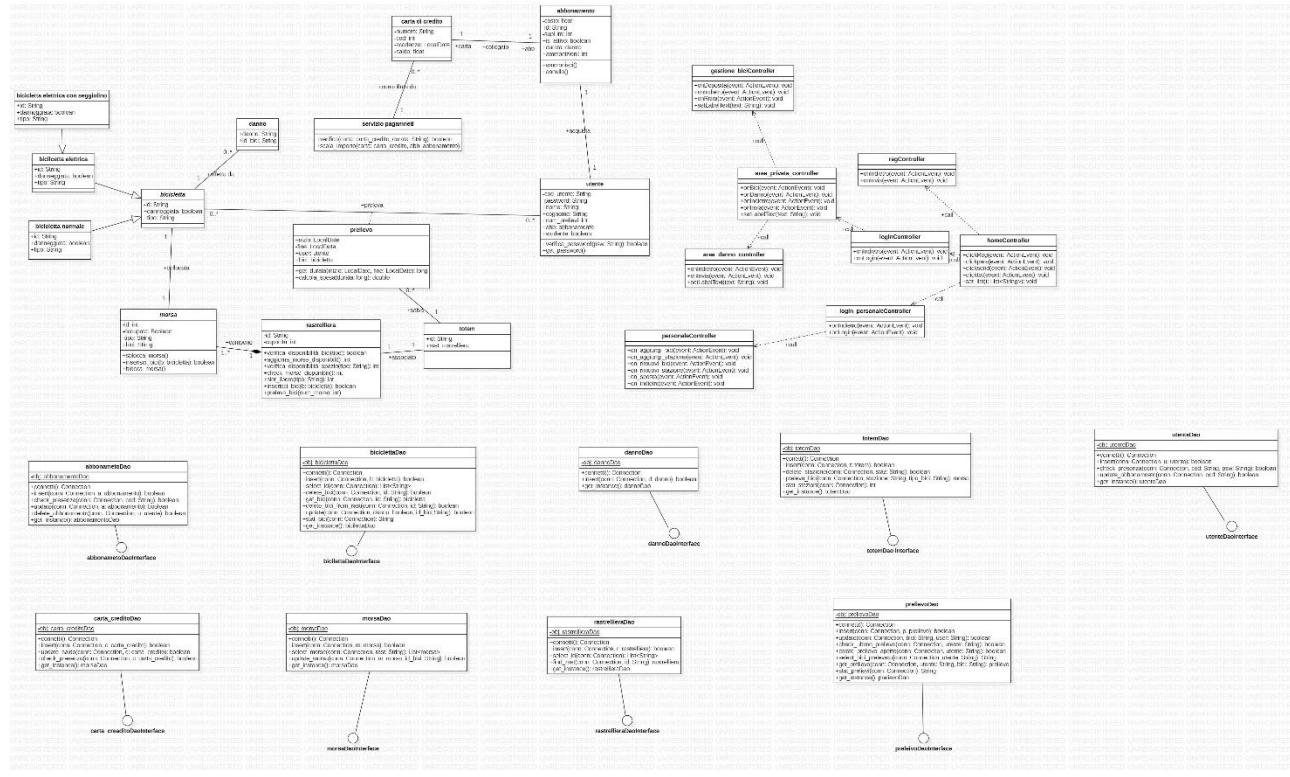


2.7 Diagramma dei componenti



3 Implementazione del sistema

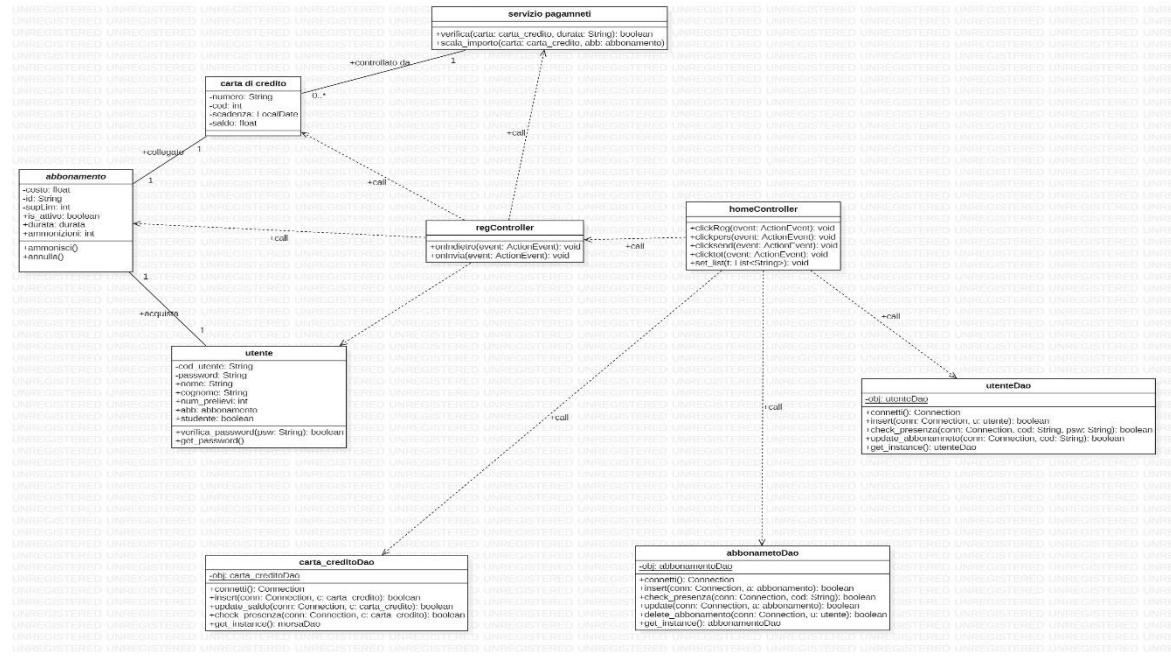
3.1 Diagramma delle classi (modello di programma)



Come possiamo notare, il diagramma sopra riportato NON è completo. È possibile osservare tre macro-aree:

1. Top-left: con classi del dominio
2. Top-right: con classi controller
3. Bottom: classi DAO

Ovviamente queste tre parti sono in stretta collaborazione l'una con l'altra però per non "sporcare" troppo il diagramma ho mantenuto queste parti separate. Nonostante la separazione, riporterò qua sotto un focus sul diagramma di programma sopra riportato per fare vedere in una piccola parte le dipendenze che intercorrono tra alcune classi.



Come possiamo notare il controller "homeController" richiama "regController". In quest'ultima classe vengono creati tutti gli oggetti inerenti alla registrazione di un utente (l'utente stesso, il suo abbonamento e la carta di credito a lui associata). Dopo ciò, questi oggetti vengono salvati nel db tramite le opportune classi DAO

3.1.1 Discussione dei Design Pattern utilizzati

DAO

Il pattern DAO (Database Access Object) ha lo scopo di raccogliere nelle classi "...Dao" il codice che gestisce gli accessi al database. In questo modo si rende il codice ben organizzato ed ogni classe viene opportunamente separata in base a scopi e funzionalità. Inoltre, per favorire l'estensibilità, queste classi implementano altrettante interfacce.

MVC

Il progetto è suddiviso in tre strati funzionali:

- 1) La parte di modello che contiene le classi cui istanze devono essere visualizzate e manipolate
- 2) La parte di vista che contiene gli oggetti utilizzati per visualizzare i dati del modello all'utente tramite interfaccia
- 3) La parte dei controller che contiene gli oggetti che gestiscono l'input dell'utente e lo elaborano

Lo scopo di questa divisione è quello di avere basso accoppiamento e alta coesione.

SINGLETON

Sono presenti diverse classi singleton nel progetto. Ad esempio, le classi DAO sono singleton: questo perché si tratta di oggetti di cui ha senso che esista una sola istanza, piuttosto che più istanze parallele. All'interno del diagramma delle classi è visibile il loro utilizzo grazie all'attributo statico obj ed al metodo statico get_instance().

OBSERVER

Il pattern Observer lo ho usato quando è stata trattata l'interfaccia grafica, sempre attraverso le classi controller. Agli elementi dell'interfaccia sono stati associati degli oggetti (labels, campi testuali, box, menù a tendina ecc...), con delle funzioni che scattano al verificarsi di eventi su tali oggetti (ad esempio cliccando un ipotetico tasto indietro scatta il metodo onIndietro che cambia ipoteticamente la scena). Questi metodi sono implementati nei controller.

CREATOR

Con il pattern creator è possibile capire quale oggetto deve avere la responsabilità di creare un altro. Solitamente si utilizzano queste linee guida comuni:

- 1) A ha la responsabilità di creare B se B aggrega oggetti di tipo A
- 2) B registra/utilizza A
- 3) B possiede le informazioni per creare A

Nel progetto. Possiamo vedere l'utilizzo del pattern creator ad esempio quando creiamo le morse. Chi è che deve crearle? Per Creator è rastrelliera perché la rastrelliera aggrega N morse.

3.2 Gestione dei dati persistenti

È stato creato un database contenente tutte le informazioni necessarie per la gestione degli utenti e la gestione degli elementi del sistema di bike sharing (bici, morse ecc).

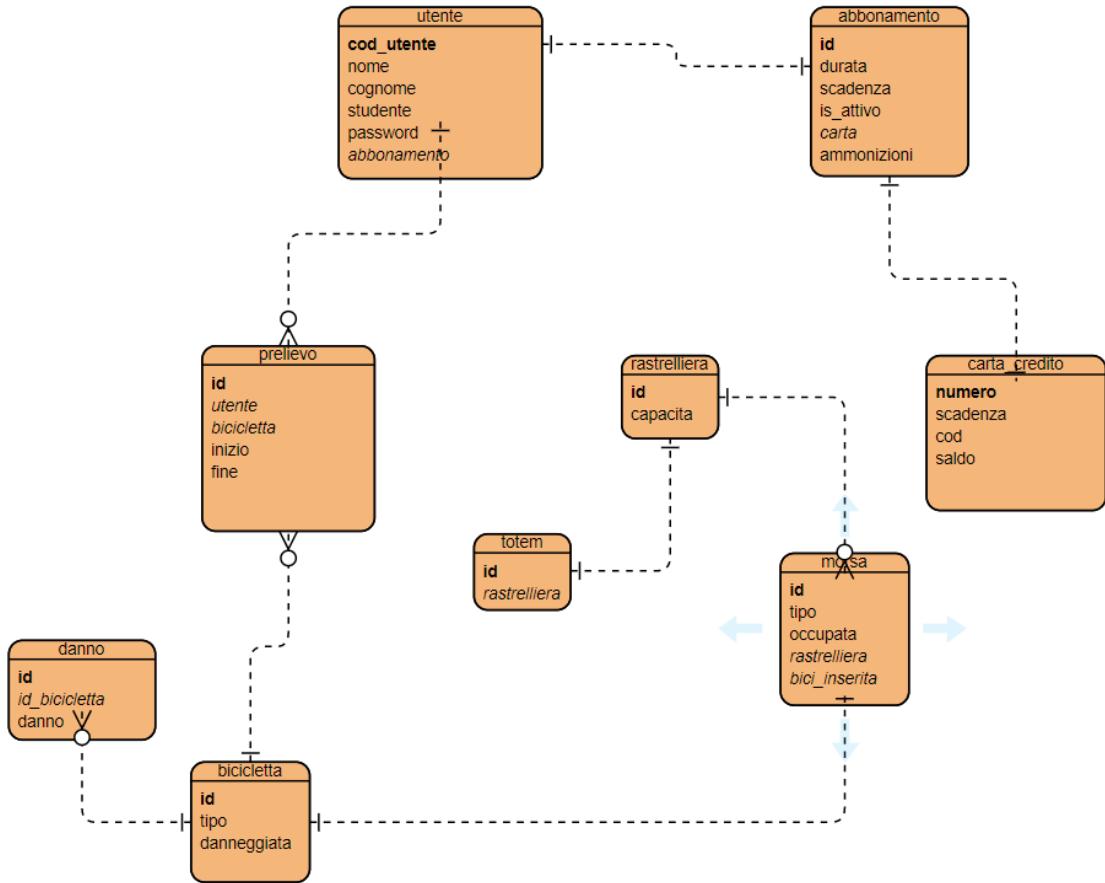
Vediamo ora come il db è strutturato:

|-----< one to many

|-----| one to one

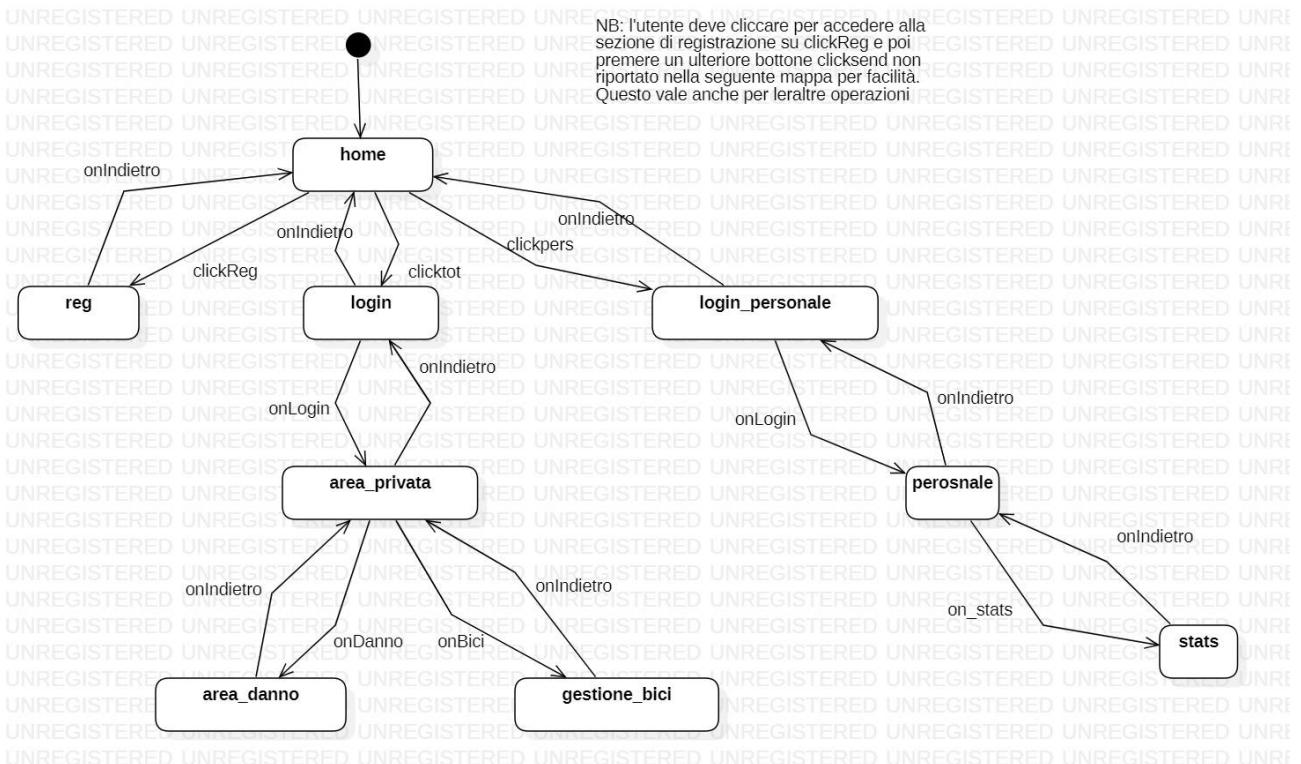
Chiave primaria

Chiave secondaria

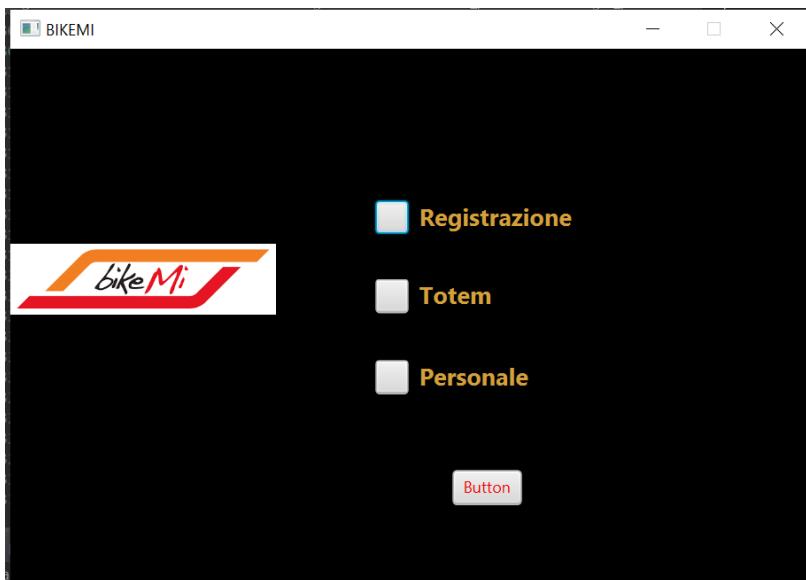


3.3 Descrizione dell'Interfaccia Grafica

Ecco la navigation map:



Ecco alcune delle schermate del mio progetto:



Rappresenta la prima pagina che incontriamo. All'interno di essa si può scegliere di registrare un nuovo abbonamento, fare il login dal totem oppure entrare nella pagina web riservata al personale che può effettuare determinate operazioni

A screenshot of a Windows application window titled "BIKEMI_REGISTRAZIONE". The title bar also includes the word "REGISTRAZIONE". The interface is light-themed. It features two main sections: "INSERIRE DATI RELATIVI ALL'UTENTE" and "INFROMAZIONI RELATIVA ALLA CARTA DI CREDITO". In the first section, there are fields for "Nome" (Name) and "Cognome" (Surname). In the second section, there are fields for "Numero" (Number), "Codice di sicurezza" (Security code), and "Scadenza" (Expiration date). Below the "Nome" field is a checkbox labeled "sono uno studente" (I am a student). At the bottom, there is a section for "INFORMAZIONI RELATIVA ALL'ABBONAMENTO" with a dropdown menu for "Tipo" (Type) and a "Invia" (Send) button.

Pagina di registrazione. Sono stati fatti alcuni controlli dell'input su:

1. Controllo presenza di tutti i campi necessari
2. Controllo sui dati della carta
3. Controlli sulla carretta scelta dell'abbonamento
4. Controllo sulla password(almeno una maiuscola e una cifra)

BIKEMI_LOGIN

LOGIN

codice utente:

password:



Semplice pagina di login nella quale l'utente inserisce codice utente e password. Esiste una pagina semi uguale per il personale.

BIKEMI_AREA_PRIVATA

BENVENUTO NELL'AREA RISERVATA



Comunica Danno



Ritira/deposita bicicletta

Menù che un utente loggato vede. Da qua l'utente può comunicare un danno o ritirare/depositare una bicicletta

BIKEMI_GESTIONE_BICI

GESTIONE BICI

selezionare stazione in cui ti trovi

RITIRO

Tip bici da ritirare:

DEPOSITO

Da questa schermata si seleziona la stazione dove si è presenti e si preleva/ deposita una bici.

NB: il tipo di bici da ritirare può anche non essere selezionato quando si effettua un deposito

PAGINA DEL PERSONALE

AGGIUNGI STAZIONE

Capacità: **Aggiungi stazione**

RIMUOVI STAZIONE

ID: **Rimuovi Stazione**

AGGIUNGI BICI

Tipo: **RIMUOVI BICI**

Stazione: **Rimuovi Bici**

SPOSTA BICI

ID bici: nuova destinazione: **Sposta**

*per stazione si intende l'insieme di totem, rastrelliera e morsa che la compone

Da qua il personale (dopo essersi autenticato in una pagina non qui riportata) può effettuare delle operazioni di aggiunta/ rimozione di bici/stazioni (rastrelliera+morse+totem).

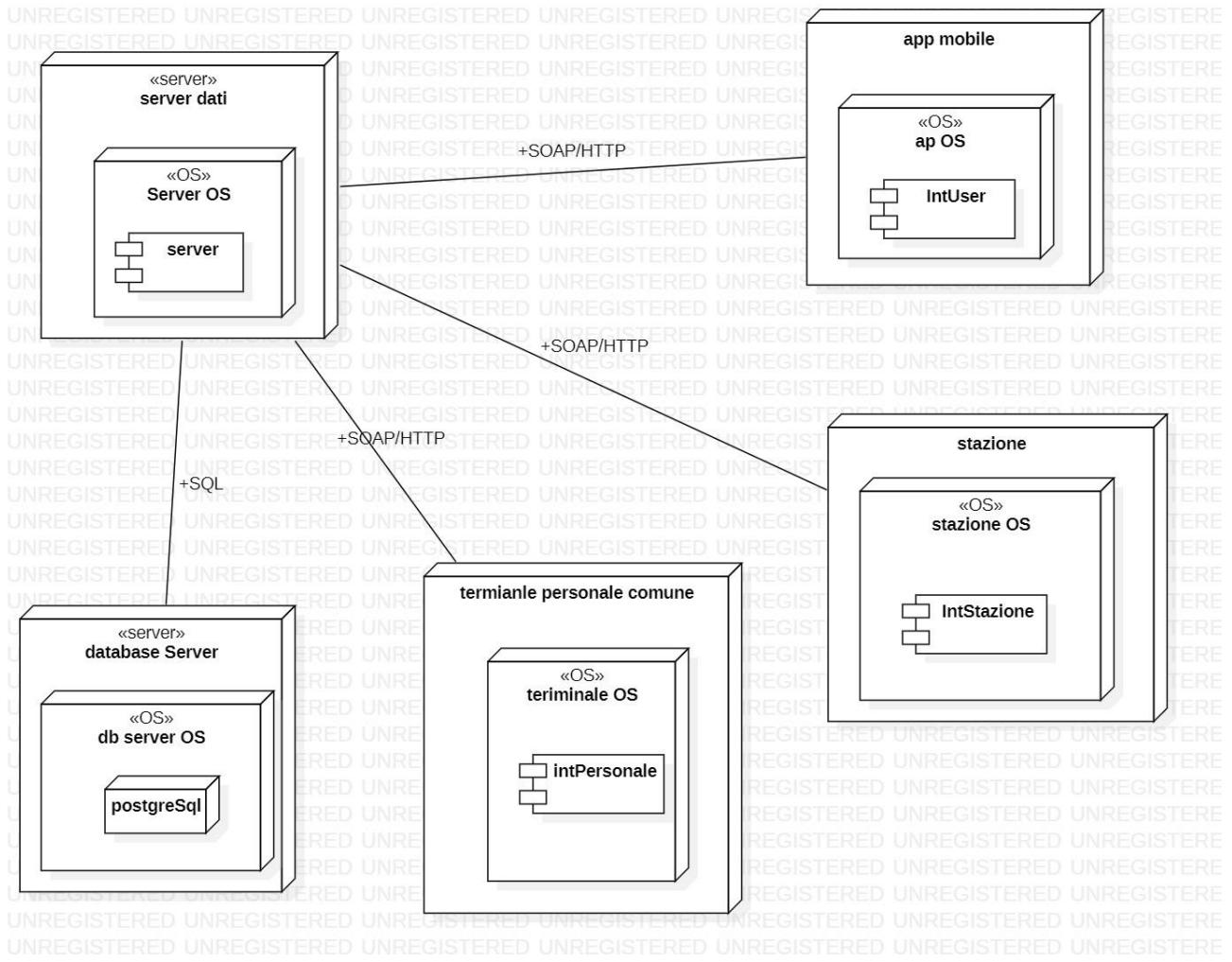
Da qua il personale può vedere anche dei semplici dati statistici

STATISTICHE

numero stazioni nel db:	2
numero utenti nell db:	1
durata media dei prelievi	00:04:19.085127
tipo bici più utilizzata	normale

Semplice pagina che mostra alcune statistiche riguardanti tempi di utilizzo e info su bici/stazioni

3.4 Diagramma di deployment



3.5 Specifica e verifica dei vincoli

ABBONAMENTO

```
{ context abbonamento
  inv: self.allInstances -> isUnique(id)
  inv: self.durata = #giornaliero or self.durata = #settimanale or self.durata= #mensile
  inv: self.ammonizioni <= 3
  inv: self.ammonizioni = 3 implies self.is_attivo = false
}
```

MAPPATURA IN JML:

```
/*@
```

```

* invariant (\forall abbonamento a1, a2; \created(a1) && \created(a2); a1 != a2
==>!(a1.id.equals(a2.id))) ;

*
* invariant durata==#giornaliero || durata==#settimanale || durata==#annuale;
* invariant self.ammonizioni>=0 && self.ammonizioni<=3
* invariant (self.ammonizioni==3) ==> (self.is_attivo==false);
*
*
@*/

```

```

{ context abbonamento::Attiva()
post: ammonizioni=ammonizioni@pre + 1
}

```

MAPPATURA IN JML:

```

/*@
*
*requires attivo==true
*ensures ammonizioni=\old(ammonizioni)+1;
*
@*/

```

BICICLETTA

```

{ context bicicletta
inv: self.allInstances -> isUnique(id)
}

```

MAPPATURA IN JML:

```

/*@

```

```

*invariant (\forall bicicletta b1, b2; \created(b1) && \created(b2); b1 != b2 ==>
!(b1.id.equals(b2.id)))

*
@*/

```

UTENTE

```

{context utente

inv: self.allInstances()->isUnique(cod_utente)

}

```

MAPPATURA IN JML:

```

/*@

*invariant (\forall utente u1, u2; \created(u1) && \created(u2); b1 != b2 ==>
!(u1.cod_utente.equals(u2.cod_utente)))

*
@*/

```

MORSA

```

{context morsa

inv: self.tipo = #elettrica or self.tipo = #normale

inv: self.tipo = #elettrica implies (self.bici_inserita.ocIsKindOf(elettrica) or
self.bici_inserita.ocIsKindOf(elettrica_con_seggiolino) or self.bici_inserita = null)

inv: self.tipo = #normale implies (self.bici_inserita.ocIsKindOf(normale) or self.bici_inserita = null)

}

```

MAPPATURA IN JML:

```

/*@

*invariant morsa.tipo.equals("normale") || morsa.tipo.equals("elettrica")

*invariant morsa.tipo.equals("normale") ==> (morsa.bici_inserita instanceof bici_normale
or morsa.bici_inserita ==null);

```

```

*invariant morsa.tipo.equals("elettrica") ==> (morsa.bici_inserita instanceof bici_elettrica
or morsa.bici_inserita instanceof bici_elettrica_con_seggiolino or morsa.bici ==null);

@*/

```

3.6 Descrizione del testing

Ho costruito i test driver con un approccio Program-Driven. Il criterio di copertura utilizzato è il Branch Coverage. Ho testato alcune operazioni sulle classi del dominio. Possiamo trovare i test nel package test (ogni file del package contiene test per una particolare classe del dominio).

Qua di seguito riporto un test fatto sul calcolo delle tariffe per i prelievi delle biciclette

```

//gli studenti le bici normali non le pagano mai
@Test
void test() {
    carta_credito c=new carta_credito("1111111111111112",112,LocalDate.now().plusYears(2),100);
    abbonamento a=new abbonamento(durata.annuale,c,true,0);
    utente u=new utente("Marco","Cavallari","Password_corretta123",true,a);
    bici_normale b=new bici_normale("CODICEBICI",false);
    prelievo p=new prelievo(u,b,LocalDateTime.now());
    long durata=p.get_durata(p.get_inizio(), LocalDateTime.now().plusMinutes(10));
    double output=p.calcola_spesa(durata);
    assertEquals(0,output);
}

//studenti enon per utilizz< a 30 min pagano 25 centesimi
@Test
void test2() {
    carta_credito c=new carta_credito("1111111111111112",112,LocalDate.now().plusYears(2),100);
    abbonamento a=new abbonamento(durata.annuale,c,true,0);
    utente u=new utente("Marco","Cavallari","Password_corretta123",true,a);
    bici_elettrica b=new bici_elettrica("CODICEBICI",false);
    prelievo p=new prelievo(u,b,LocalDateTime.now());
    long durata=p.get_durata(p.get_inizio(), LocalDateTime.now().plusMinutes(10));
    double output=p.calcola_spesa(durata);
    assertEquals(0.25,output);
}

//per l'utilizzo della bici elettrica con seggiolino di 3 ore il pagamento è di 7 euro e 75 centeimi perchè:
//costo prima mezzora di utilizzo : 0.25
//costo seconda mezzora di utilizzo : 0.50
//costo terza mezzora di utilizzo : 1.00
//costo quarta mezzora di utilizzo : 2.00
//costo per ore successive (nel nostro esempio di 1 ora oltre alle prime due) : 4
//--->7.75
@Test
void test3() {
    carta_credito c=new carta_credito("1111111111111112",112,LocalDate.now().plusYears(2),100);
    abbonamento a=new abbonamento(durata.annuale,c,true,0);
    utente u=new utente("Marco","Cavallari","Password_corretta123",true,a);
    bici_elettrica_con_seggiolino b=new bici_elettrica_con_seggiolino("CODICEBICI",false);
    prelievo p=new prelievo(u,b,LocalDateTime.now());
    long durata=p.get_durata(p.get_inizio(), LocalDateTime.now().plusMinutes(180));
    double output=p.calcola_spesa(durata);
    assertEquals(7.75,output);
}

```

3.7 Note per l'installazione e l'utilizzo

Eclipse: Version: 2021-03 (4.19.0)

Java: 14.0.2

javaFx 16

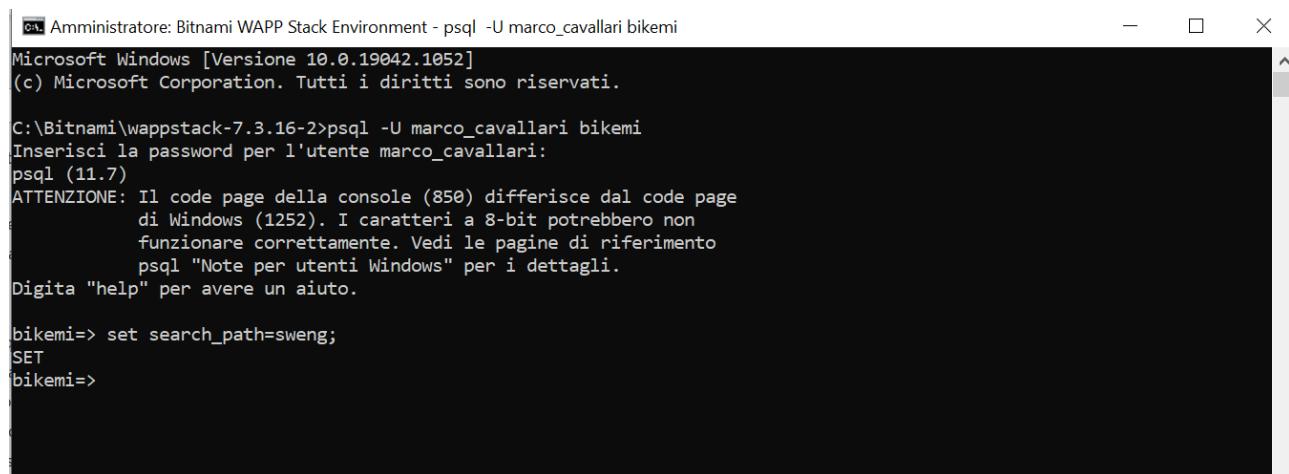
Una volta aperto il progetto in Eclipse è possibile dia molti errori causa la mancanza di javaFX o del driver di postgres. Per risolverli basterà includerli nel progetto. Ad ogni modo come prova della corretta funzione del programma allegherò un video nella quale farò vedere alcune operazioni base che si possono fare (registrazione, prelievo, deposito ecc...) .

Le credenziali di accesso al database sono [utente: marco_cavallari, password: sweng2021]

Dopodichè occorre selezionare lo schema corretto eseguendo il comando:

```
set search_path=sweng;
```

Allego foto



```
C:\ Amministratore: Bitnami WAPP Stack Environment - psql -U marco_cavallari bikemi
Microsoft Windows [Versione 10.0.19042.1052]
(c) Microsoft Corporation. Tutti i diritti sono riservati.

C:\Bitnami\wappstack-7.3.16-2>psql -U marco_cavallari bikemi
Inserisci la password per l'utente marco_cavallari:
psql (11.7)
ATTENZIONE: Il code page della console (850) differisce dal code page
di Windows (1252). I caratteri a 8-bit potrebbero non
funzionare correttamente. Vedi le pagine di riferimento
      psql "Note per utenti Windows" per i dettagli.
Digita "help" per avere un aiuto.

bikemi=> set search_path=sweng;
SET
bikemi=>
```

All'interno del sistema dovrebbe essere già presente un utente con le seguenti credenziali:

cod_utente: V78DMJKNR9

passwod: Marco1999

All'interno dovrebbero essere presenti anche due stazioni con id EBJI8S5ROM e l'altra con id YVPT5K4EIW.

In EBJI8S5ROM è presente una bici elettrica pronta per l'utilizzo.

Con questo la mia presentazione è conclusa e spero che tutto funzioni correttamente. Non esisti a contattarmi nel caso avesse qualche dubbio o problema