

Progetto d'esame: laboratorio di PISSIR

Membri:

Ferolo 20041138
Nicola 20040045
Veronese 20039081

DIVISIONE DEI COMPITI

Ferolo:

- Models User e Admin con implementazione dei relativi controller, service e repository
- Servizio di messaggistica MQTT

Nicola:

- Models Reservation e ChargeRequest con implementazione dei relativi controllers, service e repository
- Implementazione dell'MWBot

Veronese:

- Models Delayed-Reservation, Macchine e Payment e relativi controller, service e repository
- Implementazione della parte sulla sicurezza (token jwt, https)

La classe ParcheggioApplication, ossia colei che avvia il progetto intero, è stata progettata da tutti e 3 i componenti del gruppo.

La parte di Front-end è stata progettata da tutti e 3 i componenti del gruppo, andando a decidere stile e funzionalità del sito sempre confrontando i 3 pareri.

Noi abbiamo sviluppato questo sistema su macOS utilizzando come IDE IntelliJ_IDEA per la parte di Backend in Java e JetBrains Rider per la parte di Frontend.

PANORAMICA GENERALE

Abbiamo deciso di gestire il parcheggio formato da posti dedicati alla sosta e posti dedicati alla ricarica utilizzando un bot che gestirà la ricarica di un'auto alla volta.

Vi saranno tre tipologie di utenti: BASE, PREMIUM e ADMIN che avranno rispettivamente permessi differenti per operare. Ogni utente per poter accedere all'applicazione dovrà prima effettuare il login e in caso non sia registrato devono effettuare prima la registrazione del suo account inserendo nome utente, e-mail e password utilizzando un username e una email che non siano presenti già nel database, inoltre email e password dovranno essere conformi agli standard di validità.

In seguito, potrà effettuare il login e a seguito anche il logout se è necessario.

L'utente BASE avrà la possibilità di cliccare il bottone "Entra ora" se vuole accedere al parcheggio sul subito se vi sono tutte queste condizioni:

1. Abbia un saldo sufficiente per la sosta o la ricarica desiderata, nel caso prima deve ricarica il saldo, associando, se non è già presente, una carta di credito
2. Abbia una macchina associata al suo account, se così non fosse deve prima creare una macchina con dati validi e che nel caso abbia scelto la ricarica deve essere una macchina elettrica
3. Ci sia almeno un posto libero per la ricarica o per la sosta, che nel caso non vi sia verrà visualizzato quando sarà disponibile un posto.
4. Non vi siano delle prenotazioni riferite all'utente a cui vi è associato un ritardo, nel caso devono essere pagate tutte le multe prima

L'utente BASE avrà la possibilità anche di fare un abbonamento e di diventare così utente PREMIUM, solamente se ha il saldo disponibile per farlo e solamente se non è già PREMIUM oppure ADMIN.

L'utente PREMIUM, a differenza di quello BASE, potrà effettuare la prenotazione del parcheggio cliccando il bottone "Prenota" per prenotare la sosta o la ricarica nella data e ora desiderata e valgono le stesse regole che vi sono sopra, in più verrà controllato che il tempo selezionato dall'utente per la sosta oppure, in caso di ricarica, il tempo di ricarica calcolato in base alla percentuale di ricarica che l'utente aveva inserito ,non si sovrappongano con altre prenotazioni.

Ogni utente, inoltre, se vorrà potrà cliccare "Utente" e verrà fuori una tendina con la voce "Esci Prima dal parcheggio" e nel caso vi sia una macchina all'interno del parcheggio riferita all'utente, l'utente potrà così uscire dal parcheggio.

Ogni utente inoltre potrà modificare i dati di una macchina a lui associata oppure cancellarla.

Infine, vi è l'utente ADMIN, il quale può anche lui fare le operazioni che fa l'utente PREMIUM,

ma ha altre operazioni accessorie alla sua figura, ovvero può stampare tutti i pagamenti per sosta, ricarica, oppure stampare tutti i pagamenti per categoria di utente (ADMIN, PREMIUM, BASE).

L'ADMIN potrà cancellare tutto il database oppure può cancellare tutto ma mantenendo gli utenti e lo storico tramite il bottone "Cancella prenotazioni" nella tendina ADMIN.

Inoltre, potrà attraverso il bottone "Sposta le soste" cancellare le prenotazioni finite e le inserisce all'interno dello Storico.

L'ADMIN inoltre potrà registrare un account ADMIN nel caso vi sia necessità di un altro account amministratore.

Poi potrà anche aggiornare i costi della sosta e della ricarica del parcheggio.

Infine, abbiamo inserito un bottone per poter impostare un ritardo su delle prenotazioni che scegliamo in modo da poter testare il funzionamento del bot sulla gestione dei ritardi e la gestione generale del parcheggio, in modo che se una prenotazione non è segnata in ritardo viene inserita, una volta completata, all'interno dello storico; altrimenti la prenotazione viene inserita nella tabella delle prenotazioni in ritardo, ovvero quelle che hanno sfiorato le tempistiche di uscita dal parcheggio.

COMPONENTI

Abbiamo 2 cartelle che sono "parcheggio" e "ParcheggioGUI", il primo è per la parte di back-end e conterrà oltre al codice riguardante il back-end, anche la parte che riguarda la Sbarra e l'emulatore, il secondo riguarda invece tutta la parte grafica che permetterà all'utente di poter operare sul parcheggio.

Nella cartella Parcheggio c'è la suddivisione di tutta la parte che gestisce le richieste tramite API, difatti quando l'utente sceglierà quale operazione eseguire in base a quelle proposte sul sito, verrà richiamata l'API dedicata che è definita dentro ad una delle sette classi che fanno da controller e difatti si trovano all'interno della classe "Controllers":

- AdminController
- ChargeRequestController
- DelayedReservationController
- MacchineController
- PaymentController
- ReservationController
- UserController

Al link <https://localhost:8443/swagger-ui/index.html#/> vi è la documentazione riguardante le API che si trovano all'interno dei controller sopra citati. (prima di aprire il link avviare il programma)

Ad ogni API è riferito un metodo e le classi Controller andranno a richiamare dei metodi delle classi Service che faranno dei controlli e eseguiranno delle operazioni sui dati che vengono passati, infine verranno poi chiamati i relativi metodi delle classi Repository, i quali ad ognuno verrà associata una query che verranno eseguite sul database.

I Service:

- ChargeRequestService
- MacchineService
- PaymentService
- ReservationService
- StoricoService
- UserService

I Repository:

- ChargeRequestRepository
- DelayedReservationRepository
- MacchinaRepository
- ParkingSpotRepository
- PaymentRepository
- ReservationRepository
- StoricoRepository
- UserRepository

All'Interno della cartella Parcheggio vi è la cartella Security dove vi sono rispettivamente i seguenti file:

- **HttpsRestTemplateConfig**: configura un RestTemplate per effettuare chiamate con HTTPS e legge da un trust store(keystore.jks) per validare le connessioni HTTPS. Utilizza **Apache HttpClient** per creare un client HTTP personalizzato con supporto SSL/TLS. Quindi con il RestTemplate risultante possiamo effettuare richieste in maniera sicura e affidabile.
- **JwtAuthenticationFilter**: permette di filtrare tutte le richieste in modo da vedere ad esempio se contengono un Token JWT e nel caso validarlo.
- **JwtUtil**: contiene tutti quei metodi per creare, gestire e manipolare il token
- **SecurityConfig**: permette di configurare la sicurezza dell'applicazione realizzata tramite SpringBoot utilizzando Spring Security e come prima cosa viene utilizzato un filtro di autenticazione basato su JWT che viene eseguito prima del filtro di autenticazione con username e password. Inoltre, vengono definite le politiche di accesso ai vari end-point, configura il CORS permettono così richieste anche da altri domini specificati, richiede HTTPS per le richieste e inoltre da il permesso di utilizzo dell'interfaccia H2 per la gestione del database.

MQTT

Mosquitto è un broker MQTT open-source che consente la comunicazione tra dispositivi IoT. Per garantire la sicurezza, abbiamo configurato il sistema affinché richieda l'autenticazione con username e password.

Il nostro file MqttConfig.java si andrà a connettere al broker mosquitto tramite username e password, inizializziamo la connessione con `tcp://localhost:1883`, impostiamo l'username e password con `options.setUserName()` e `options.setPassword()`, si conserverà al broker tramite `client.connect(options)`.

Elenco dei Topic:

- bot/
 - charging
 - gate
 - lights
- charges/
 - sosta
 - ricarica
 - create
 - delete
- users/
 - delete
 - register
 - general
 - login
 - saldo
 - carta
 - abbonamento
- errors/
 - user
 - payment
 - charge
 - reservations
 - delayed-reservations
 - macchine
 - general
 - bot
- macchine/
 - delete
 - create
 - update
 - fetch
- payments/
 - create
 - delete
 - fetch
 - filter
- delayed-reservations/
 - fetch
- reservations/
 - delete

- update
- fetch
- manage
- delay
- storico

Non abbiamo adottato una struttura precisa come formato dei messaggi MQTT, viene sempre mandato un messaggio in formato stringa; per leggere i messaggi bisogna iscriversi tramite linea di comando, inserendo per la sicurezza username e password come i comandi sotto elencati.

Esempi su come iscriversi ai topic:

- per iscriversi a tutti i topic:
mosquitto_sub -h localhost -t # -u Username -P Password
- per iscriversi al topic padre e leggere tutti i figli:
mosquitto_sub -h localhost -t "padre/#" -u Username -P Password
- per iscriversi ad un topic specifico:
mosquitto_sub -h localhost -t "padre/figlio" -u Username -P Password
- per iscriversi ad un topic e prendere quelli immediati scrivere
mosquitto_sub -h localhost -t "padre/" -u Username -P Password

BOT

Abbiamo deciso di implementare un bot che in base alla coda delle prenotazioni. esso prende quella in testa, e simula la ricarica della macchina aspettando il tempo di carica che era stato calcolato in precedenza, poi viene notificato tramite MQTT l'inizio e la fine della carica della macchina, l'apertura e la chiusura della sbarra e infine anche lo stato di occupazione del parcheggio invocando la funzione per l'aggiornamento dello stato delle lampadine. (se l'emulatore è attivato tramite il bottone direttamente in esso)
Il bot controllerà costantemente se vi sono auto da ricaricare, e se non vi sono auto attende un certo periodo di tempo prima di riprovare.
Il bot è eseguito in maniera asincrona in modo da permetterne l'esecuzione su un thread separato senza bloccare l'applicazione principale.

SBARRA

La componente Sbarra è formata principalmente da un UI che rappresenterà la sbarra a livello grafico, un model chiamato “BarrierState”, ed infine un controller che ha all'interno due API che sono mappate nel seguente modo:

- /api/barrier/open → per l'apertura della sbarra
- /api/barrier/close → per la chiusura della sbarra

Al link <http://localhost:9090/swagger-ui/index.html#/> vi è la documentazione riguardante le API di Sbarra. (il programma deve essere avvito prima)

PARCHEGGIOGUI

Riguarda il front-end ed è formato da tutti quegli elementi che permetteranno all'utente di accedere alla home del parcheggio e così poter prenotare il parcheggio e svolgere altre operazioni. L'utente potrà vedere nella home lo stato del parcheggio vedendo quante auto sono in carica e quante in sosta ed inoltre vedrà la posizione del parcheggio sulla mappa. L'interfaccia grafica dell'applicazione è sviluppata e organizzata secondo il framework ASP.NET Core Razor Pages / Blazor e avremo una cartella chiamata "Pages" dove vi saranno i vari file dei quali ognuno svolgerà un'operazione differente del parcheggio, manipolando i dati e richiamando le API del back-end attraverso il codice C# e si occuperà quindi anche della rappresentazione delle pagine definendo l'HTML con codice Razor.

H2-CONSOLE

Abbiamo deciso di utilizzare h2 perché è un database leggero che funziona all'interno dell'applicazione senza bisogno di un server separato, inoltre ha una console web che permette di eseguire query, visualizzare dati e gestire il database. Inoltre, non richiede un'installazione complessa e si può integrare con Spring Boot in pochi passi.

DIAGRAMMI DI SEQUENZA

EntraOraSeq e PrenotaSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l'username dell'utente tramite token e in seguito controlliamo se all'utente sono associate delle prenotazioni in ritardo, nel caso vi siano l'utente non potrà proseguire e non potrà così entrare nel parcheggio; se non vi sono così prenotazioni in ritardo associate all'utente potrà selezionare come prima cosa se vuole che avvenga la ricarica della macchina cosa potrà inserire i kw altrimenti selezionerà la sosta e così potrà inserire il tempo di sosta. Verranno visualizzate in seguito solo le macchine elettriche se è richiesta la ricarica, altrimenti tutte le macchine associate dall'utente. Nel caso l'utente non abbia macchine a lui associate non potrà proseguire fino a che non ne registrerà una. Dopo aver selezionato la macchina viene verificata poi la disponibilità per controllare se vi è almeno un posto libero per la macchina dell'utente, se non vi è disponibilità viene visualizzato l'orario in cui l'utente potrà riprovare per accedere al parcheggio. Nel caso vi sia disponibilità allora viene creata la richiesta e così l'utente può entrare nell'immediato nel parcheggio.

AbbonamentoSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l'username e ruolo dell'utente tramite token, in seguito andiamo a controllare se l'utente è BASE e in questo caso potrà proseguire e si controllerà se il saldo dell'utente è sufficiente per fare l'abbonamento; in caso non sia sufficiente verrà chiesto di ricarica il saldo e di riprovare successivamente l'operazione, se invece è sufficiente si chiamerà la relativa API e l'utente passerà da BASE a PREMIUM.

Aggiorna-costoSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l'username dell'utente tramite token. L'utente che in questo caso è l'ADMIN potrà modificare i costi per la ricarica oppure per la sosta, nel caso il valore del costo dovrà essere uguale o maggiore

di 0 e solamente in questo caso verrà richiamata la relativa API per andare a modificare i costi.

AggiornaCartaSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l'username dell'utente tramite token. In seguito, l'utente inserirà il numero della propria carta e verrà verificato che il numero fornito sia valido, nel caso si si andrà a richiamare l'API appropriata per andare ad assegnare il relativo numero di carta all'utente, altrimenti verrà segnalato che il numero di carta inserito non è valido.

AggiornasaldoSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l'username dell'utente tramite token. Subito dopo verrà verificato se l'utente abbia un numero di carta associato, solamente se ha una carta associata potrà proseguire con l'aggiornamento del saldo e l'utente dovrà inserire un saldo che sia maggiore di 0 altrimenti verrà segnalato errore fino a che il saldo non è corretto. Dopo aver inserito un valore corretto per il saldo si andrà così ad aggiornare il saldo dell'utente.

CancellaprenotazioniSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l'username dell'utente tramite token. In seguito, sia andranno ad eliminare le prenotazioni, le richieste di carica e i pagamenti.

CancellatuttoSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l'username dell'utente tramite token. Poi vengono chiamate le Api relative alla cancellazione delle tabelle Reservation, DelayedReservation, Payment, ChargeRequest, Macchina, User.

CreaEModificaMacchinaSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l'username dell'utente tramite token. Si controlla poi che la targa inserita dall'utente sia valida, nel caso quindi l'utente inserisce il modello e i kw della batteria se l'auto è elettrica.

EliminaMacchinaSeq

Come prima cosa l'utente seleziona la targa della macchina a lui associata che vuole eliminare, dopo aver scelto la targa clicca il bottone appropriato per l'eliminazione della macchina, anche verrà richiamata l'API e si procederà all'eliminazione della macchina. Se l'utente non ha una macchina associata comparirà un messaggio che dirà che finché non aggiungerà una macchina non potrà eliminarne alcuna.

EsciPrimaSeq

Vengono mostrate a video tutte le prenotazioni che l'utente ha fatto e in quel caso l'utente clicca il bottone riferito alla riga della prenotazione riferita alla macchina che vuole che esca prima.

LoginSeq

Dopo che l'utente avrà inserito la mail e la password corretti verrà chiamata l'API relativa al login che permetterà quindi l'accesso alle funzioni dell'applicazione all'utente ora che si è autenticato e ritornerà un token il quale sarà salvato nel LocalStorage in modo da essere utilizzato per le chiamate delle API che richiederanno nell'intestazione il token.

OccupazioneSeq

Verrà visualizzato lo stato attuale del parcheggio chiamando l'API per l'occupazione del parcheggio.

PagaRitardoSeq

Come prima cosa si ottiene il token dell'utente, poi vengono caricate le prenotazioni in ritardo che sono associate all'utente, l'utente selezionerà la prenotazione la prenotazione che vuole di cui vuole pagare la multa e prima di poter pagare verrà controllato se ha saldo sufficiente, e solamente nel caso in cui ce l'abbia, ovvero che sia almeno uguale al valore della multa, potrà pagare la multa, altrimenti dovrà prima aggiornare il saldo.

RegistraSeq

L'utente inserirà username e e-mail che devono essere unici e non già esistenti ed inoltre dovrà inserire una password che sia valida.

RegistraAdminSeq

L'utente inserirà username e e-mail che devono essere unici e non già esistenti ed inoltre dovrà inserire una password che sia valida. Questa funzione può essere eseguita solamente da un ADMIN.

SetRitardoSeq

Verranno caricate le prenotazioni esistenti e l'ADMIN selezionerà la prenotazione a cui vuole assegnare un ritardo.

SpostaSosteSeq

Al click del bottone le soste che sono state ultimate verranno salvate all'interno dello Storico ed è eliminate dalla tabella Reservation, se la prenotazione era in ritardo viene anche salvata all'interno della DelayedReservation

AllPrenotazioniSeq

Una volta cliccato nella tendina Utente il bottone Visualizza Prenotazioni, apparirà una pagina in cui verranno caricate le prenotazioni attive fatte dall'utente (se ci sono), con ID della prenotazione, Targa della macchina, Giorno, Ora Inizio, Ora fine, se l'utente vuole che avvenga la ricarica del veicolo, e se vi è assegnato un ritardo nell'uscita del parcheggio.

Stampa-PagamentiSeq

Questa operazione può essere solamente svolta da un utente ADMIN, e permette di stampare tutti i pagamenti che sono stati fatti da qualunque utente, filtrandoli per ruolo dell'utente (BASE/PREMIUM/ADMIN) oppure per tipo di prenotazione (Ricarica/Sosta)

Noi abbiamo sviluppato questo sistema su macOS utilizzando come IDE IntelliJ_IDEA per la parte di Backend in Java e JetBrains Rider per la parte di Frontend.

ISTRUZIONI

1. POSIZIONE DEI FILE ALL'INTERNO DELLE CARTELLE

- a. Scaricare i seguenti file: parcheggio.zip, ParcheggioGUI.zip, Hue-Emulator.zip e Sbarra.zip.
- b. Crea una cartella con un nome a scelta.
- c. Scompattare parcheggio.zip e ParcheggioGUI.zip e riporli all'interno della cartella appena creata.
- d. All'interno della cartella appena scompattata avremo __MACOSX e poi un'altra cartella chiamata anch'essa parcheggio.
All'interno di quest'ultima inserisci i file scompattati di Sbarra.zip e Hue-Emulator.zip (insieme quindi a src, data e target).

2. PARCHEGGIO (BACK-END)

- a. Aprire all'interno di IntelliJ IDEA la cartella parcheggio (quella con al suo interno src, sbarra e Hue-Emulator) e andare nel terminale (in basso a sinistra). Quando si è lì fare, controllare che il terminale sia posizionato sulla cartella parcheggio (quella con dentro src) e fare il comando "mvn clean package" così da creare una cartella target una volta finito (in arancione), successivamente controllare che dentro ci siano 3 file con estensione .jar.
- b. All'avvio del programma verrà fatto partire in automatico:
 - Mosquitto, con le impostazioni per l'autenticazione in un terminale.
 - L'Emulatore, che necessiterà di premere il tasto start e il bottone tondo in basso a sinistra di esso, senza questo passaggio il programma non andrà avanti.
 - La Sbarra in un altro terminale separato
 - L'MWBot in un altro terminale separato, esso andrà avanti in automatico a controllare che nel database ci siano richieste di ricarica, se ne trova ricarica quella macchina, altrimenti aspetta 1 minuto e ricontrolla.
 - Successivamente andiamo a iscriverci all'MQTT aprendo un altro terminale nuovo ed eseguire linea di comando "mosquitto_sub -h localhost -p 1883 -t '#' -u 'Username' -P 'Password'", inserendo "#" ci iscriveremo a tutti i topic, se vogliamo selezionarlo lo sostituiamo e inseriamo il topic desiderato, per completare l'iscrizione inseriamo utente e password validi.
(Username = Admin, Password = Admin)
- c. Poi per accedere al database andare su: <https://localhost:8443/h2-console/login.jsp> dopo che Parcheggio è stato avviato. Inserire queste

*Nel caso uscissero degli errori nella console di IntelliJ IDEA ricaricare le dipendenze Maven.

impostazioni per accedere al database:

The screenshot shows a 'Login' dialog box with the following fields:

- Saved Settings: Generic H2 (Server)
- Setting Name: Generic H2 (Server) (with Save and Remove buttons)
- Driver Class: org.h2.Driver
- JDBC URL: jdbc:h2:file:/data/demo
- User Name: admin
- Password: (redacted)
- Buttons: Connect and Test Connection

Password = admin

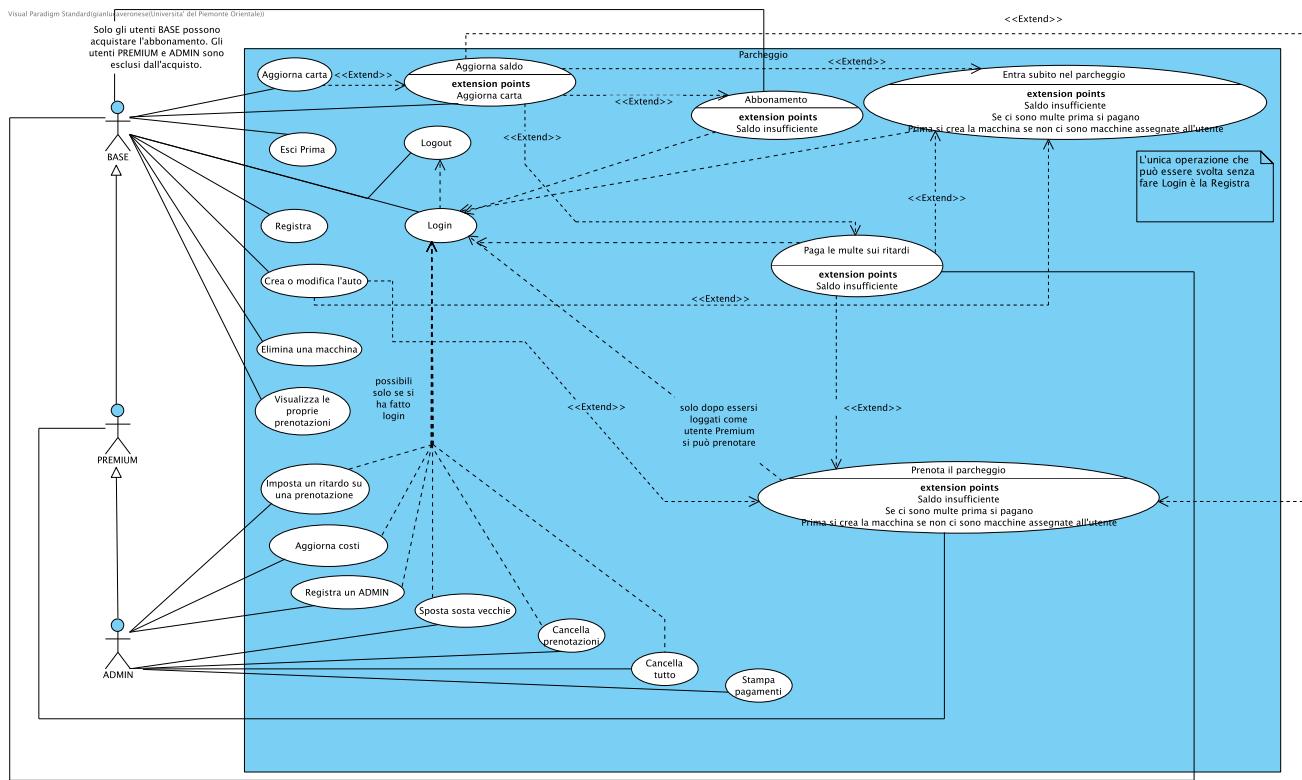
3. PARCHEGGIOGUI (FRONT-END)

- a. Aprire la cartella ParcheggioGUI all'interno di Rider.
- b. Aprire il terminale e inserire il comando "cd ParcheggioGUI", così da spostarsi nella cartella parcheggiGUI.
- c. Aprire Rider e all'interno del terminale eseguire questi 2 comandi:
 - i. dotnet add package Blazored.LocalStorage
 - ii. dotnet add package System.IdentityModel.Tokens.Jwt
 - iii. dotnet clean
 - iv. dotnet build
- d. In seguito dopo che è stato fatto tutto questo avviare l'applicazione tramite il tasto play oppure avviandola tramite il comando dotnet run e andando al link <http://localhost:5014/>. (in automatico verrà aperto google chrome su questo link)

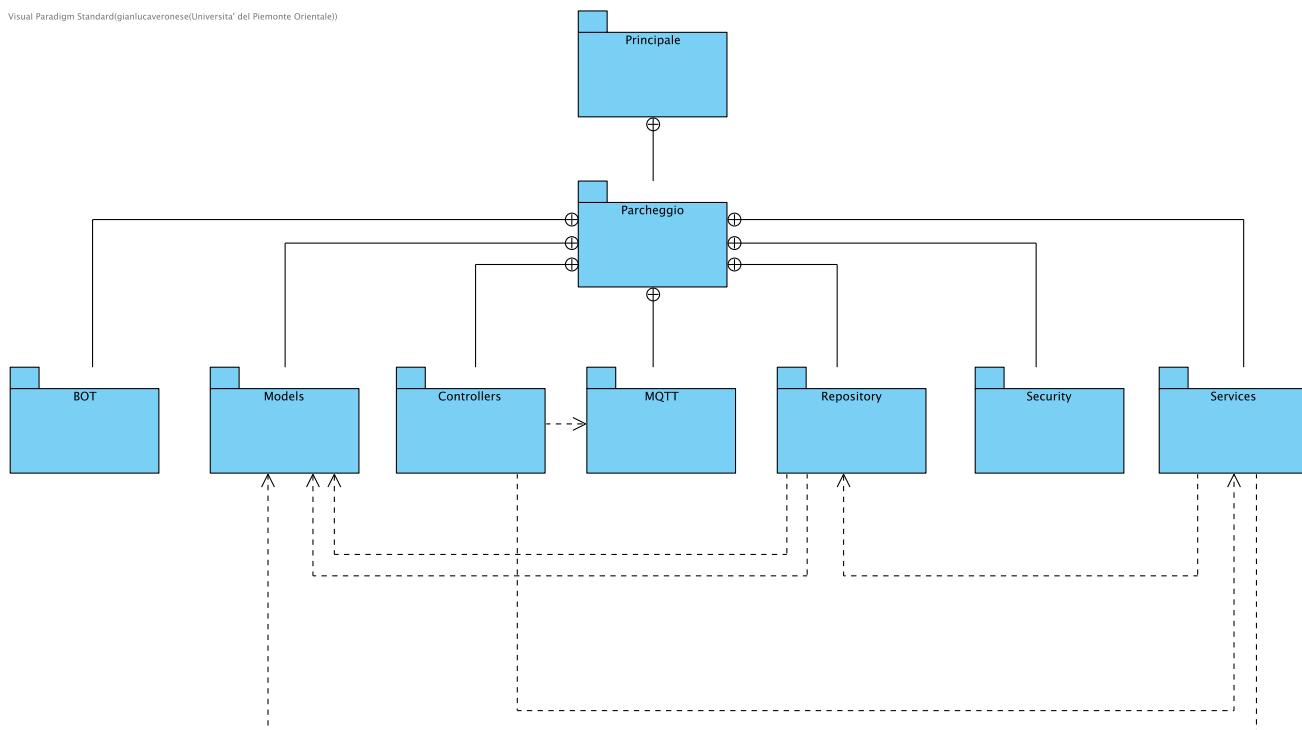
DIAGRAMMI

FEROLO NICOLA VERONESE

1. DiagrammaCasiD'uso

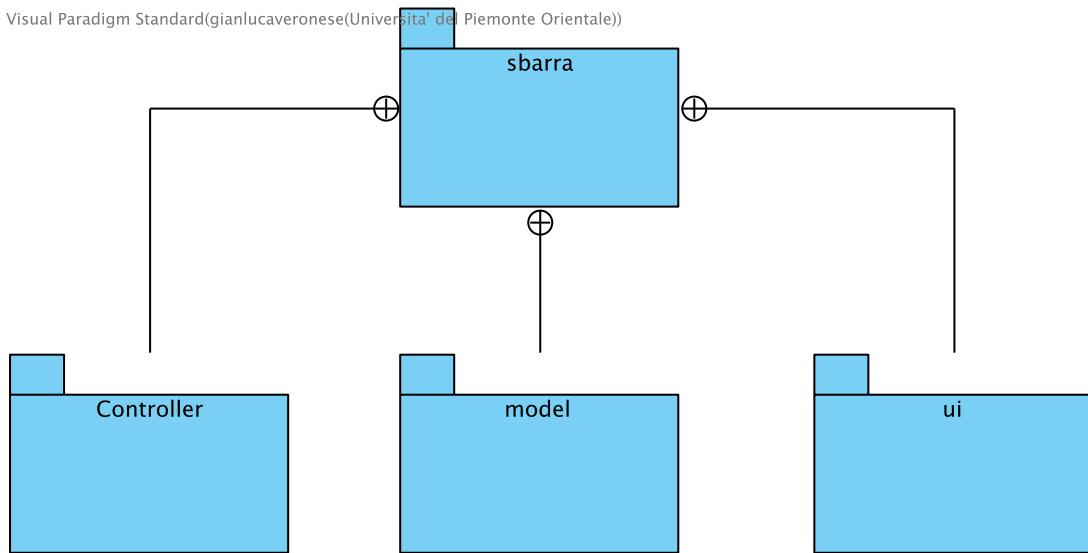


2. Packageparcheggio



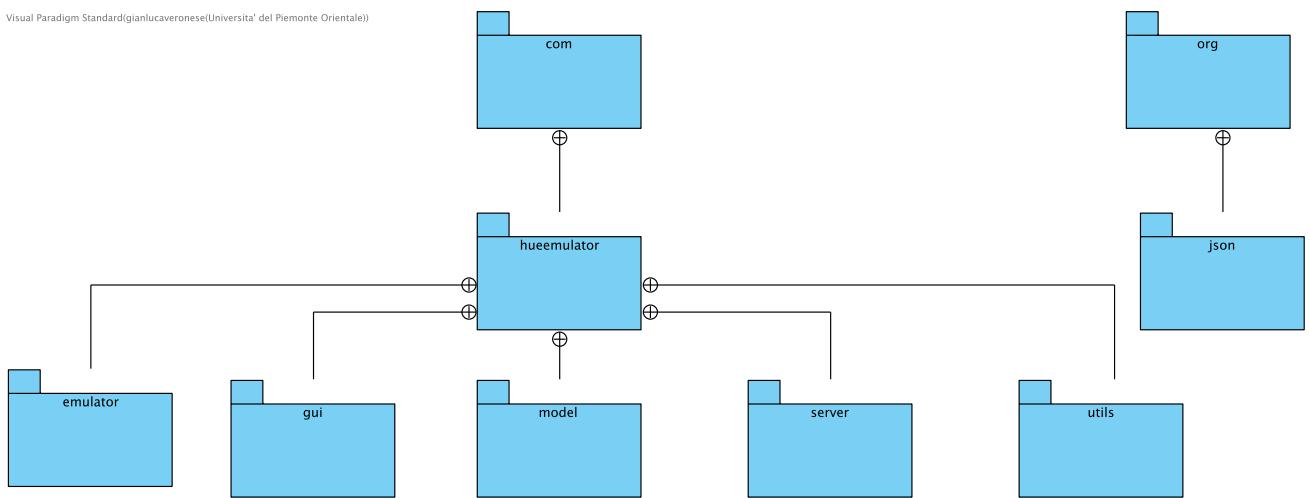
3. Packagesbarra

Visual Paradigm Standard(gianlucaveronese(Universita' del Piemonte Orientale))



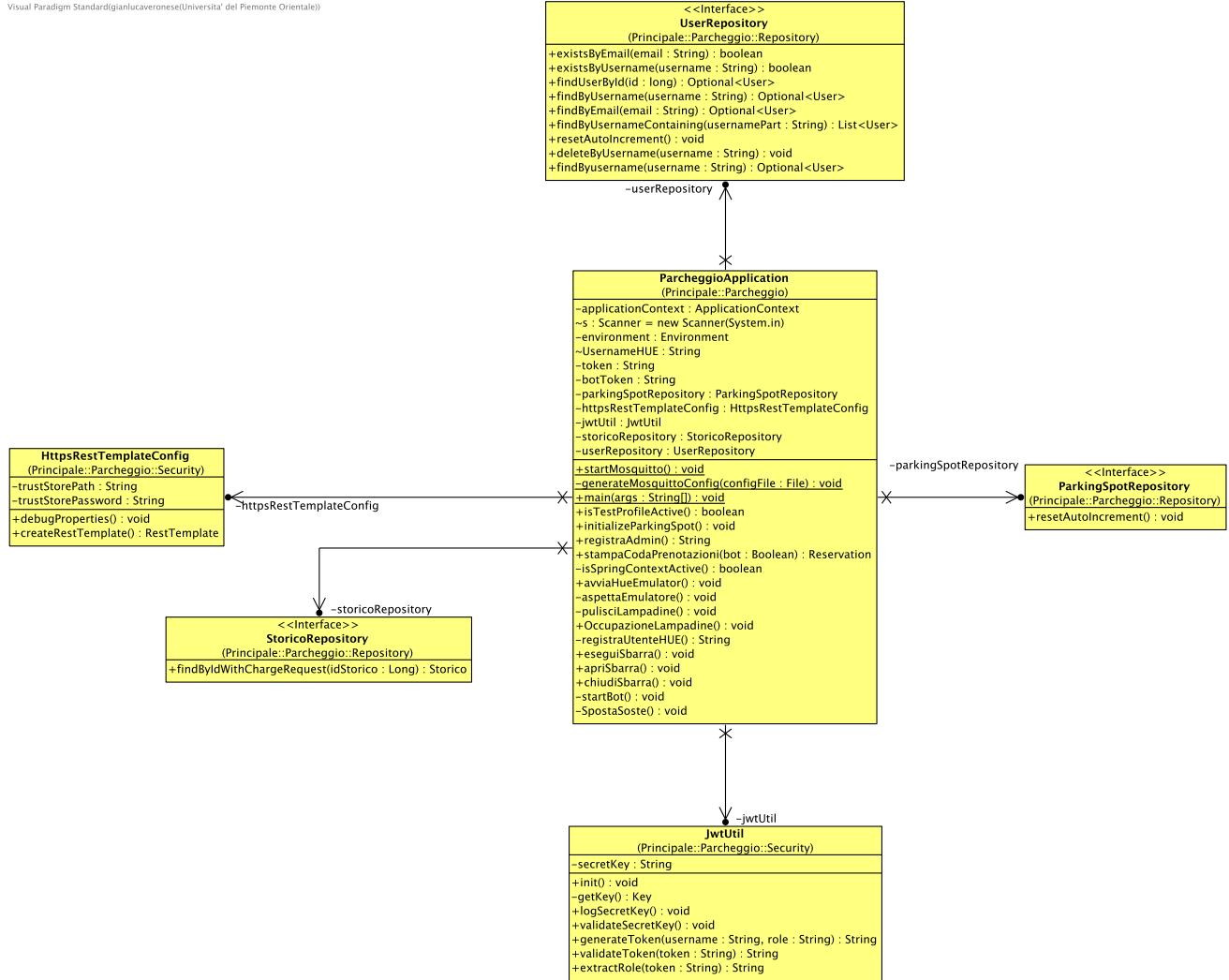
4. PackageHueEmulator

Visual Paradigm Standard(gianlucaveronese(Universita' del Piemonte Orientale))

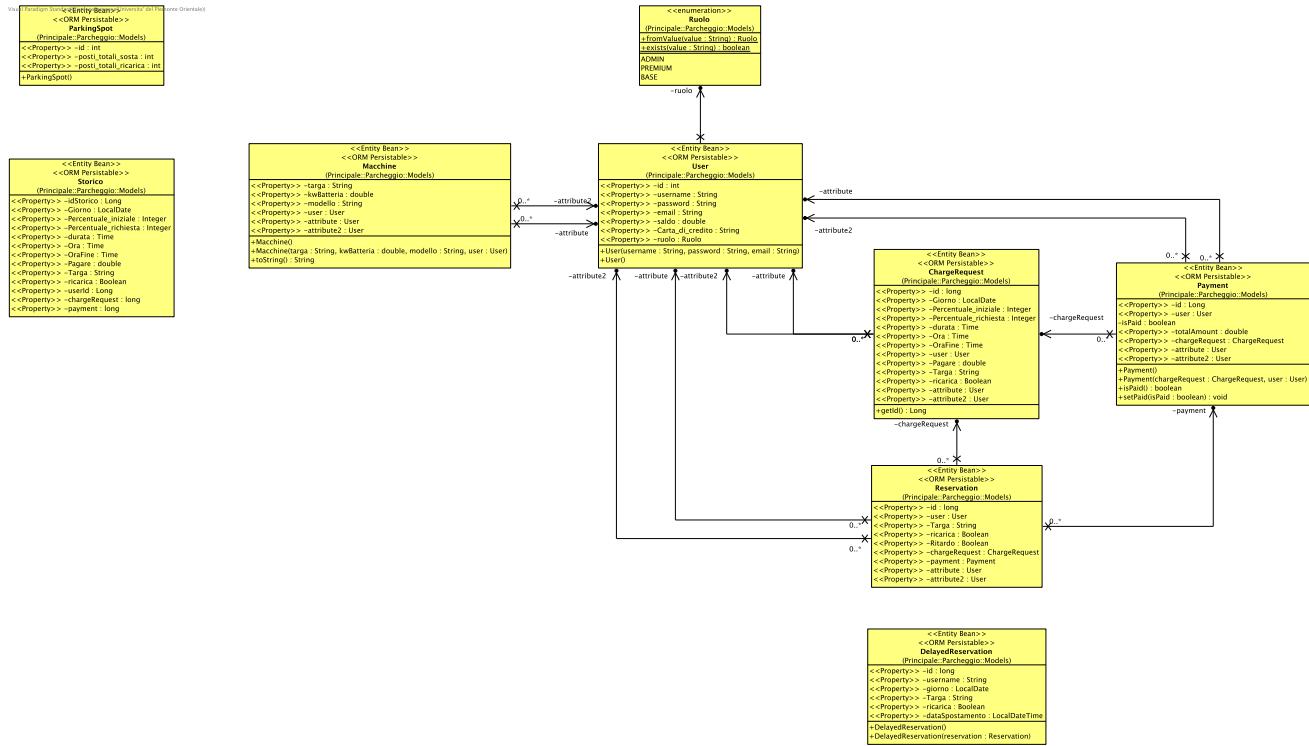


5. Principale.Parcheggio

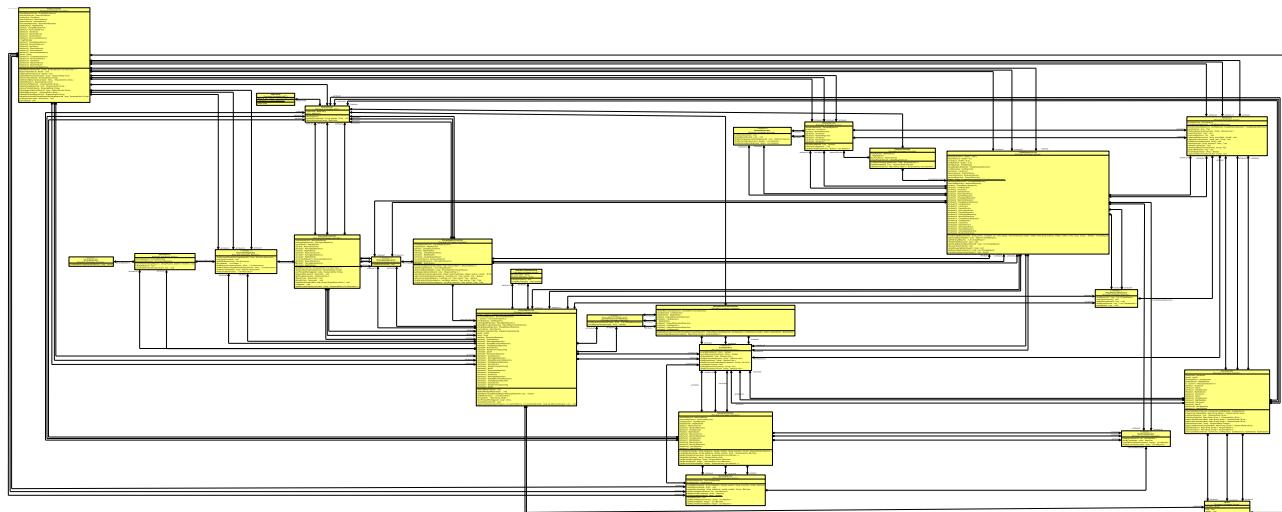
Visual Paradigm Standard (gianlucaveronese@universita' del Piemonte Orientale)



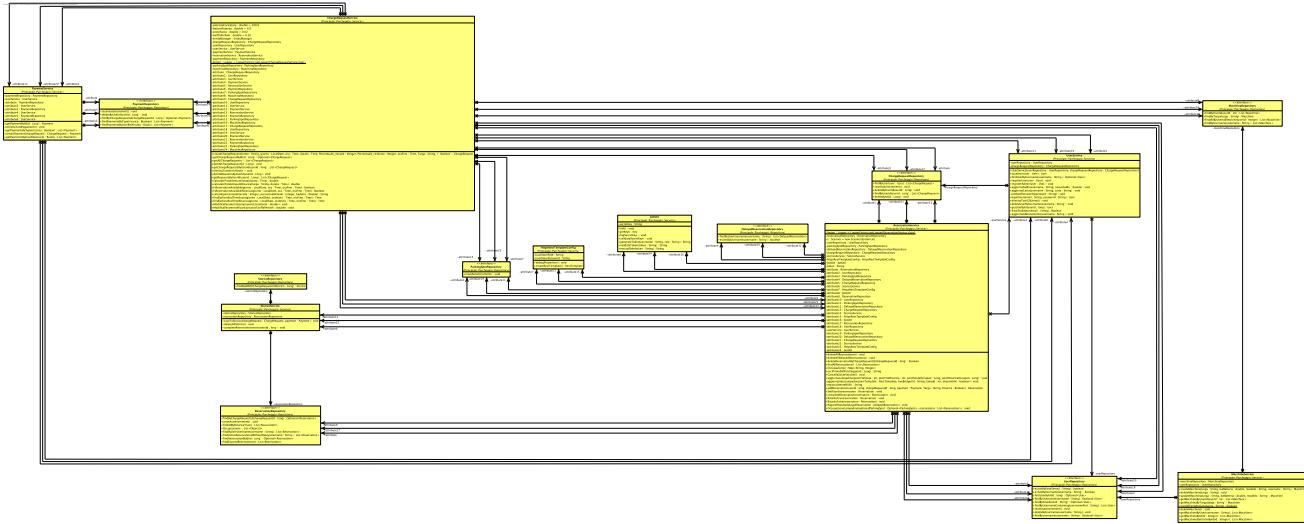
6. Principale.Parcheggio.Models



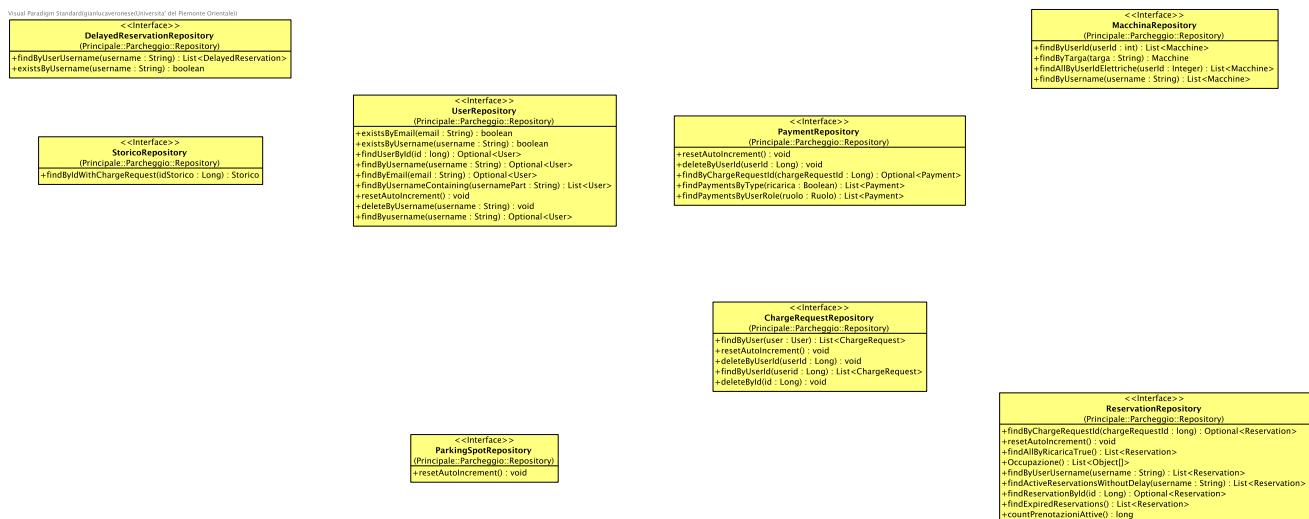
7. Principale.Parcheggio.Controllers



8. Principale.Parcheggio.Services

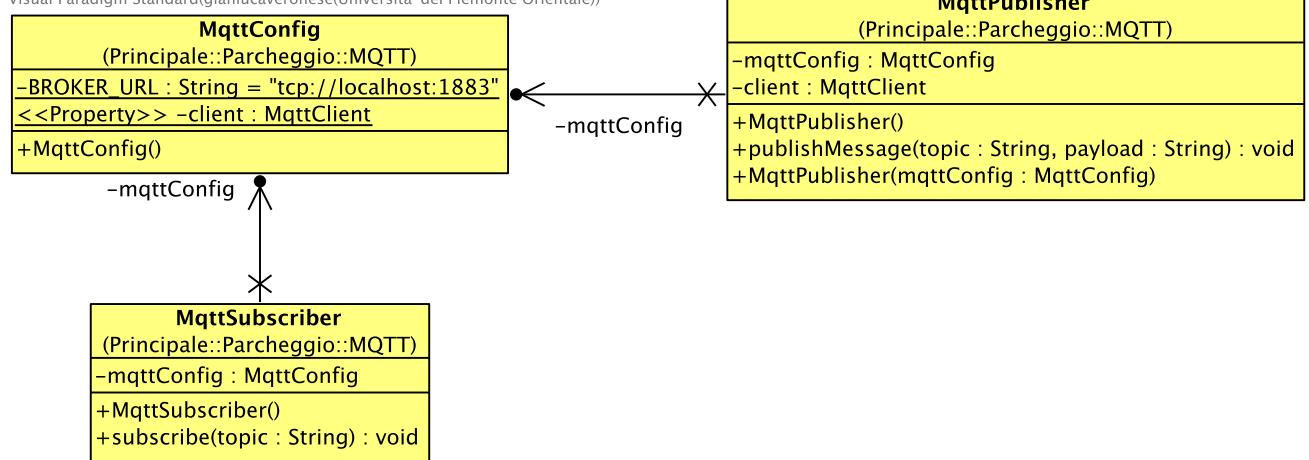


9. Principale.Parcheggio.Repository

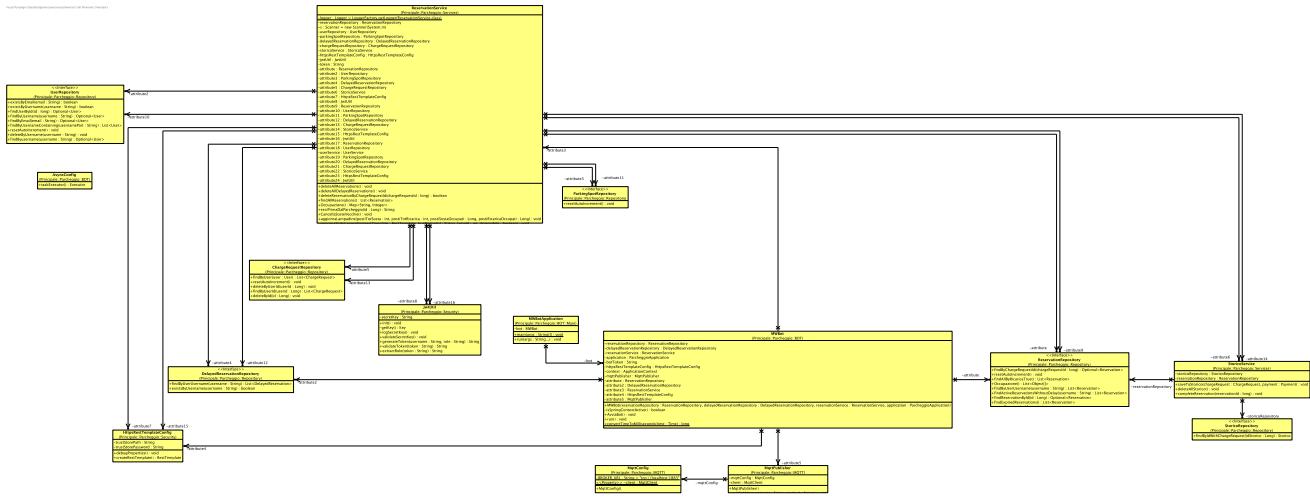


10. Principale.Parcheggio.MQTT

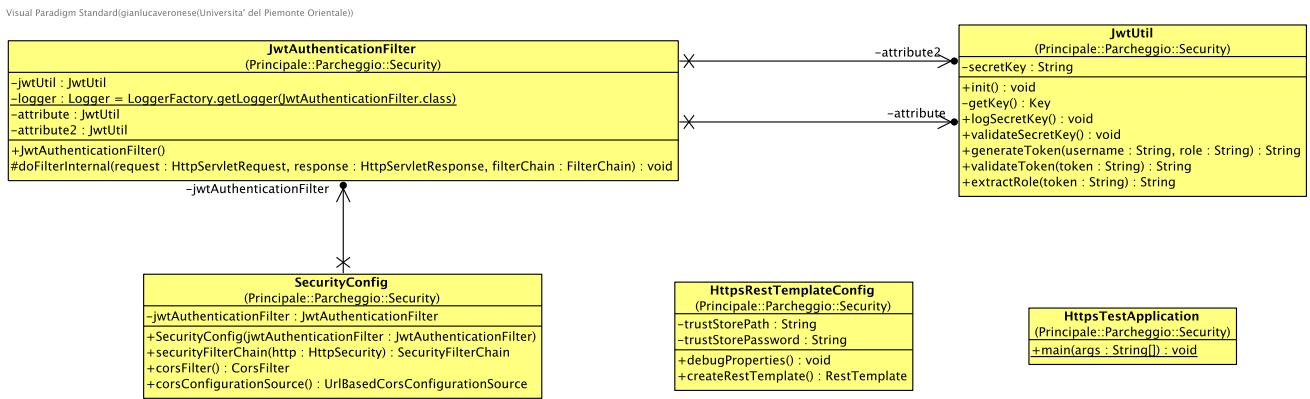
Visual Paradigm Standard(gianlucaveronese(Universita' del Piemonte Orientale))



11. Principale.Parcheggio.BOT

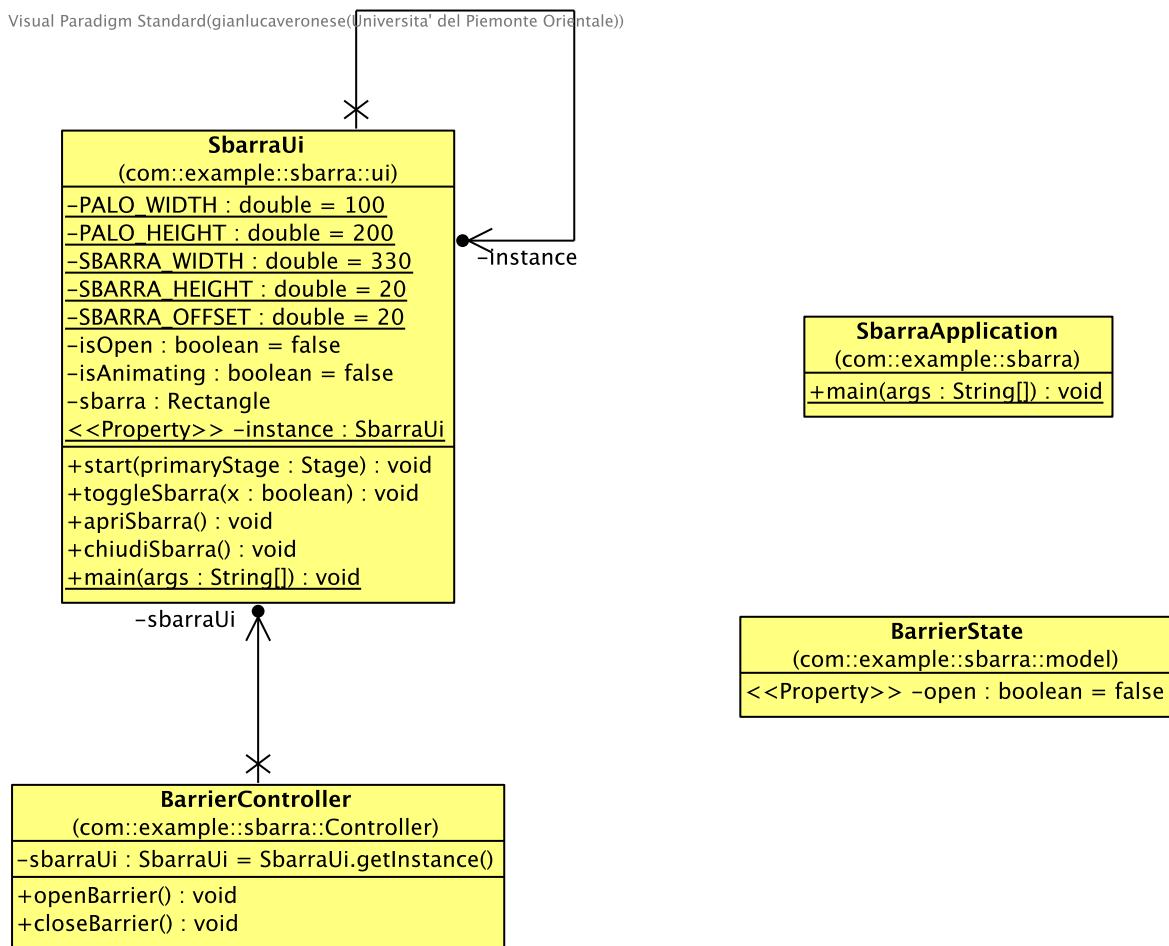


12. Principale.Parcheggio.Security

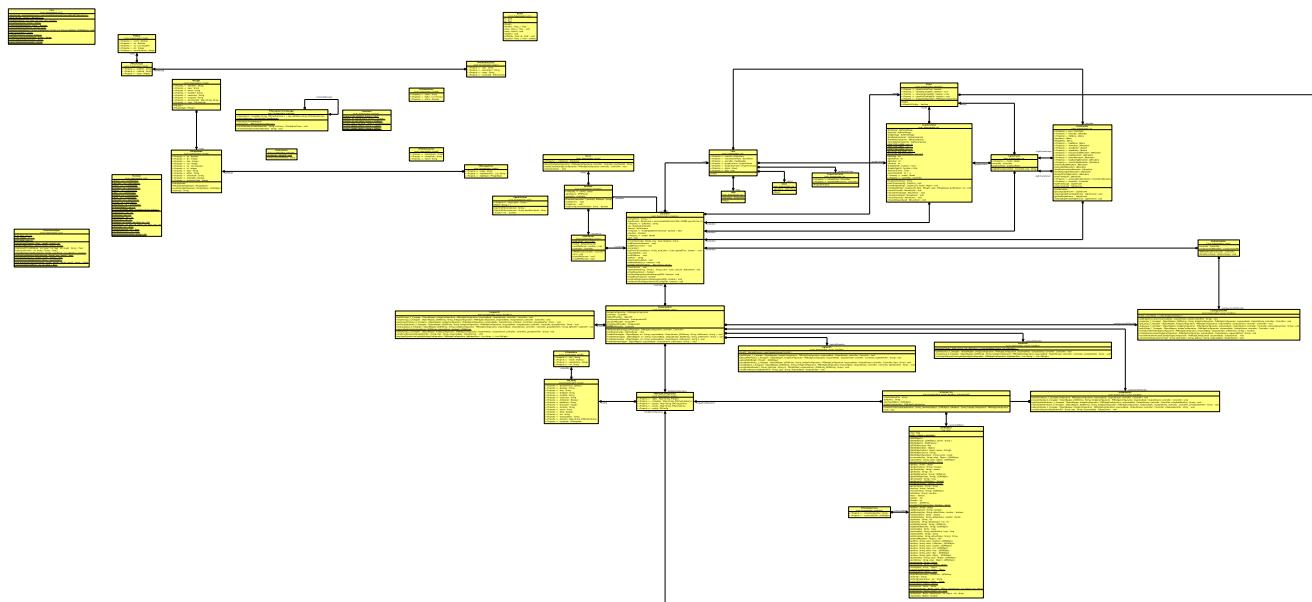


13. com.example.sbarra

Visual Paradigm Standard(gianlucaveronese(Universita' del Piemonte Orientale))



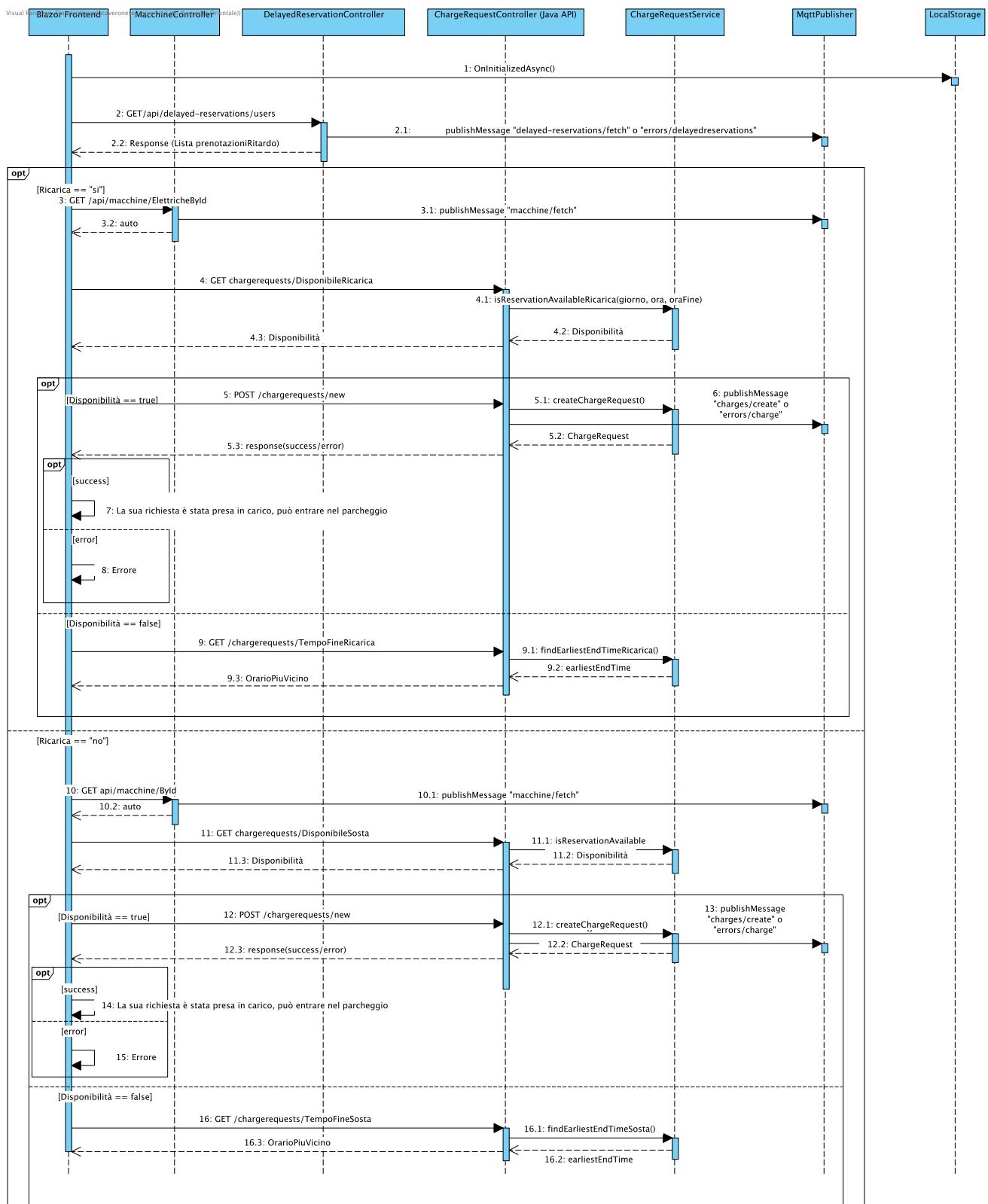
14. com.hueemulator



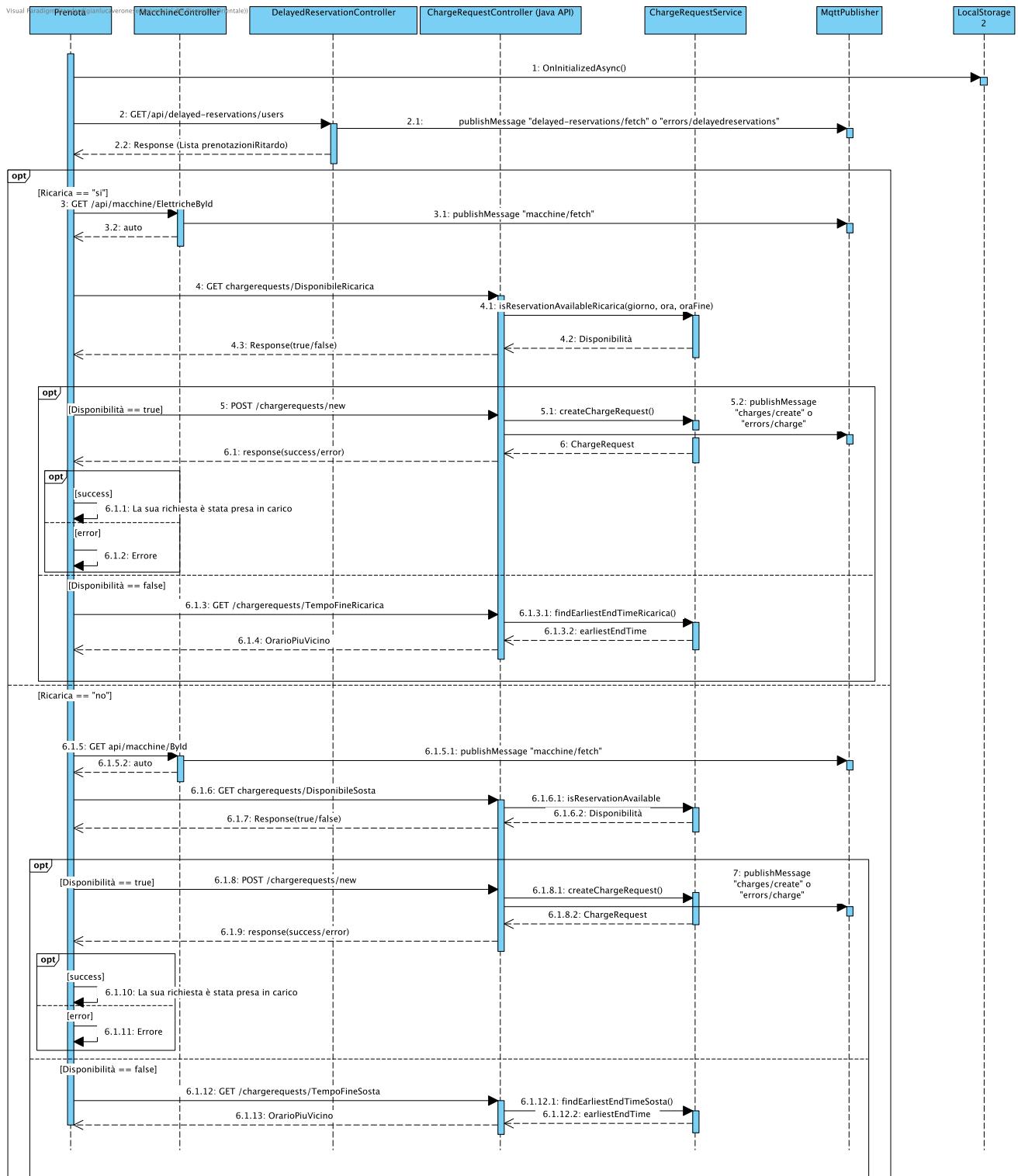
DIAGRAMMI DI SEQUENZA

FEROLO NICOLA VERONESE

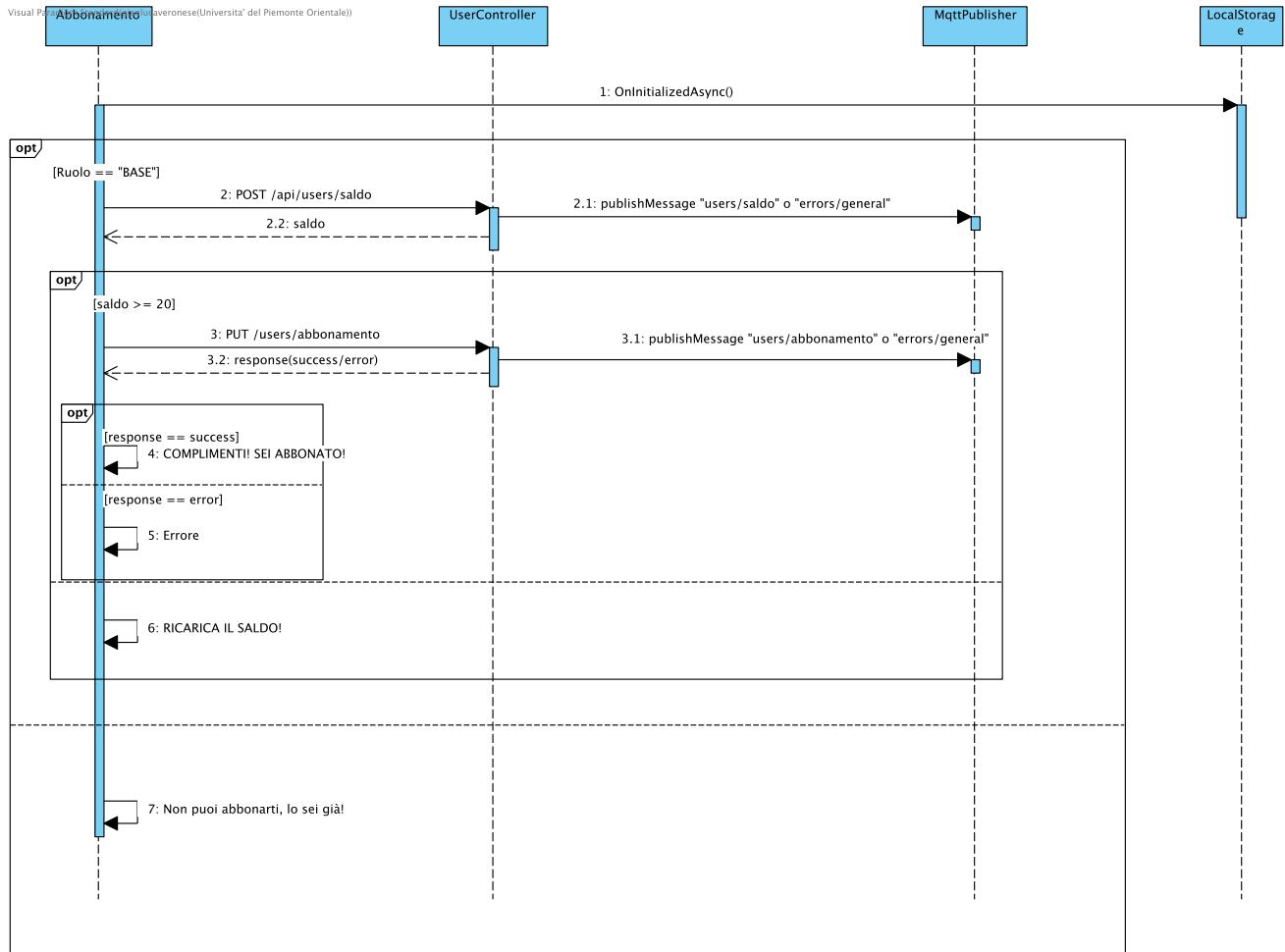
1. EntraOraSeq



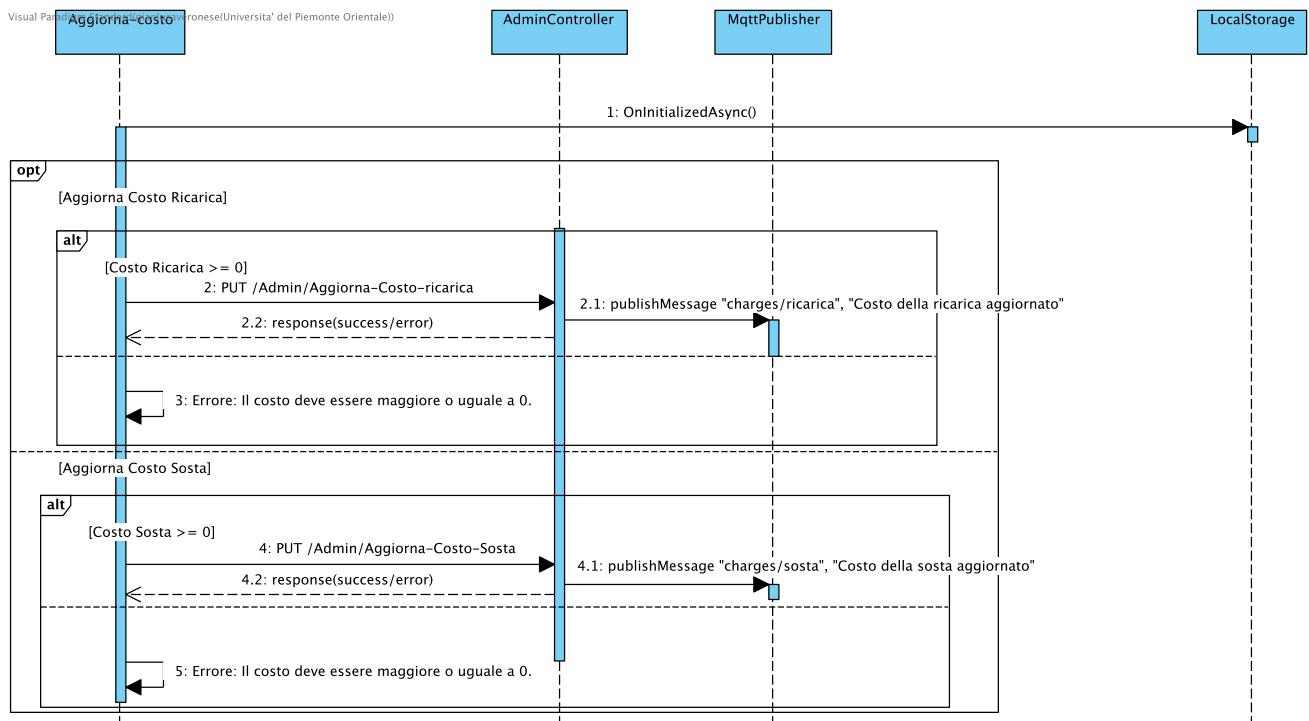
2. PrenotaSeq



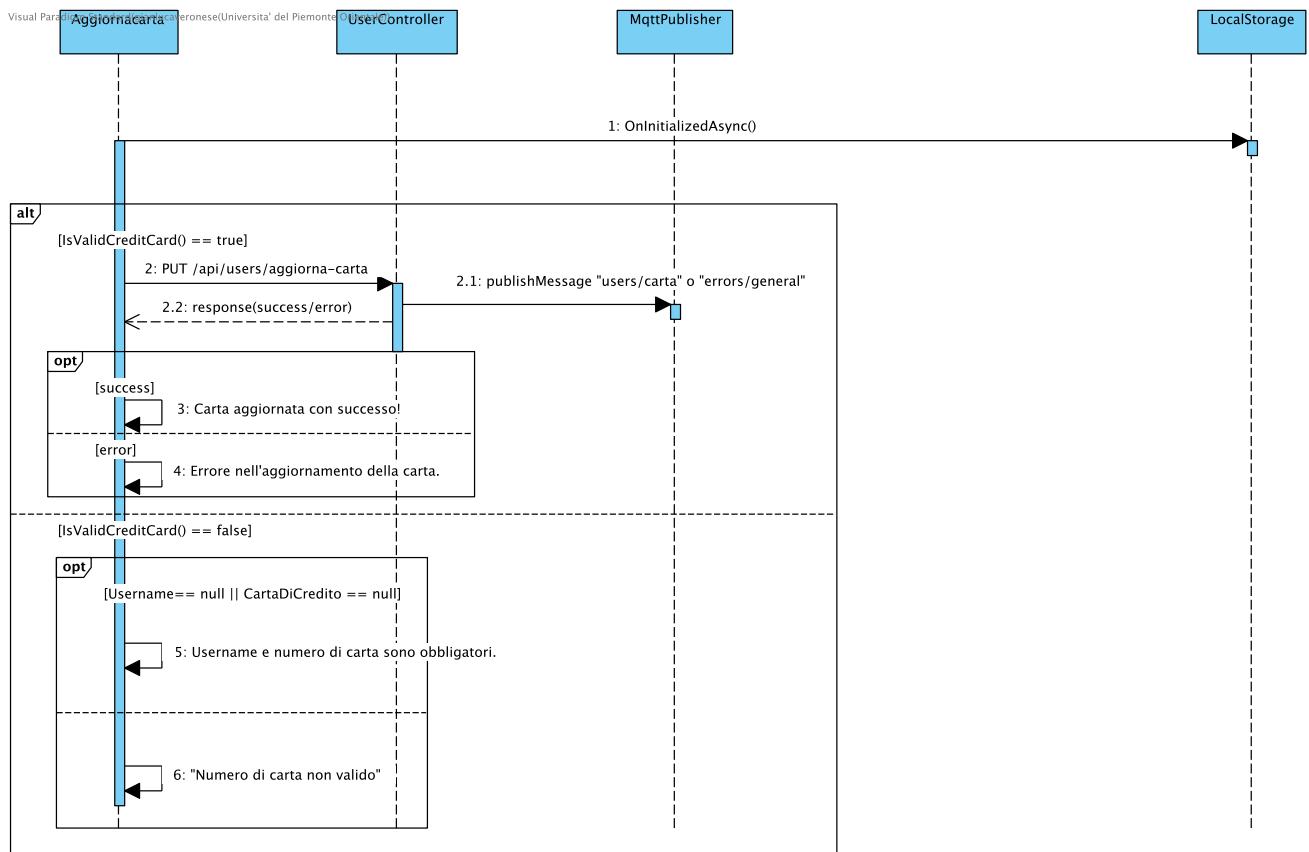
3. AbbonamentoSeq



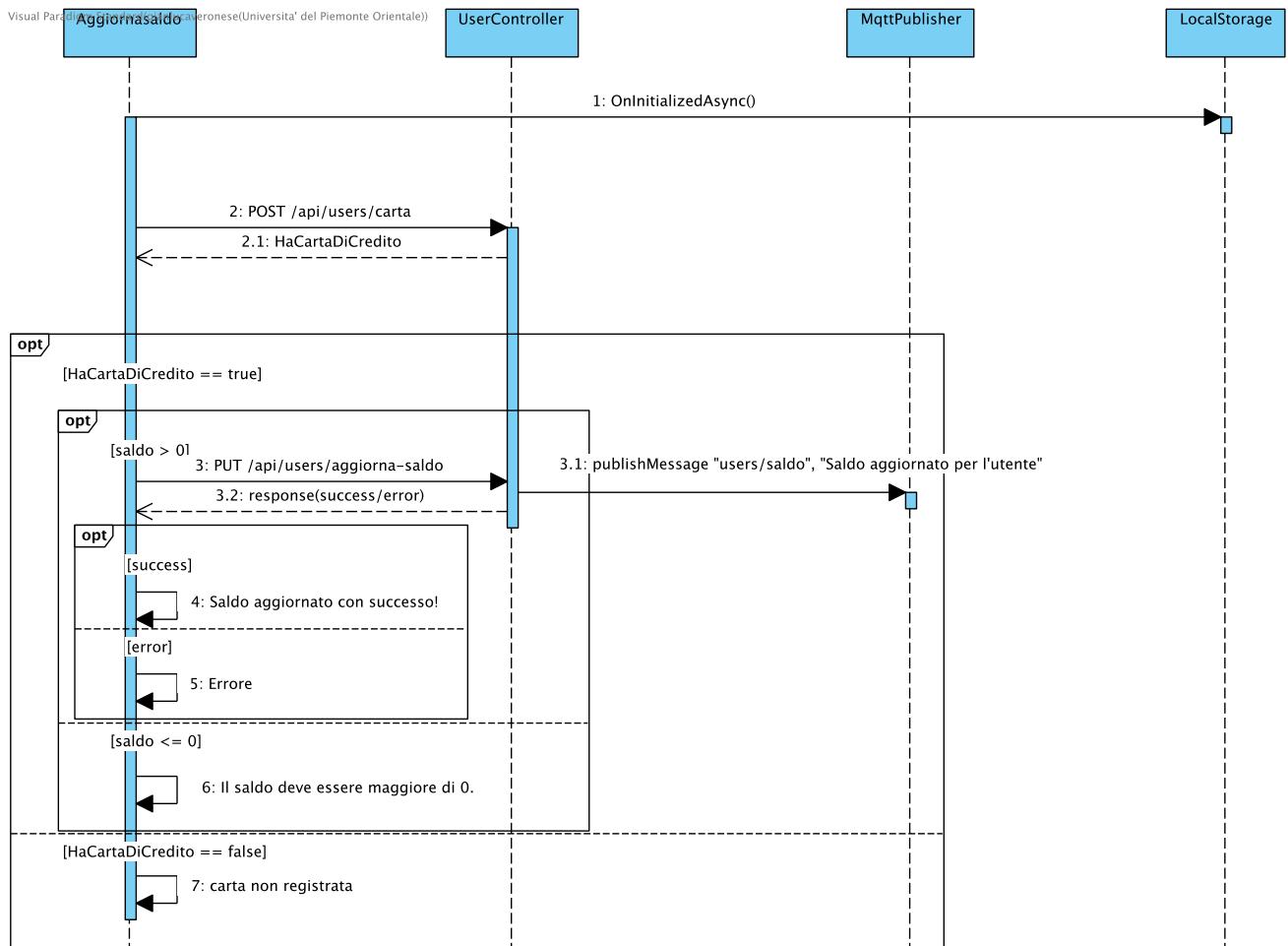
4. Aggiorna-costoSeq



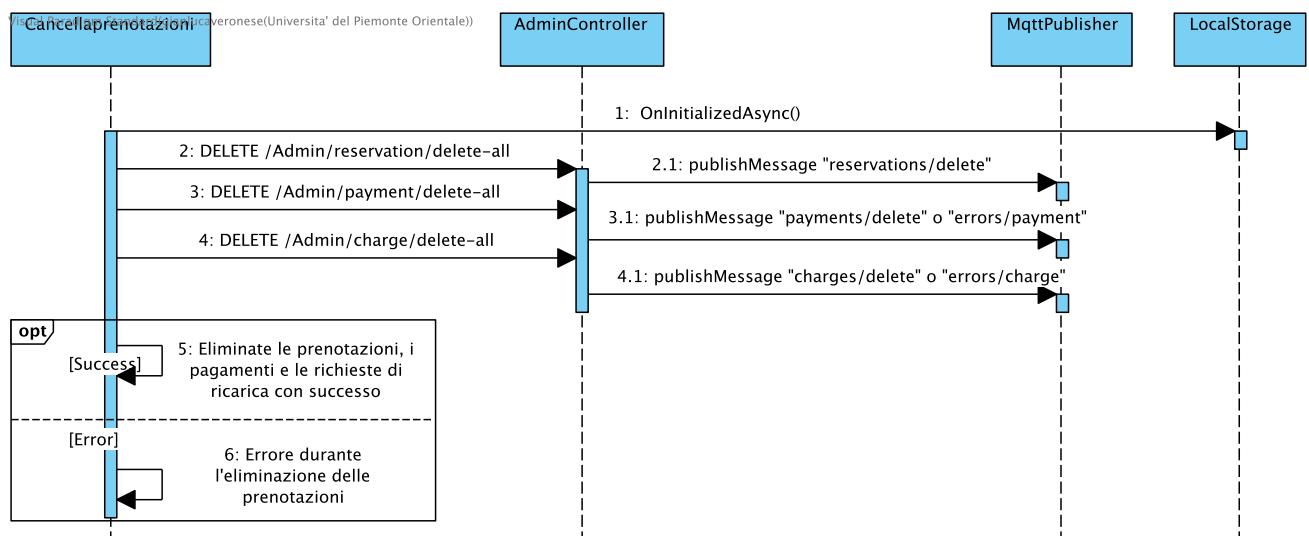
5. AggiornaCartaSeq



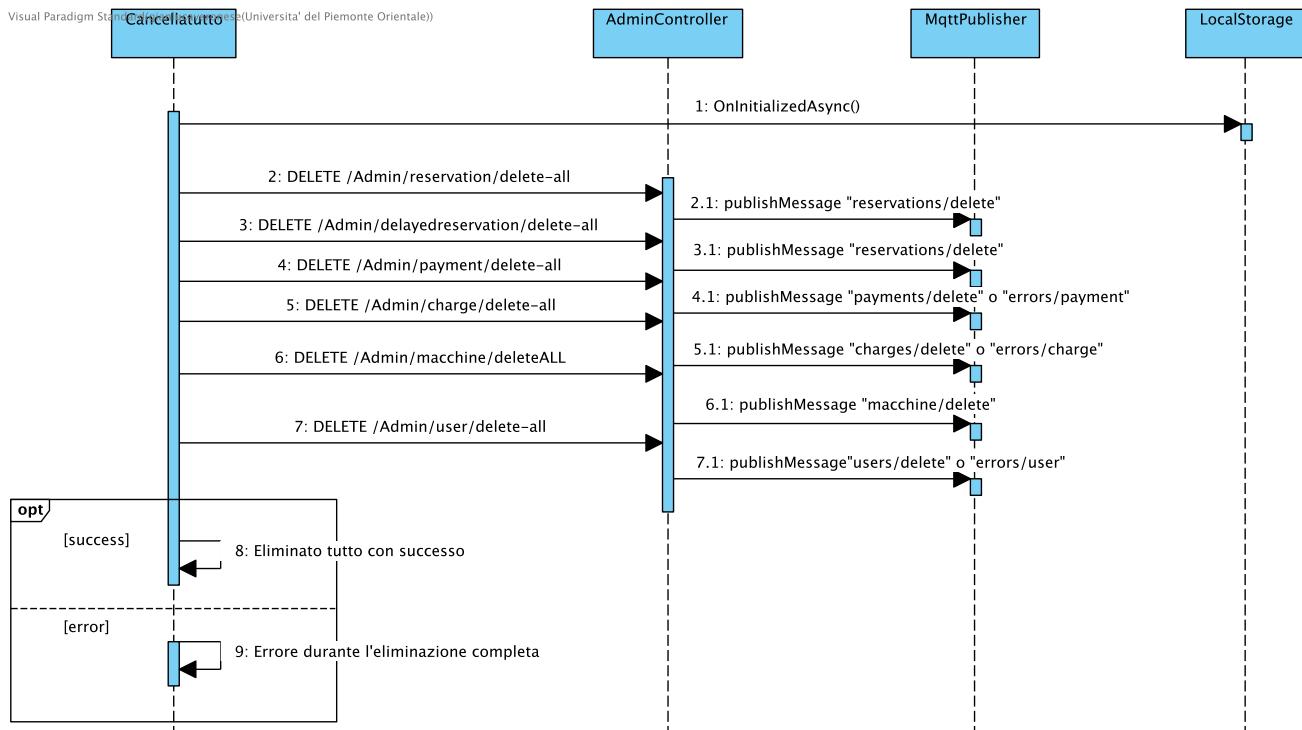
6. AggiornasaldoSeq



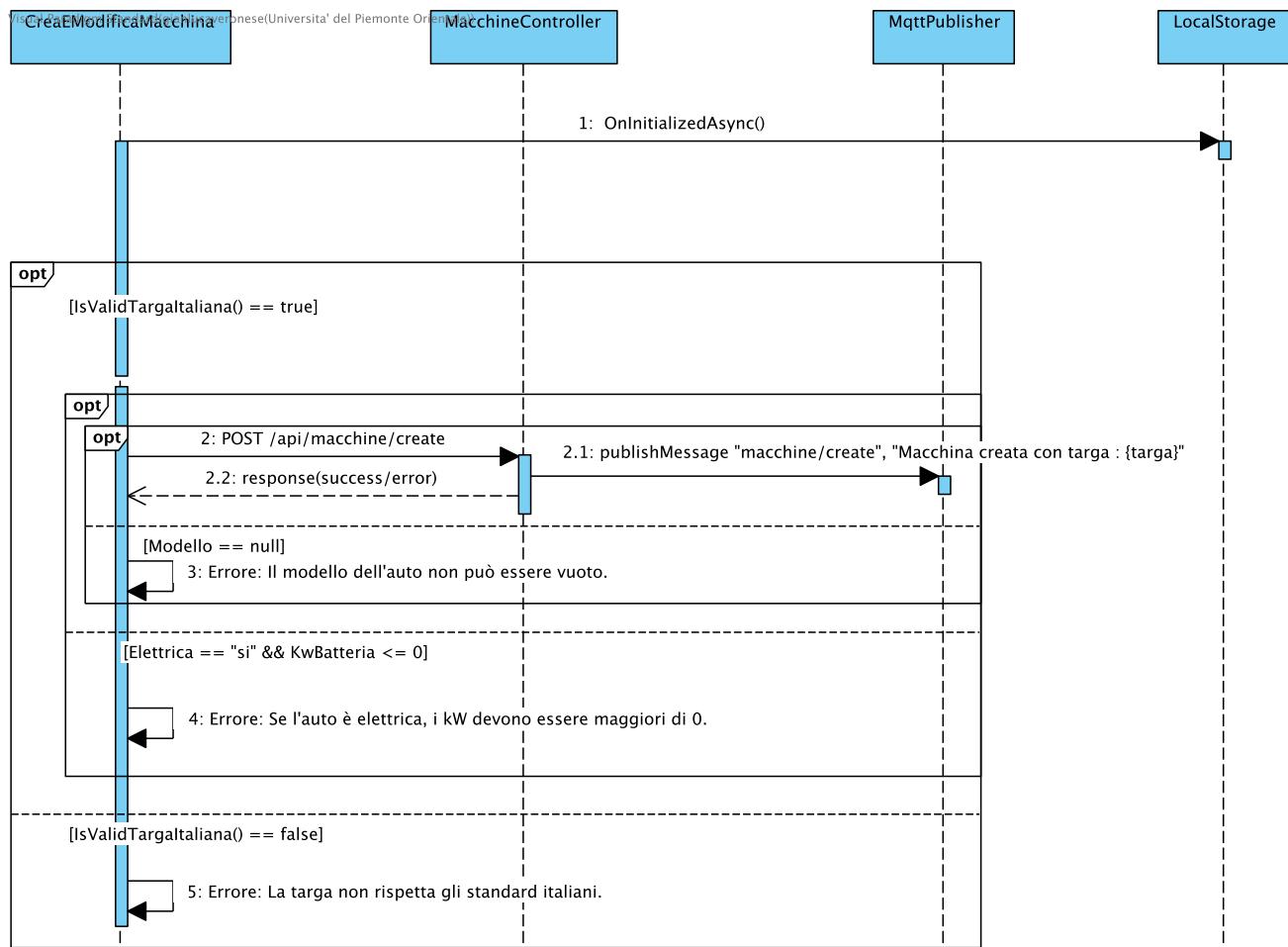
7. CancellaprenotazioniSeq



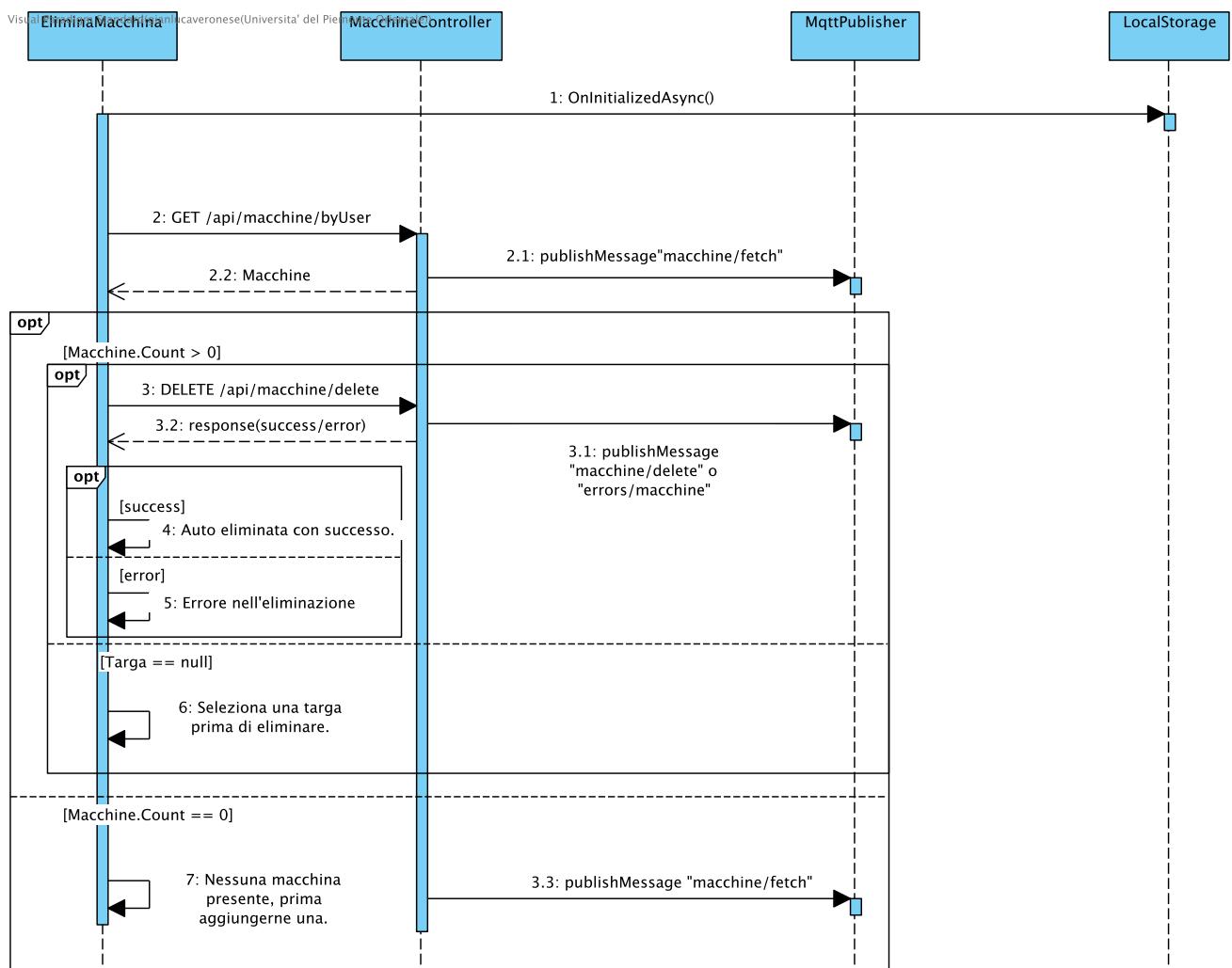
8. CancellatuttoSeq



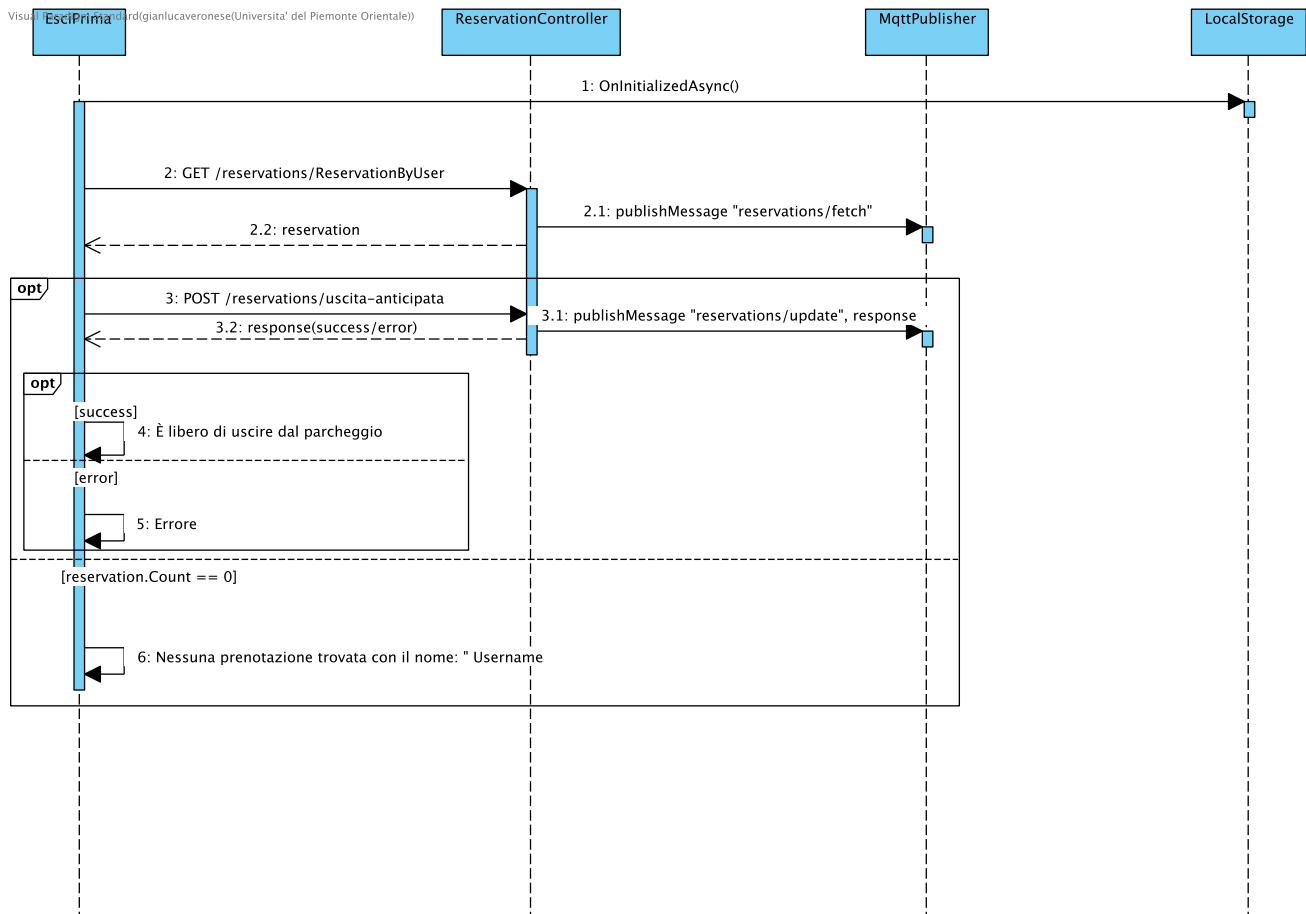
9. CreaEModificaMacchinaSeq



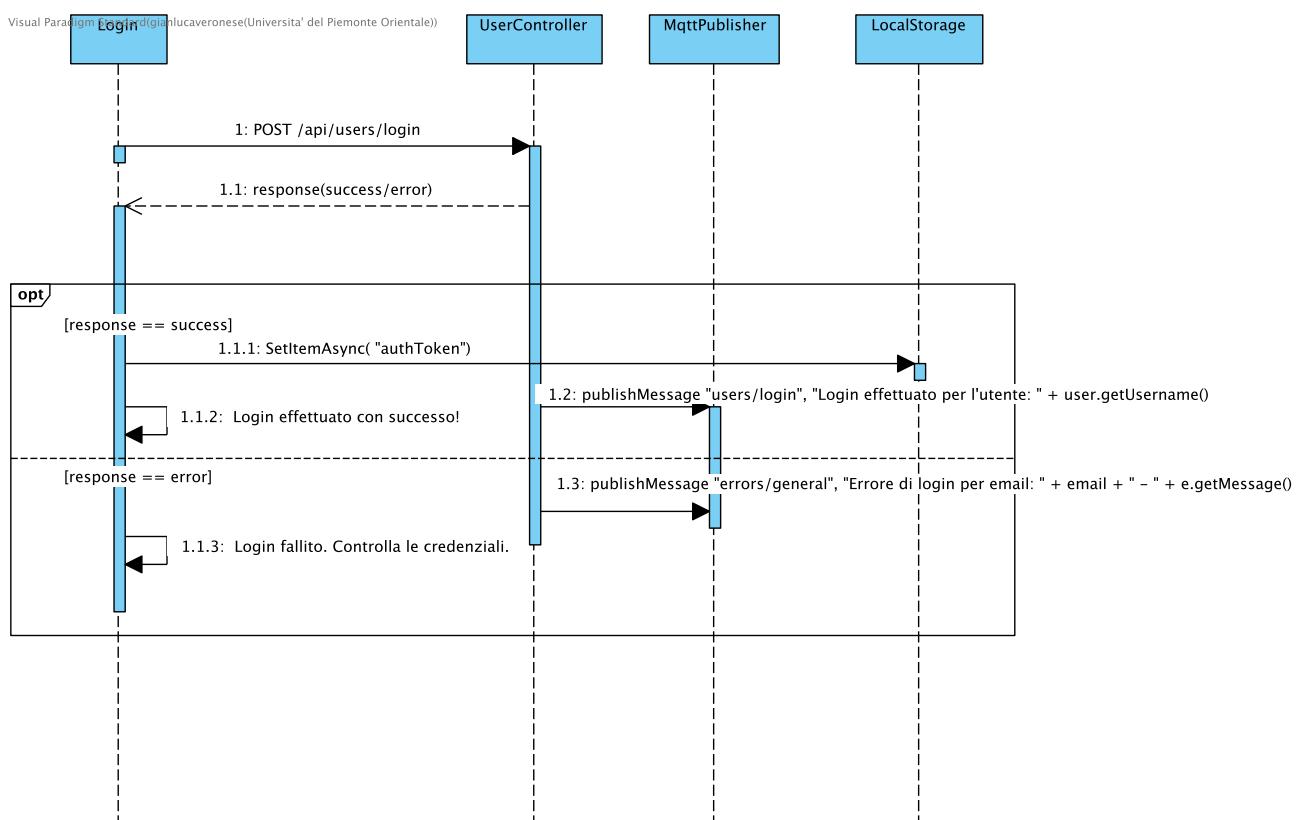
10. EliminaMacchinaSeq



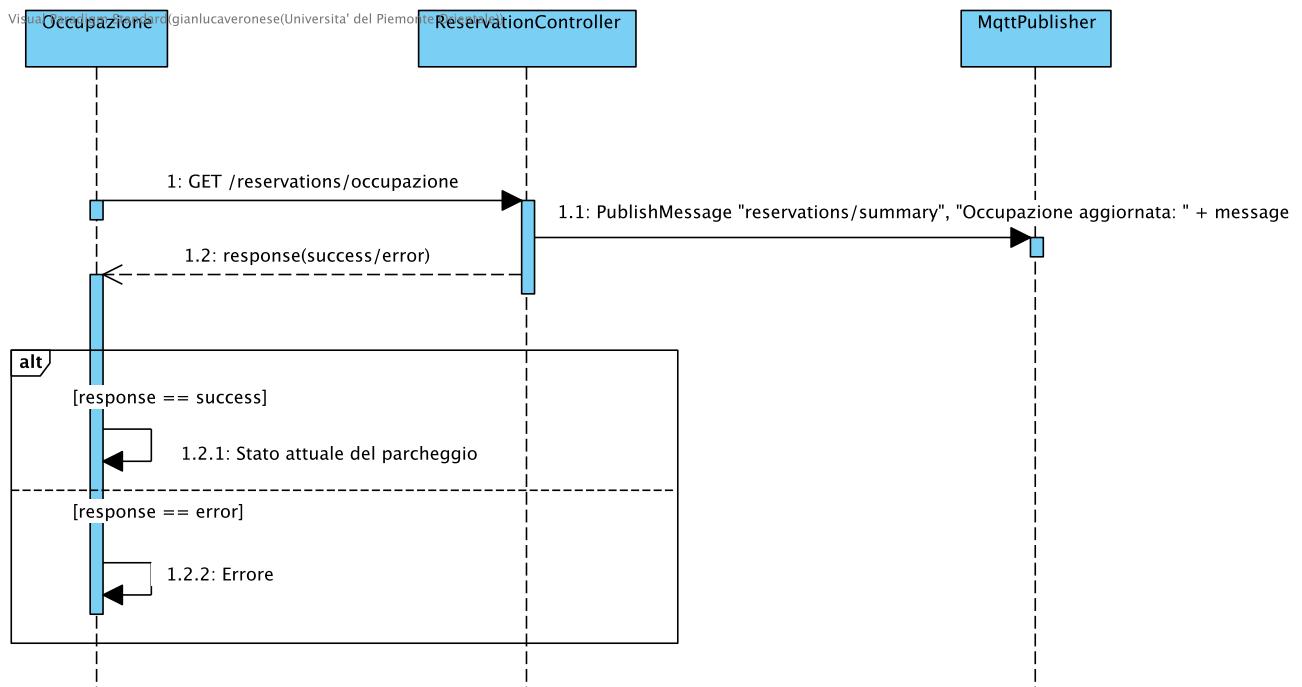
11. EsciPrimaSeq



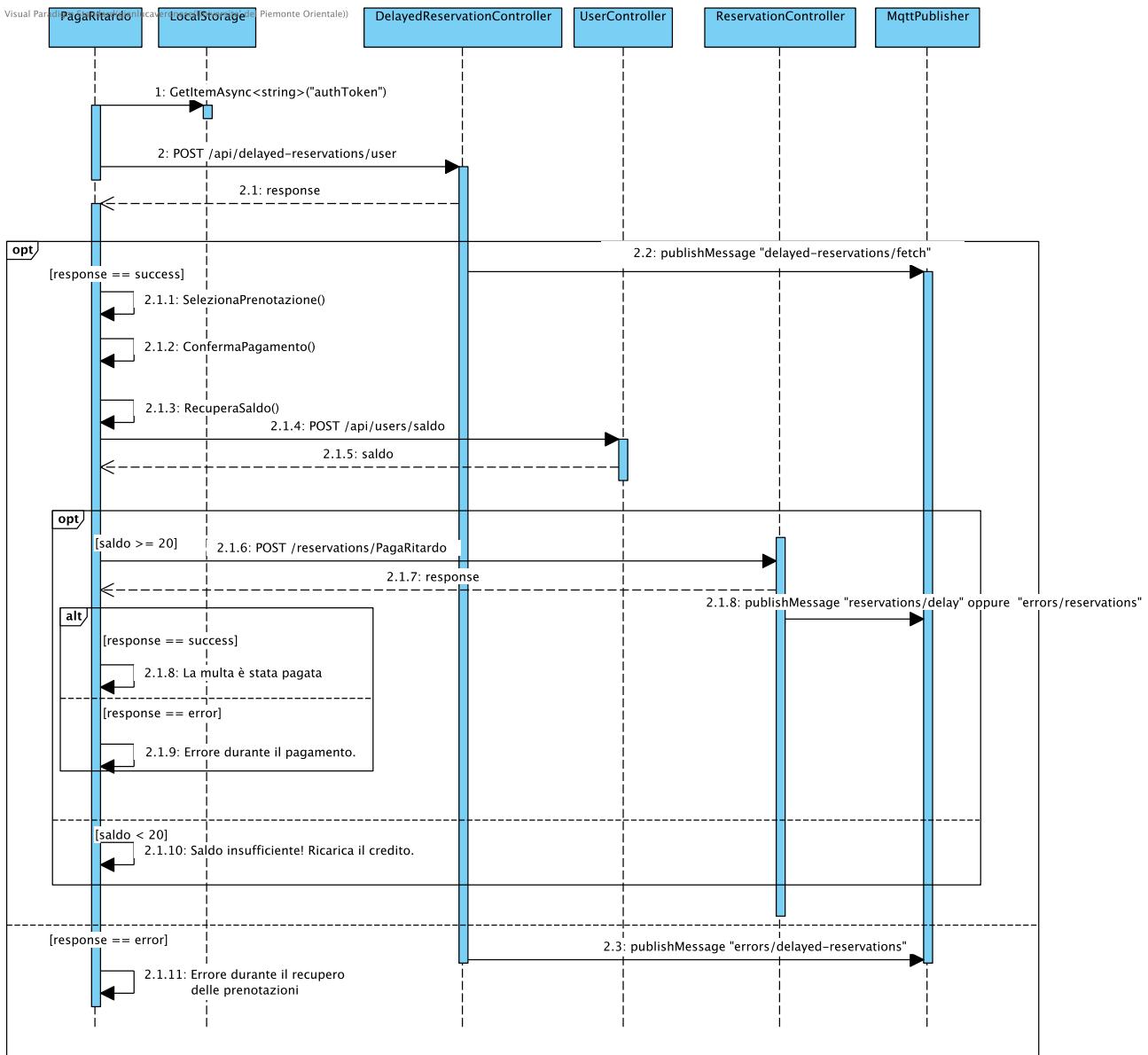
12. LoginSeq



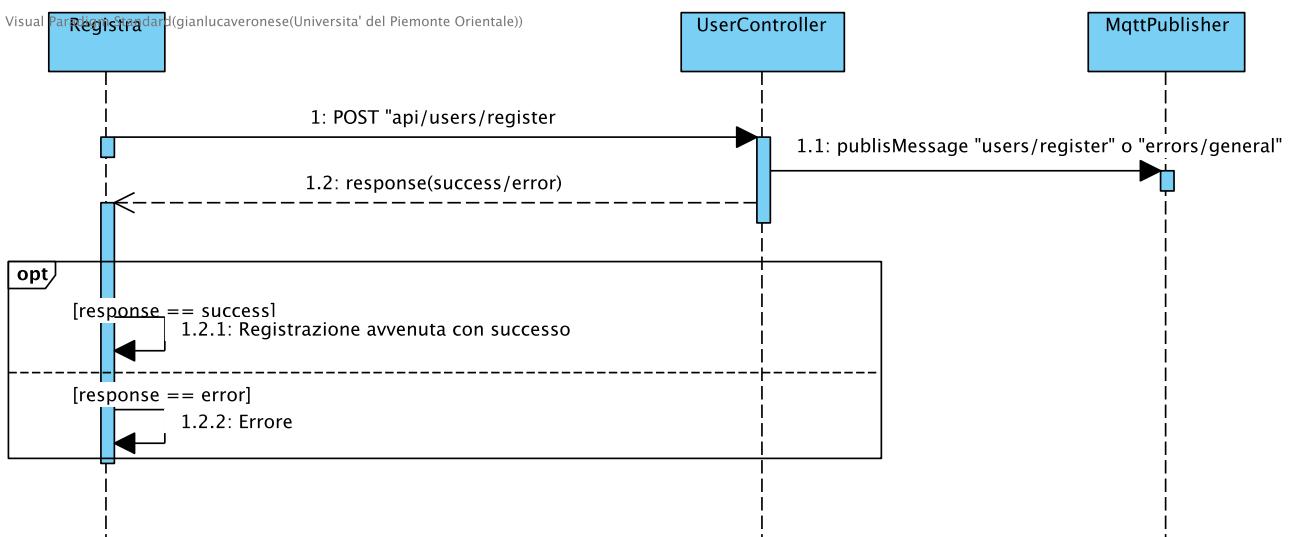
13. OccupazioneSeq



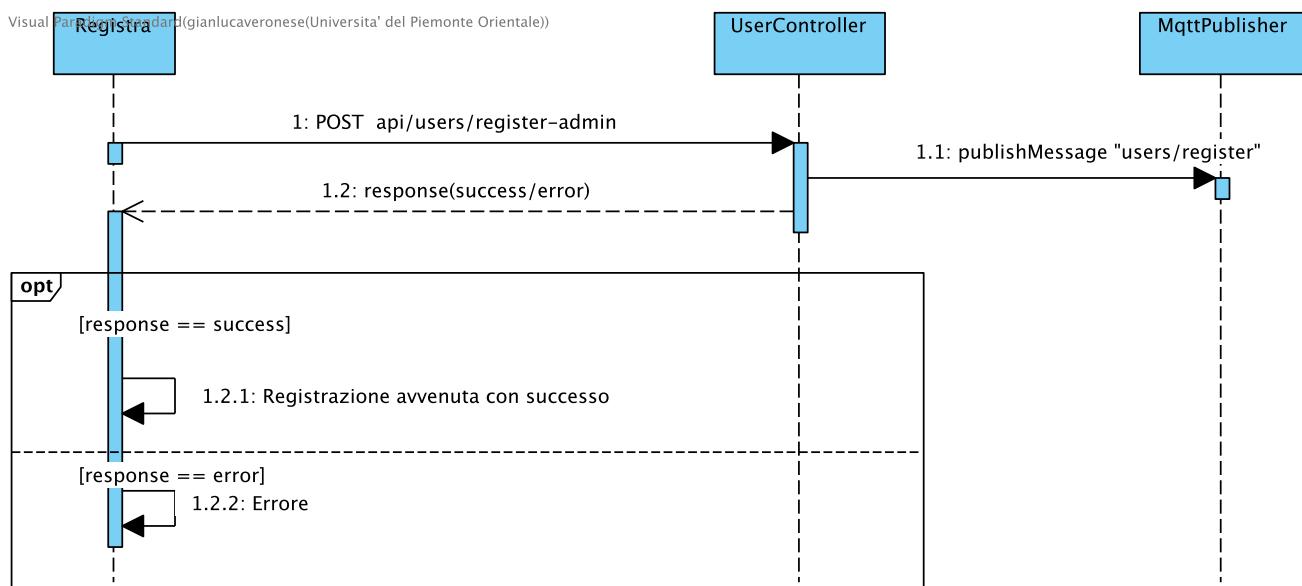
14. PagaRitardoSeq



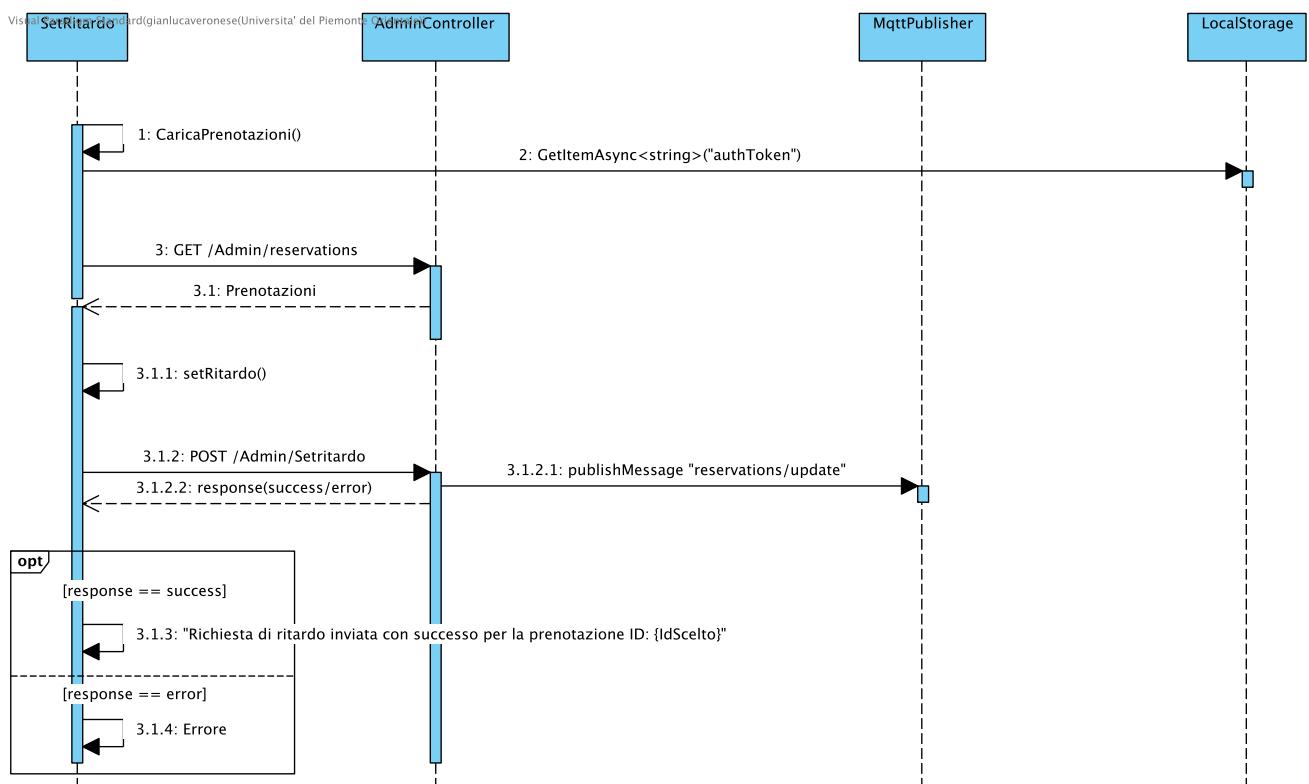
15. RegistraSeq



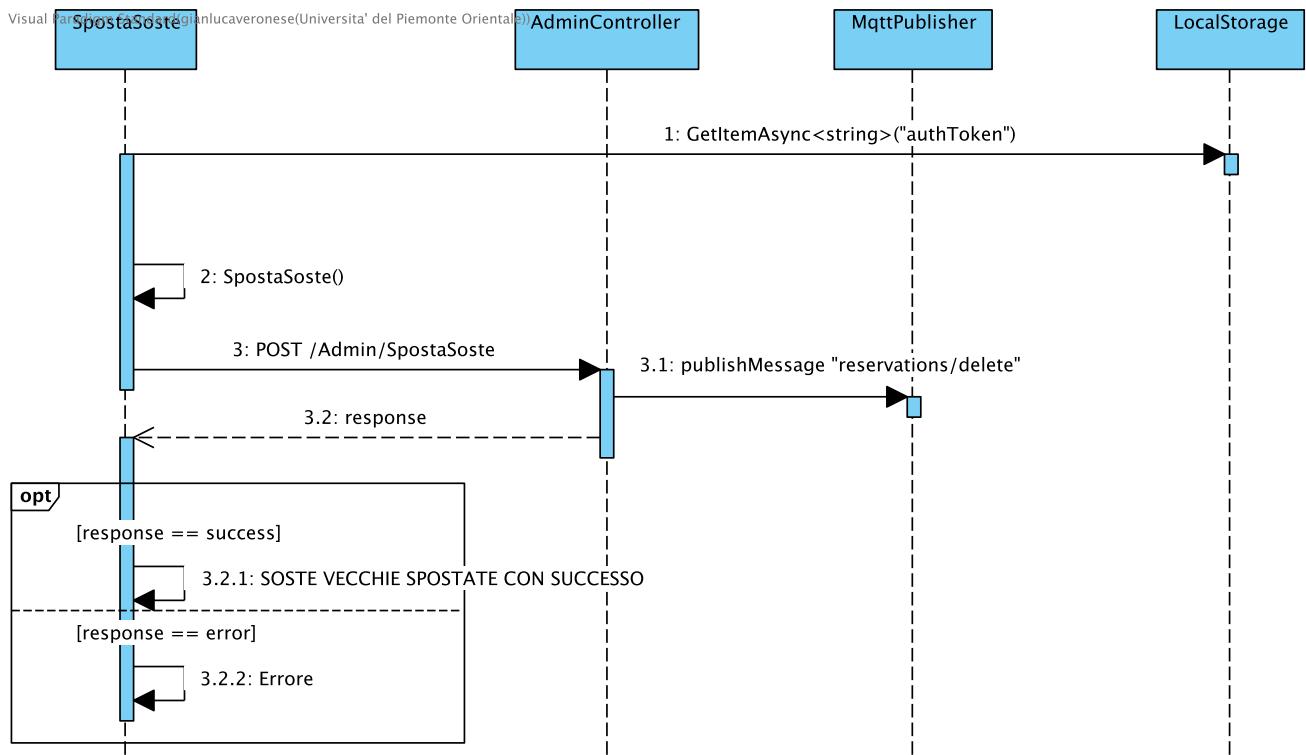
16. RegistraAdminSeq



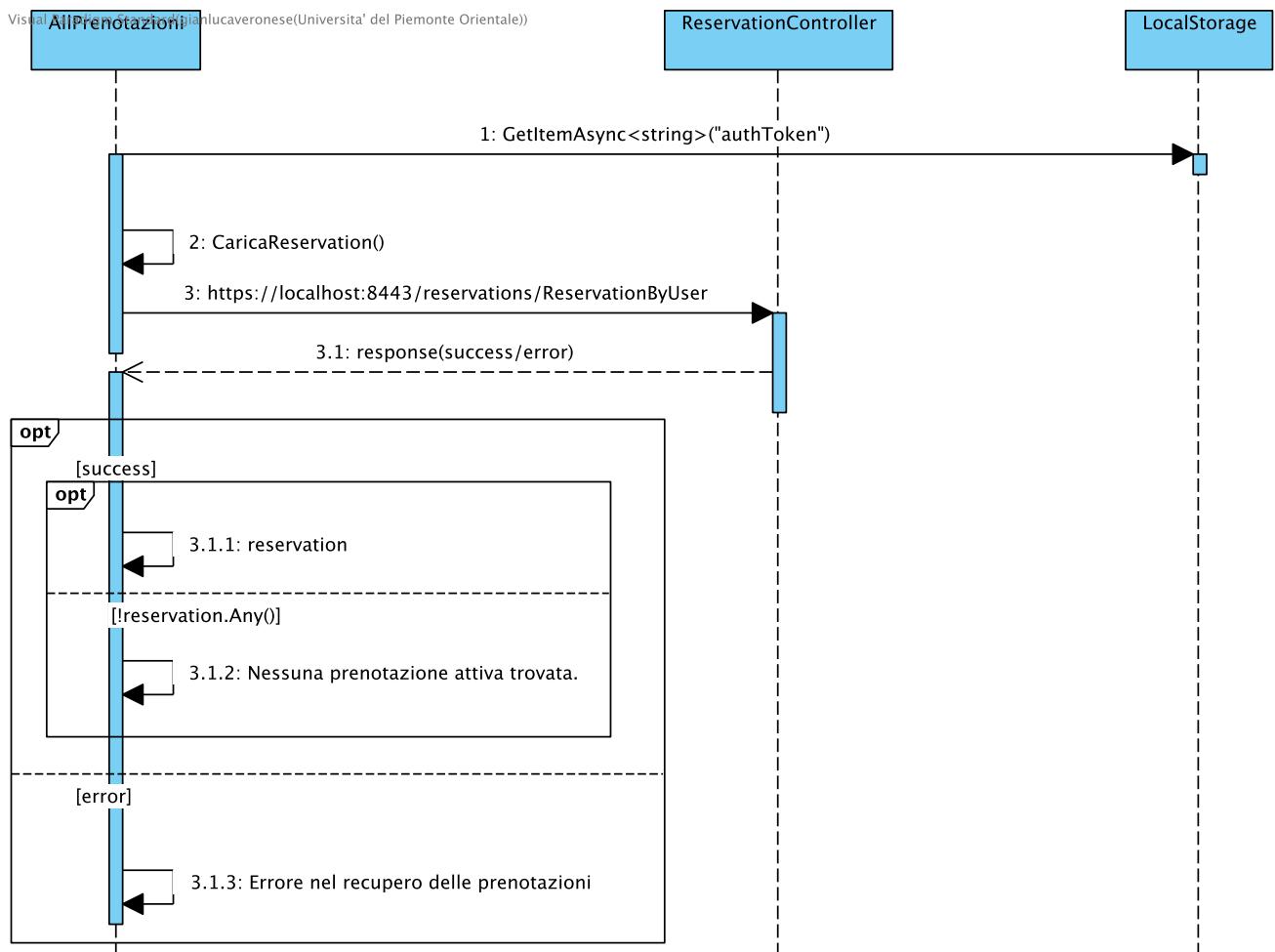
17. SetRitardoSeq



18. SpostaSosteSeq



19. AllPrenotazioniSeq



20. Stampa-PagamentiSeq

