Autori: **Emanuele Marzone** (20049568), **Samuel Titonel** (20052016)

Documentazione Progetto PISSIR

Introduzione

Il seguente progetto ha l'obiettivo di modellare e simulare un sistema di gestione delle reti autostradali attraverso componenti che comunicano tramite un broker sfruttando il protocollo MQTT. Oltre a ciò è stata progettata e sviluppata anche una parte gestionale per amministratori e impiegati per poter visualizzare e modificare elementi del sistema.

Fase di Specifica

Casi d'uso

Utente Veicolo

- Richiede ingresso (manuale o telepass)
- Richiede uscita e paga (manuale)
- Uscita automatica (telepass)

Server Centrale

- Registra ingresso e uscita
- Calcola tratta e pedaggio
- Gestisce guasti o incongruenze

Amministratore/Impiegato

- Consulta caselli, tariffe, passaggi
- Modifica tariffe
- Gestisce pagamenti telepass
- Opzionale: visualizza statistiche e multe

Diagramma dei Casi d’uso

NOTA: I casi d’uso interni al server centrale (es. calcolo pedaggi, gestione anomalie) non sono stati rappresentati nel diagramma, poiché si tratta di logiche di backend non attivabili direttamente dagli attori esterni.

Composizione del Casello Autostradale

Ogni casello autostradale è composto da **quattro dispositivi principali**, ciascuno con una funzione distinta:

- **Dispositivo di ingresso manuale**: consente al veicolo di richiedere un biglietto. Il dispositivo assegna una matricola univoca, registra targa, data, ora e nome del casello, e invia tutte le informazioni al server centrale.

- **Dispositivo di ingresso automatico**: rileva la presenza di un trasmettitore telepass, registra ID, targa e timestamp, e comunica l'ingresso al server.

- **Dispositivo di uscita manuale**: riceve il biglietto, lo legge e chiede al server centrale le informazioni sulla tratta. Visualizza il pedaggio e attende il pagamento prima di aprire la sbarra.

- **Dispositivo di uscita automatico**: riceve la richiesta di uscita dal trasmettitore telepass, raccoglie la targa, invia le informazioni al server e apre la sbarra.

Opzionalmente, ogni casello può includere **una telecamera** per la lettura automatica della targa e **una sbarra** per far entrare e uscire i veicoli . Tutti i dispositivi comunicano tramite **messaggi MQTT** inviati a specifici topic.

Funzionalità del Livello Gestionale

Il sistema gestionale è pensato per **amministratori e impiegati** e consente:

- La **consultazione dei caselli**, delle tratte, delle tariffe e dei passaggi registrati.

- La **modifica delle tariffe** o la **gestione di nuovi caselli** (aggiunta/rimozione).

- La **gestione dei pagamenti telepass**, riscuotere pagamenti telepass in sospeso.

- La **visualizzazione di statistiche e report** (es. tratte più battute, multe, ricavi), se prevista.

Ogni utente gestionale accede al sistema tramite autenticazione, con ruoli differenziati:

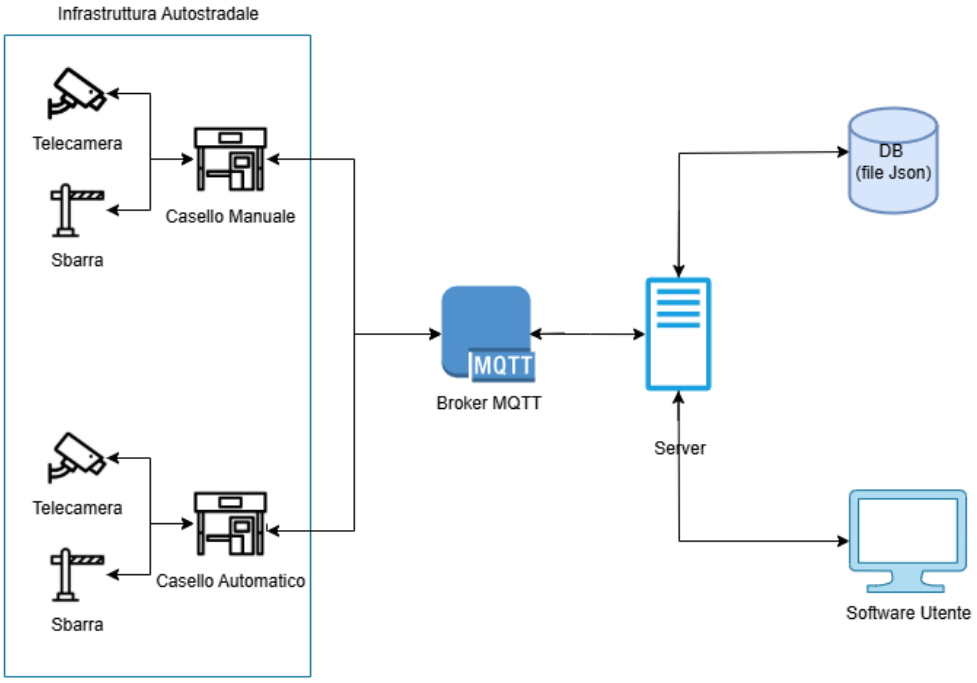
- **Amministratore**: pieno accesso a tutte le funzionalità.

- **Impiegato**: accesso in sola lettura (consultazione e statistiche).

Fase di Progettazione

Nome Task	**Chi ha svolto la Task**
Fase di Specifica	Entrambi
Fase di Progettazione	Entrambi
Creazione Diagrammi	Emanuele
Stesura Documentazione	Entrambi
Creazione topic MQTT	Entrambi
Sviluppo Caselli, Telecamera e Sbarra in Java	Entrambi
Sviluppo Server in Java	Entrambi
Configurazione CA e Mosquitto con TLS	Emanuele
Configurazione KC e Implementazione	Samuel
Creazione API in java	Entrambi
Creazione Frontend Gestionale	Entrambi

Diagramma delle componenti



Le componenti coinvolte sono:

- **Casello**: interagisce col veicolo e comunica tramite MQTT
- **Telecamera**: interagisce col casello e comunica tramite MQTT
- **Sbarra**: interagisce col casello, col server, col veicolo e comunica tramite MQTT
- **Server Centrale**: riceve eventi dai caselli, gestisce la logica, risponde e fornisce API REST.
- **Database**: formato da file Json, permette la persistenza dei dati sul sistema
- **Software Utente**: software fornito a impiegati e amministratori per svolgere operazioni di visualizzazione e modifica di elementi/dati del sistema

NOTA: La sbarra e la telecamera possono essere considerate come un'estensione del casello

Casello

Input: dati provenienti dalla telecamera (targa), biglietto dal veicolo, dati provenienti dal server (dati tratta percorsa dal veicolo)
Output: azioni fisiche (erogazione biglietto, ritiro biglietto, visualizzazione importo da pagare, emissione resto, etc…) e pubblicazione su topic MQTT

Telecamera

Input: richiesta lettura targa dal casello
Output: azioni fisiche (lettura targa) e pubblicazione su topic MQTT

Sbarra

Input: richiesta apertura sbarra dal server
Output: azioni fisiche (apertura e successiva chiusura sbarra) e pubblicazione su topic MQTT

Server Centrale

Input: messaggi su topic MQTT dai dispositivi, richieste di dati tramite API
Output: risposte su topic MQTT, aggiornamento database, dati verso il software utente

Database

Input: richieste di scrittura/lettura dati dal server
Output: risposta alla richiesta del server

Software Utente

Input: interazione utente
Output: risposta visiva all'utente, chiamate alle API

Implementazione

Requisiti tecnici

- **Sistema operativo**: Windows 10 o superiore, Linux
- **Linguaggi**: Java, Javascript, HTML, CSS
- **Componenti da Installare e configurare**: Mosquitto, Keycloak, file CA per comunicazione TLS su mosquitto, file mosquitto.pwd per accesso sicuro ai topic
- **Database**: file Json

File configurazione mosquitto.conf

 # mosquitto.conf - Configurazione base per Windows

 # Ascolta sulla porta standard MQTT (1883) -> 8883 dopo aver usato la CA
listener 8883

 # Permette connessioni anonime (solo per sviluppo/test)
allow_anonymous false

 # Percorso file log (puoi cambiare o commentare)
log_dest file <proprio percorso\log\mosquitto.log>

 # Abilita la persistenza dei messaggi

persistence true
persistence_location <proprio percorso\data>

#Path file delle password
password_file <proprio percorso\mosquitto.pwd>

#ca.cert
cafile <proprio percorso\ca.crt>

#broker.cert
certfile <proprio percorso\server.crt>

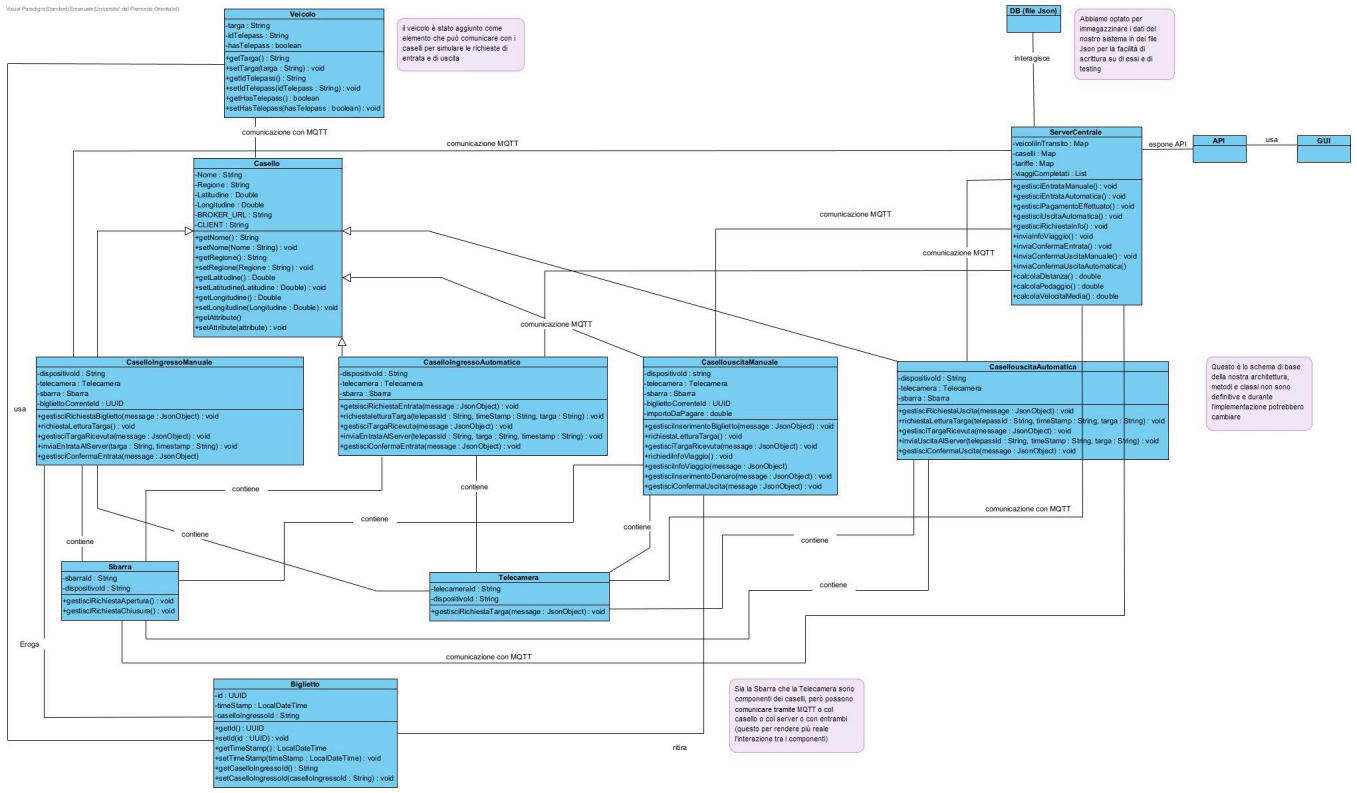
#broker.key
keyfile <proprio percorso\server.key>

require_certificate true

#per autenticare il client:
use_identity_as_username true

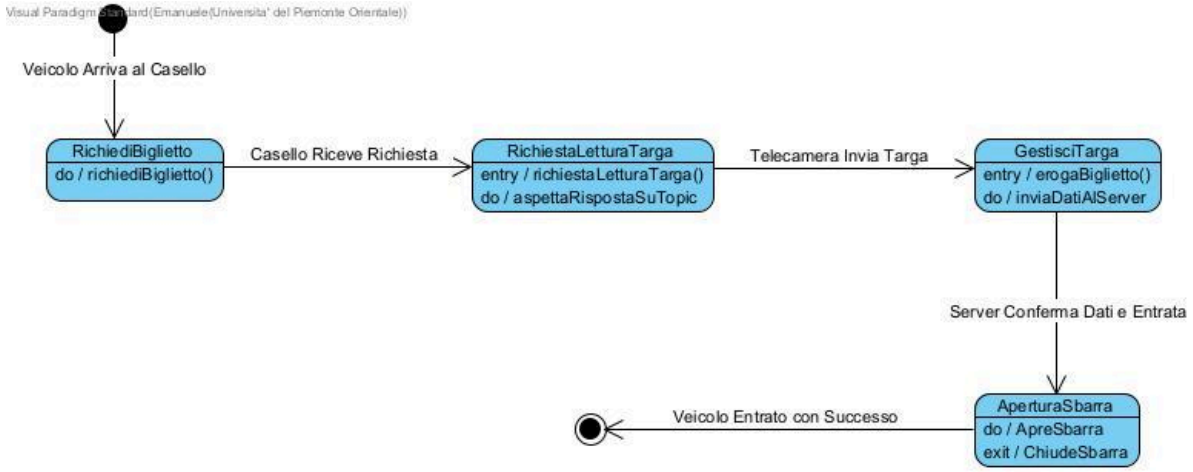
Autostrada

Diagramma di classe

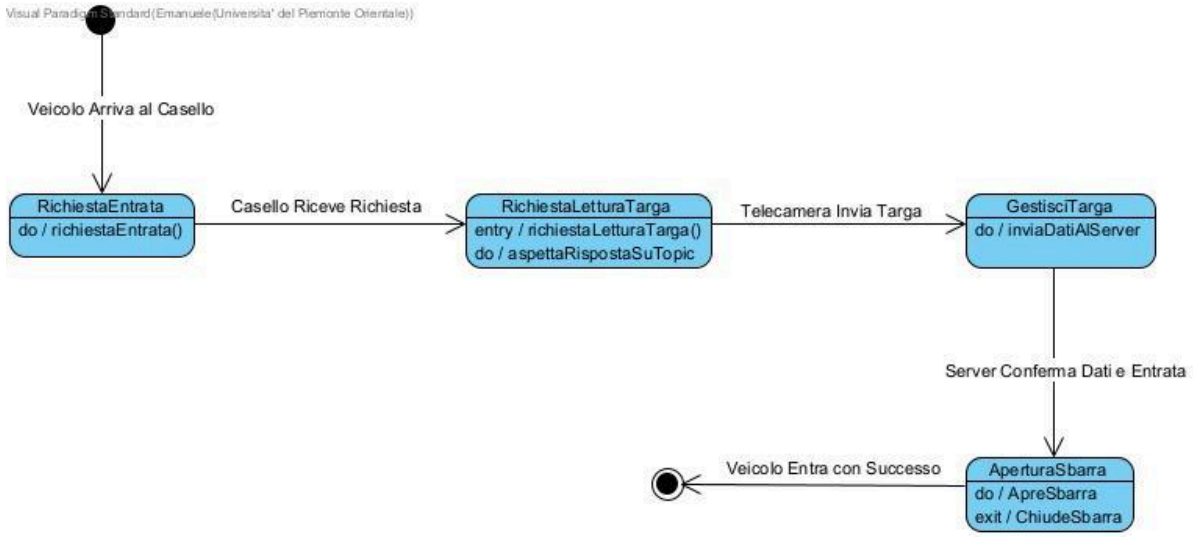


State machine diagrams

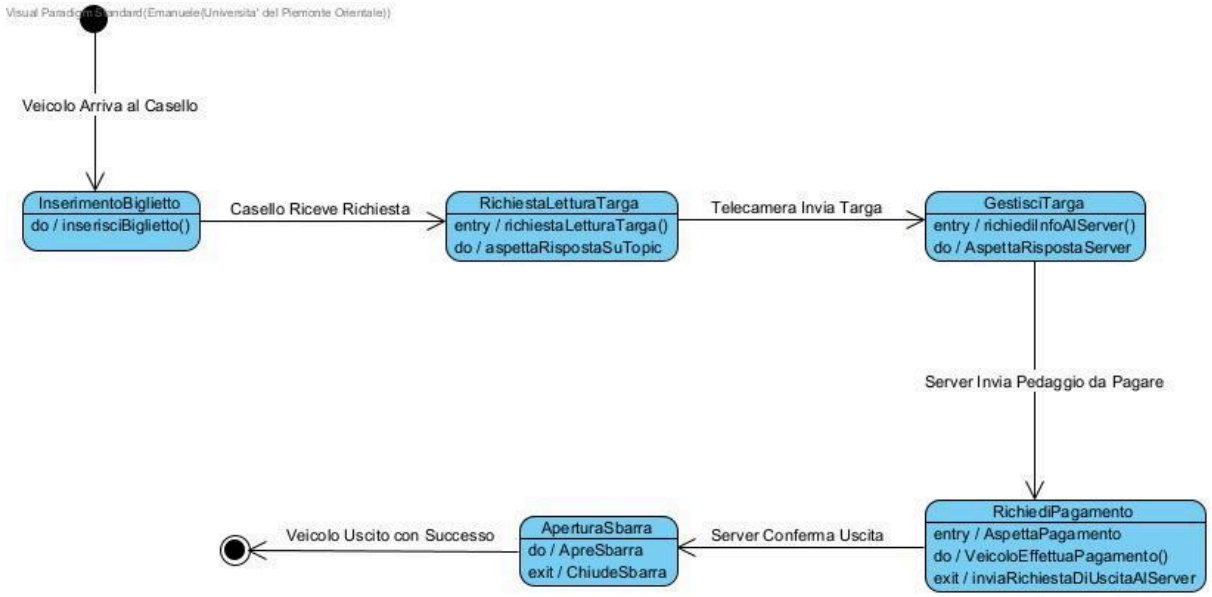
Ingresso veicolo da casello manuale



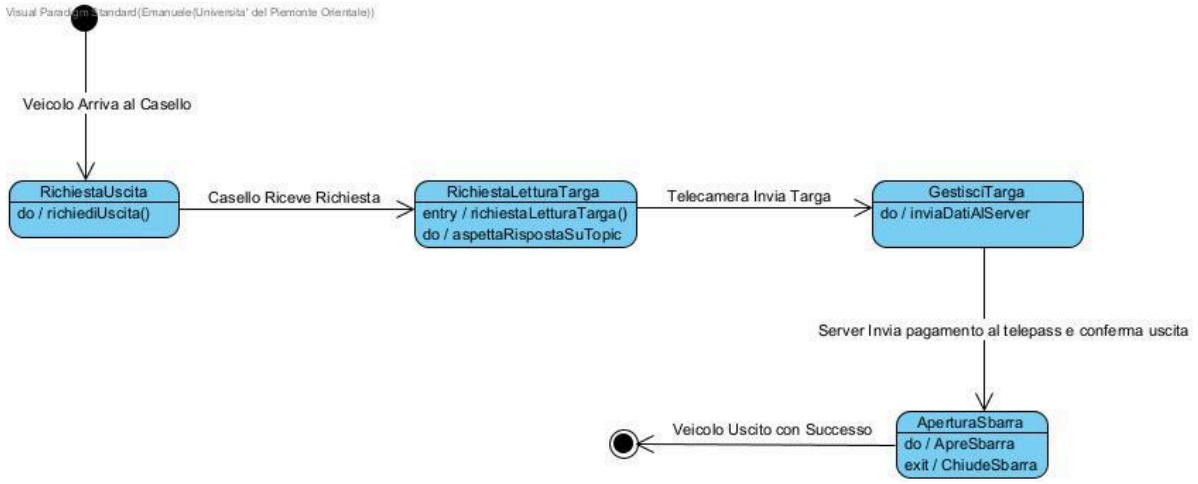
Ingresso veicolo da casello automatico



Uscita veicolo da casello manuale

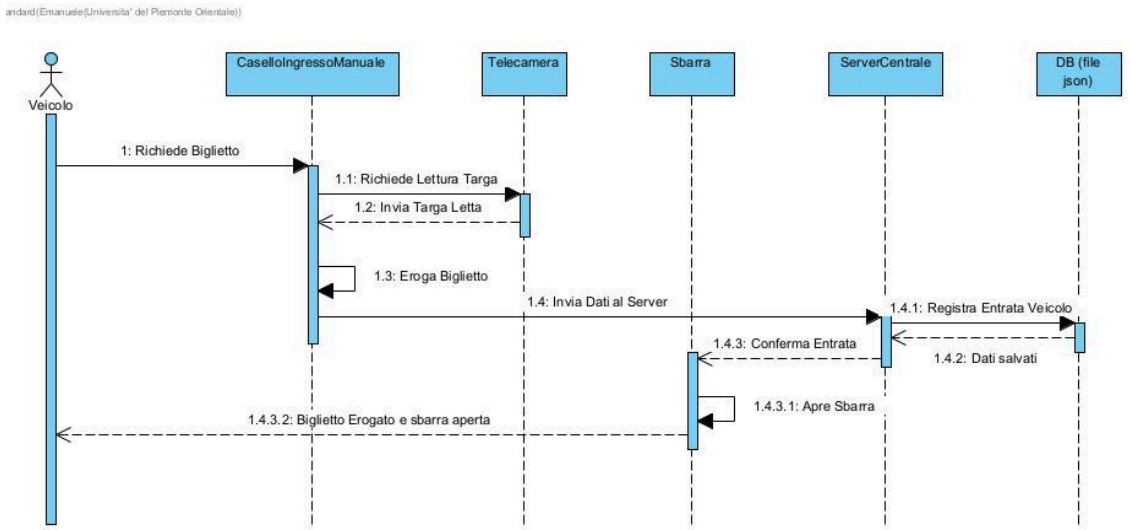


Uscita veicolo da casello automatico

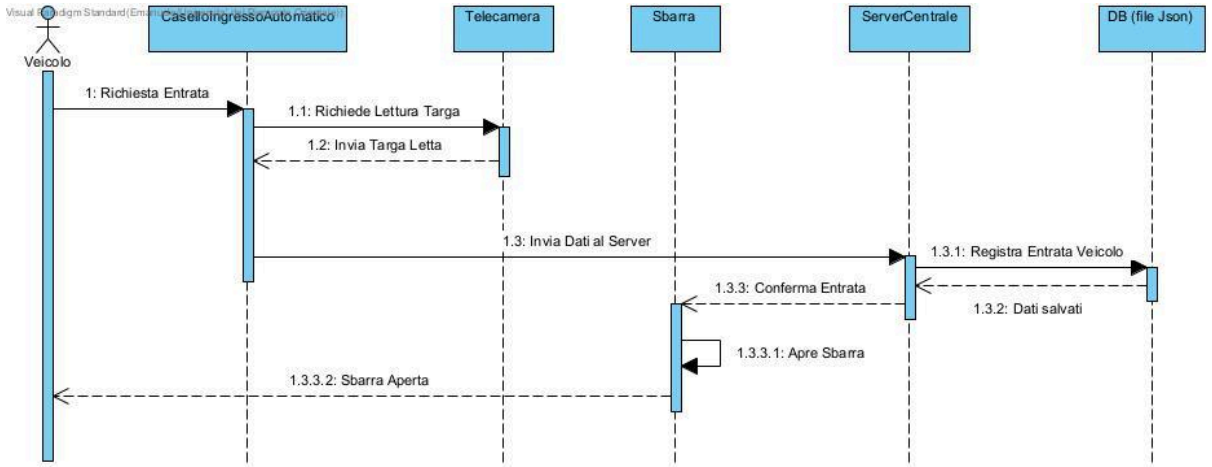


Diagrammi di sequenza

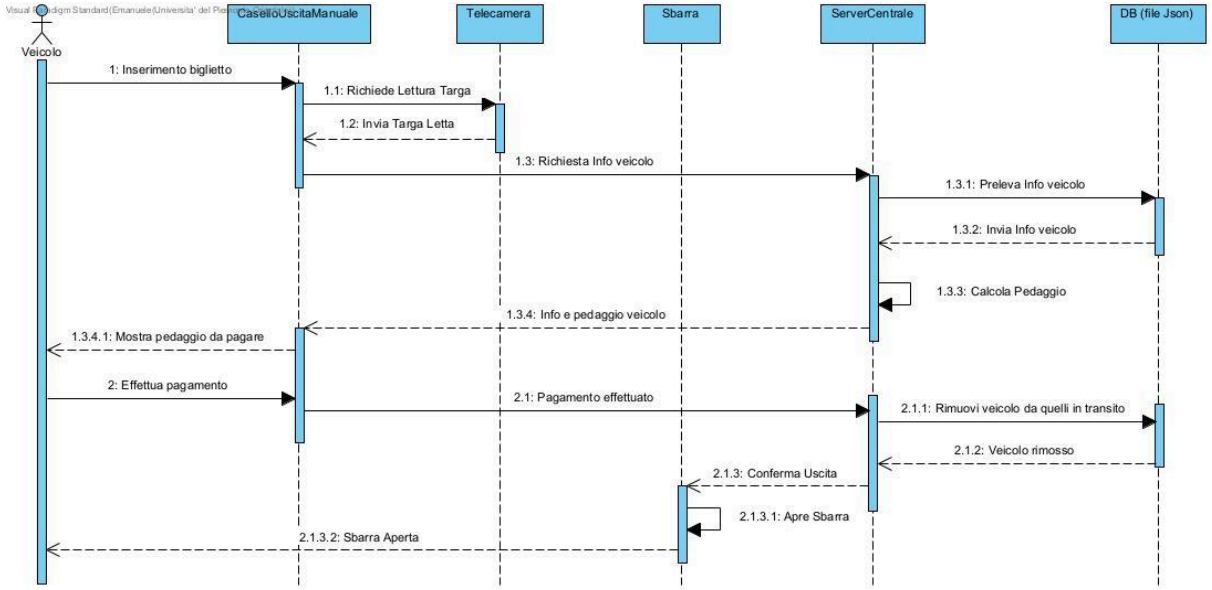
Ingresso veicolo da casello manuale



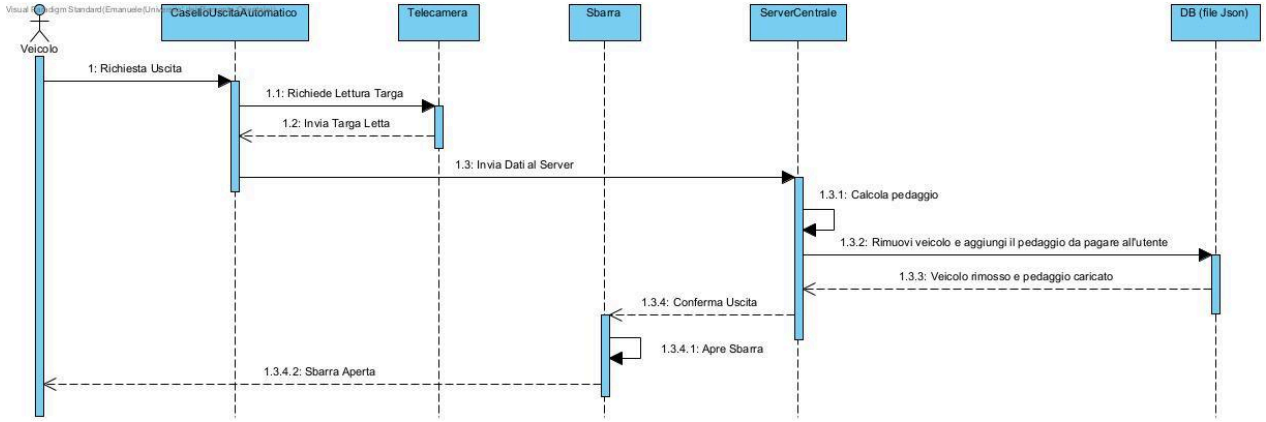
Ingresso veicolo da casello automatico



Uscita veicolo da casello manuale



Uscita veicolo da casello automatico



Utenti mosquitto

- ingressoAuto
- ingressoMan
- uscitaAuto
- uscitaMan
- serverCentrale
- telecamera
- sbarra
- veicolo --> è anche lui un utente mosquitto per poter avere la possibilità di simulare il sistema, nel caso reale verrebbe sostituito da sensori dentro il casello

Il server centrale è unico mentre per gli altri utenti ce ne possono essere più di uno, questo per permettere l'aggiunta modulare di componenti. In un caso reale ci potrebbero essere più

server divisi per zone geografiche per migliorare l'efficienza e l'affidabilità (ridondanza e copertura di territorio più ampia)

Topic Mosquitto

Ogni dispositivo può sottoscriversi e pubblicare messaggi su dei topic, qui di seguito vengono elencati nella loro interezza, suddivisi per componenti che inviano messaggi su quel topic.
Abbiamo optato per una nomea dei topic abbastanza esplicativa così da rendere più semplice e intuitivo il loro utilizzo, sono formati da 3 parti principali

1. Chi compie l'azione.
2. Parametri di supporto (ad esempio: ID dispositivo, tipologia del dispositivo che compie l'azione, etc).
3. L'azione effettiva.

NOTA: la "|" all'interno dei topic serve per indicare un "oppure" (A|B) quindi il topic può esistere sia con A che con B in quel punto. Questa pratica è stata inserita per alleggerire la visibilità dei vari topic nella documentazione.

Casello (ingresso)

- casello/{id}/ingresso/manuale/biglietto_erogato
- casello/{id}/ingresso/automatico/invia_dati_server

Casello (uscita)

- casello/{id}/uscita/manuale/richiesta_info
- casello/{id}/uscita/manuale/pagamento_effettuato
- casello/{id}/uscita/automatico/invia_dati_server

Sbarra

- casello/{id}/sbarra/ingresso|uscita/sbarra_aperta

Telecamera

- casello/{id}/telecamera/ingresso|uscita/manuale|automatico/richiesta_targa
- casello/{id}/telecamera/ingresso|uscita/manuale|automatico/lettura_targa

Veicolo

- veicolo/{idCasello}/ingresso/automatico/richiesta_entrata
- veicolo/{idCasello}/ingresso/manuale/richiesta_biglietto
- veicolo/{idCasello}/uscita/automatico/richiesta_uscita
- veicolo/{idCasello}/uscita/manuale/inserimento_biglietto
- veicolo/{idCasello}/uscita/manuale/inserimento_denaro

Server

- server/casello/{id}/uscita/manuale/info_viaggio
- server/casello/{id}/conferma_entrata
- server/casello/{id}/conferma_uscita

Protocolli di comunicazione dispositivi

Come già visto visivamente tramite i diagrammi sopra abbiamo definito dei passaggi precisi per la comunicazione tra dispositivi sfruttando i topic MQTT, di seguito vengono riportate anche le versioni testuali che esplicitano l'uso dei topic nelle varie fasi

- Entrata Manuale
    - veicolo/{idCasello}/ingresso/manuale/richiesta_biglietto (veicolo->casello)
    - casello/{id}/telecamera/ingresso/manuale/richiesta_targa (casello->telecamera)
    - casello/{id}/telecamera/ingresso/manuale/lettura_targa (telecamera->casello)
    - casello/{id}/ingresso/manuale/biglietto_erogato (casello->server)
    - server/casello/{id}/conferma_entrata (server->sbarra)
    - casello/{id}/sbarra/ingresso/sbarra_aperta (sbarra->veicolo)

- Entrata Automatica
    - veicolo/{idCasello}/ingresso/automatico/richiesta_entrata (veicolo->casello)
    - casello/{id}/telecamera/ingresso/automatico/richiesta_targa (casello->telecamera)
    - casello/{id}/telecamera/ingresso/automatico/lettura_targa (telecamera->casello) (ottiene anche il telepass)
    - casello/{id}/ingresso/automatico/invia_dati (casello-> server)
    - server/casello/{id}/conferma_entrata (server->sbarra)
    - casello/{id}/sbarra/ingresso/sbarra_aperta (sbarra->veicolo)

- Uscita Manuale
    - veicolo/{idCasello}/uscita/manuale/inserimento_biglietto (veicolo->casello)
    - casello/{id}/telecamera/uscita/manuale/richiesta_targa (casello->telecamera)
    - casello/{id}/telecamera/uscita/manuale/lettura_targa (telecamera->casello)
    - casello/{id}/uscita/manuale/richiesta_info (casello->server) (prende il biglietto, la targa letta e chiede le informazioni al server)
    - server/casello/{id}/uscita/manuale/info_viaggio (server->casello, veicolo) (controlla che i dati siano corretti e mostra la tariffa da pagare)
    - veicolo/{idCasello}/uscita/manuale/inserimento_denaro (veicolo->casello)
    - casello/{id}/uscita/manuale/pagamento_effettuato (casello->server, veicolo) (veicolo per restituire il resto)

    - server/casello/{id}/conferma_uscita (server->sbarra)
    - casello/{id}/sbarra/uscita/sbarra_aperta (sbarra->veicolo)

- Uscita Automatica
    - veicolo/{idCasello}/uscita/automatico/richiesta_uscita (veicolo->casello)
    - casello/{id}/telecamera/uscita/manuale/richiesta_targa (casello->telecamera)
    - casello/{id}/telecamera/uscita/automatico/lettura_targa (telecamera->casello)
    - casello/{id}/uscita/automatico/invia_dati (casello-> server)
    - server/casello/{id}/conferma_uscita (server->sbarra)
    - casello/{id}/sbarra/uscita/sbarra_aperta (sbarra->veicolo)

Imprevisti

Abbiamo implementato durante il pagamento del pedaggio, da parte del veicolo, la possibilità di pagare di più o di meno di quello richiesto e questo si tramuta in una nuova richiesta di pagamento se è insufficiente o un'attesa del resto se superiore.

Abbiamo anche implementato le multe che vengono caricate all'interno dello storico dei viaggi (funzionalità creata sfruttando valori randomici, perchè con la logica effettiva dei timestamp non vengono generate multe, timestamp troppo vicini per dati reali)

Software Utente (Gestionale)

La parte del software lato utente (gestionale), è stata creata sfruttando diverse tecnologie quali, Javalin per creare le API fornite dal backed e le rotte che renderizzano le pagine nel frontend, Keycloak per l'autenticazione dei client (definizione dei ruoli, gruppi di utenti, configurazione reame), HTML, CSS e Javascript per la visualizzazione dinamica delle pagine e l'interazione con esse.

Architettura

- **Presentazione**: pagine HTML con Javascript e CSS
- **Business Logic**: codice Java con Javalin
- **Comunicazione**: tramite rotte e chiamate REST
- **Autenticazione**: basata su Keycloak

Sono presenti due tipi di ruoli per un utente, l'amministratore e l'impiegato, un impiegato può svolgere solo operazioni di visualizzazione di alcuni elementi del sistema, mentre l'amministratore può visualizzare tutti gli elementi ed apportare modifiche (aggiunta, aggiornamento e rimozione).

API REST

(Potrebbero subire modifiche durante l'implementazione)

Il sistema espone un insieme di API REST per l'interazione con le risorse gestite (caselli, tratte, guadagni, telepass, multe). Ogni endpoint è accessibile in base al ruolo dell'utente autenticato: **amministratore** o **impiegato**.

Endpoints relativi ai caselli

- **`GET /caselli`**
  Restituisce l'elenco completo dei caselli, comprensivo di nome, regione e coordinate geografiche.
  Accesso: impiegato, amministratore

- **`POST /caselli`**
  Consente di aggiungere un nuovo casello al sistema, specificando i dati necessari.
  Accesso: solo amministratore

- **`PUT /caselli/{nomeCasello}`**
  Permette di modificare le informazioni di un casello esistente, individuato tramite il parametro **`nomeCasello`**.
  Accesso: solo amministratore

- **`DELETE /caselli/{nomeCasello}`**
  Rimuove un casello dal sistema.
  Accesso: solo amministratore

Endpoints relativi alle tratte

- **`GET /tratte`**
  Restituisce tutte le tratte presenti nel sistema, con le relative informazioni: casello di ingresso, casello di uscita, distanza, pedaggio e numero di utilizzi.
  Accesso: impiegato, amministratore

- **`PUT /tratte/{trattaId}`**
  Consente di modificare il **pedaggio** associato a una tratta specifica.
  Accesso: solo amministratore

Endpoints relativi ai guadagni

- **`GET /guadagni`**
  Restituisce il guadagno complessivo ottenuto da tutte le tratte percorse.
  Accesso: solo amministratore

- **`GET /guadagni/{nomeCaselloIngresso}/{nomeCaselloUscita}`**
  Restituisce il guadagno generato dalla tratta specificata tra due caselli.
  Accesso: solo amministratore

Endpoints relativi ai telepass

- **`GET /telepass`**
  Mostra l'elenco dei pagamenti effettuati con Telepass (riscossi e non).
  Accesso: solo amministratore

- **`PUT /telepass/{telepassId}`**
  Riscuote il pagamento relativo ad un viaggio svolto col telepass.
  Accesso: solo amministratore

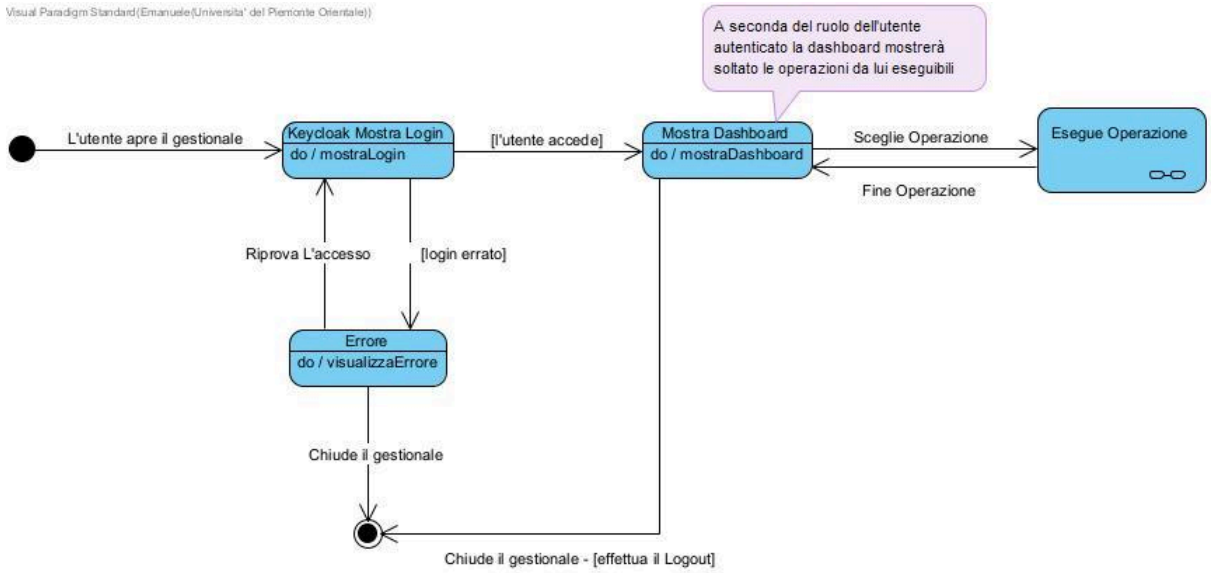
- **`PUT /telepass`**
  Riscuote tutti i pagamenti svolti col telepass.
  Accesso: solo amministratore

Endpoints relativi alle multe

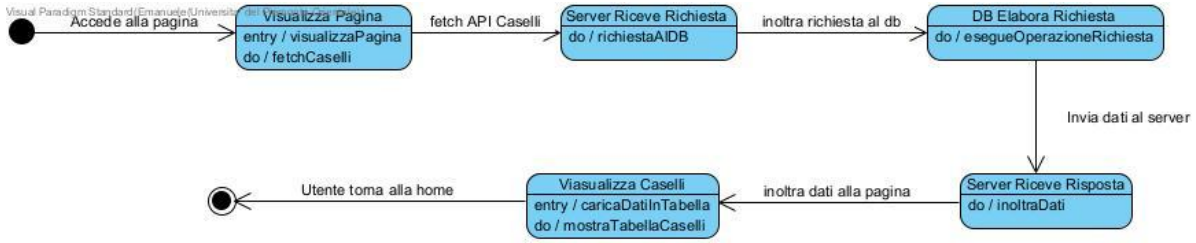
- **`GET /multe`**
  Restituisce l'elenco di tutte le multe registrate nel sistema.
  Accesso: impiegato, amministratore

State Machine Diagrams

Accesso Utente

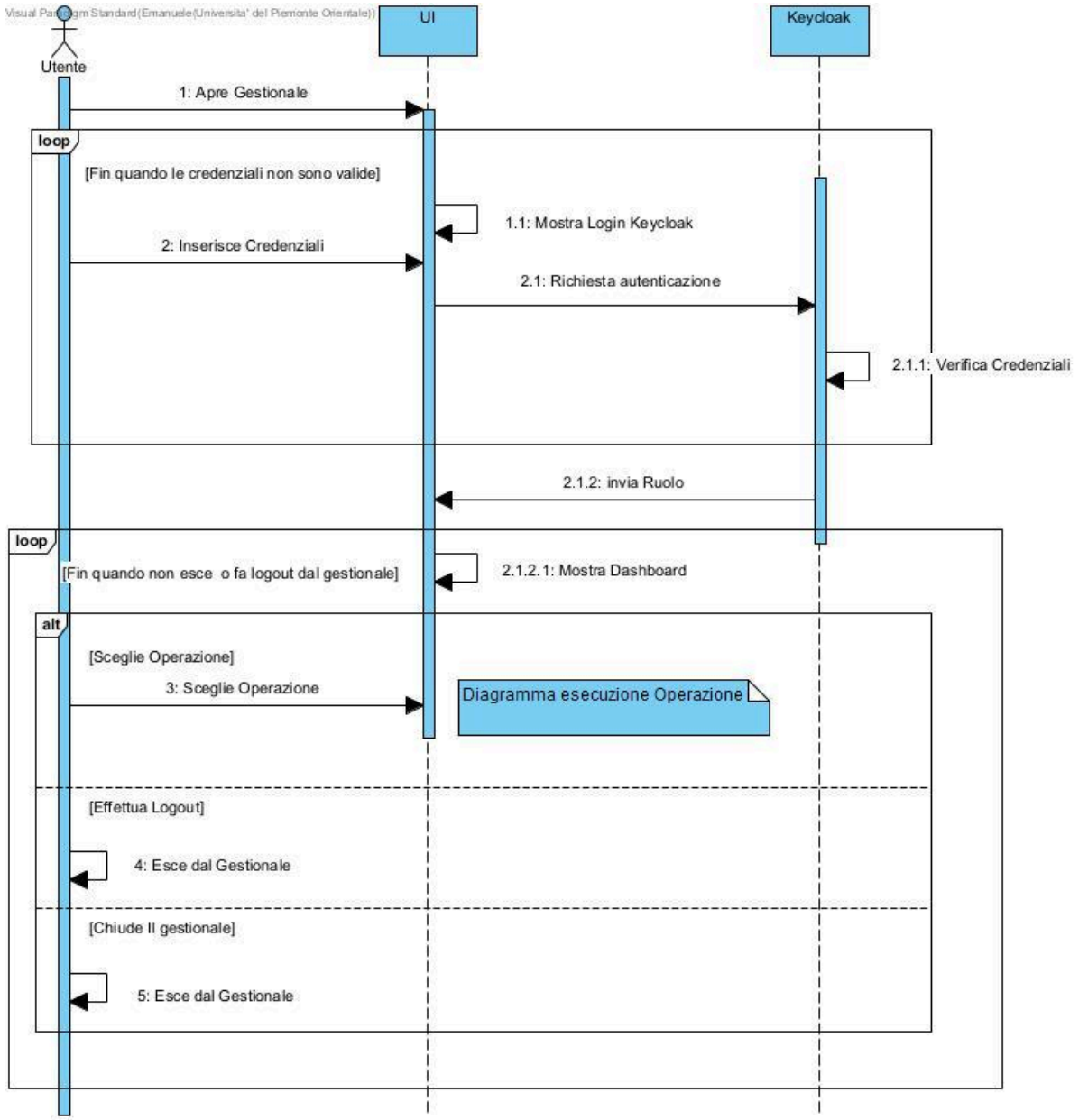


Visualizza Caselli

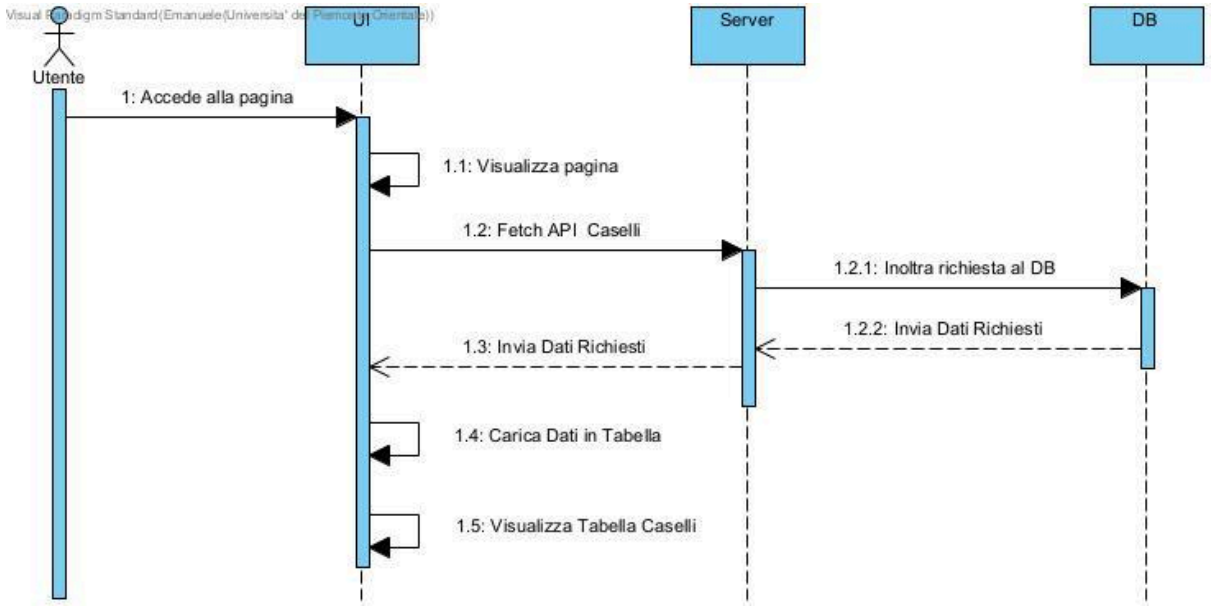


Diagrammi di sequenza

Accesso Utente



Visualizza Caselli



NOTA: Abbiamo optato per la creazione di solo alcuni dei diagrammi delle operazioni dell'utente perché questi ultimi erano tutti molto simili tra loro (operazione utente - fetch API al server - completamento operazione).
Abbiamo aggiunto anche l'accesso dell'utente perché al contrario delle operazioni coinvolge un componente diverso (keycloak) non presente in altri diagrammi.

Demo Autostrada

Per testare correttamente i microservizi del sistema autostradale, è necessario installare e configurare **Mosquitto**, il broker MQTT utilizzato per la comunicazione tra i componenti.
La configurazione deve includere:

- i **certificati della Certificate Authority (CA)**, per abilitare la comunicazione sicura tramite **TLS**;

- le **credenziali di autenticazione** dei microservizi, necessarie per la pubblicazione e la sottoscrizione ai vari **topic MQTT**.

Una volta completata la configurazione, è possibile:

1. Avviare il **server centrale**, responsabile della gestione dei messaggi e del coordinamento tra i microservizi;

2. Avviare la classe **Main** per osservare, tramite terminale, i **log delle operazioni**.

Tra le operazioni registrate nei log vi sono:

- la sottoscrizione ai topic MQTT;
- l'invio e la ricezione dei messaggi;
- il salvataggio delle informazioni su file **JSON** (simulando un database);
- la gestione dei veicoli in transito, inclusa la **rimozione al termine del viaggio**;
- Il salvataggio del viaggio e dell'eventuale multa;
- l'apertura e chiusura delle sbarre ai caselli;
- l'elaborazione e il pagamento dei **pedaggi**.

All'avvio del main verranno anche svolte le operazioni di inizializzazione: i caselli in uso, le tratte e verranno fatti partire dei veicoli; sia quelli che useranno i caselli manuali, sia quelli che useranno i caselli automatici.

Demo Gestionale

Per utilizzare correttamente il sistema, l'utente deve innanzitutto avviare e configurare **Keycloak**, che gestisce l'autenticazione degli utenti.

Dopo aver scaricato la versione 12.0.4 di Keycloak, è necessario eseguirlo utilizzando lo script *standalone.bat* su Windows oppure *standalone.sh* su Linux o macOS. Una volta avviato, Keycloak sarà accessibile all'indirizzo **http://localhost:8080**.

Nel pannello di amministrazione di Keycloak, l'utente deve:

- Creare un nuovo **realm**, chiamato AutostradaPISSIR;
- All'interno di questo realm configurare un **client** con ID autostradaPissir;
- Come **Root URL** si dovrà impostare http://localhost:8081, cioè l'indirizzo su cui sarà in esecuzione il server centrale;
- Due **ruoli** *amministratore* e *impiegato*;
- **Utenti** con i diversi ruoli, per visualizzare le diverse UI.

Nella nostra configurazione abbiamo creato due utenti le cui credenziali sono:

1. **Nome Utente:** amministratore

   Password: amministratore

2. **Nome Utente**: impiegato

   Password: impiegato

Terminata la configurazione di Keycloak e lasciato in ascolto, l'utente può procedere con **l'avvio** del Server Centrale, eseguendo il file **ServerCentrale**. Questo server sarà in ascolto all'indirizzo **http://localhost:8081**.

Una volta che il server è in esecuzione, l'utente può accedere tramite browser a http://localhost:8081. Verrà automaticamente reindirizzato alla schermata di login di Keycloak.

Inserendo le credenziali di uno degli utenti precedentemente creati, l'utente verrà autenticato e verrà reindirizzato alla pagina principale del gestionale.

In questa schermata **compariranno** i **pulsanti** e le **funzionalità specifiche**, **in base al ruolo** dell'utente autenticato: l'amministratore avrà pieno accesso a tutte le funzionalità, mentre l'impiegato potrà visualizzare e interagire solo con alcune operazioni previste per lui.

Demo Test DAO

Per garantire il corretto funzionamento delle componenti del sistema autostradale, sono stati implementati test automatici con **JUnit5**.
Questi test verificano il comportamento delle classi DAO responsabili della gestione dei dati salvati in formato JSON.
In particolare:

- **CaselliJsonDAOTest**
  Verifica che i caselli siano correttamente caricati dal file JSON e che sia possibile recuperarli tramite nome oppure selezionandone uno casuale.
  I test non modificano il file caselli.json, garantendo così l'integrità dei dati originali.
- **PagamentiTelepassJsonDAOTest**
  Controlla l'aggiunta di un nuovo pagamento Telepass, la sua corretta lettura e la possibilità di segnarlo come pagato.
  Per evitare interferenze con i dati reali, i test lavorano su un file di test temporaneo che viene eliminato automaticamente al termine dell'esecuzione.
- **MulteJsonDAOTest**
  Simula la creazione di nuove multe, il loro salvataggio e il recupero da file JSON.
  Anche in questo caso viene utilizzato un file dedicato ai test, che viene ripulito dopo ogni esecuzione, evitando modifiche al file reale.

- **TratteJsonDAOTest**
  Verifica le operazioni di aggiunta e rimozione di tratte, assicurandosi che il salvataggio e la cancellazione funzionino correttamente.
  Include inoltre test sul calcolo della distanza tra due caselli e sul calcolo del pedaggio in base ai chilometri percorsi.
- **VeicoliJsonDAOTest**
  Controlla la corretta gestione dei veicoli registrati e di quelli in transito.
  I test includono il salvataggio e il recupero dei veicoli, la rimozione da file e la verifica della disponibilità di una targa manuale per simulare l'ingresso manuale in autostrada.
- **ViaggiJsonDAOTest**
  Verifica la creazione, il salvataggio e la ricerca dei viaggi effettuati.
  I test garantiscono che sia possibile aggiungere un nuovo viaggio, recuperarlo tramite ID e leggere correttamente la lista dei viaggi memorizzati, senza compromettere i dati reali.