Progetto OBDSight

Membri del team di sviluppo:

Memma Luca 0000792725

Palamara Daniel 0000789227

Raimondi Matteo 0000792427

Abstract

Lo scopo del progetto è sviluppare un applicativo che permetta di comunicare con le centraline di diverse autovetture mediante un'apposita interfaccia, utilizzando i comandi dello standard OBD-II, compatibile con la maggior parte dei veicoli diffusi a livello globale.

Con gli applicativi attualmente disponibili si è vincolati ad effettuare la diagnostica restando all'interno dell'abitacolo, ciò richiede costantemente la presenza fisica di un esperto.

Si vuole offrire pertanto la possibilità di effettuare la diagnostica sia in locale, restando all'interno del veicolo, che in remoto, tramite un secondo dispositivo connesso a quello in locale.

Sono previsti sistemi di autenticazione e di permanenza dei dati, che garantiscono il diritto di accedere al veicolo e la possibilità di consultare uno storico dei risultati prodotti in un secondo momento.

L'interfaccia grafica, semplice ed intuitiva, è comune a tutti i dispositivi:

- Può essere utilizzata sia in locale che in remoto
- Offre la possibilità di inviare comandi e mostrare le relative risposte da parte della centralina
- Permette di importare ed esportare i risultati prodotti

Analisi dei requisiti

Raccolta dei requisiti

- L'applicativo deve permettere di inviare dei comandi standard di tipo OBD-II all'interfaccia mediante una GUI semplice ed intuitiva, visualizzando poi in un campo di testo la risposta ottenuta.
- Il sistema all'avvio richiede l'autenticazione dell'utente, necessaria sia per operare in locale che in remoto.
- Il sistema fornisce dei metodi per salvare i dati ottenuti dalle analisi su un file di testo, inglobando anche informazioni come data e ora, il file di testo può essere consultato mediante applicativi esterni o il programma stesso.
- L'utente, ogni qualvolta che l'interfaccia viene collegata al veicolo e al PC, deve effettuare la connessione alla porta COM relativa, oppure avviare una connessione remota.
- Il programma permette di importare dati relativi ad un'analisi; i quali possono essere filtrati per creatore del file oppure data e ora.
- Il sistema permette di connettersi in remoto ad un'altra unità con lo stesso software in esecuzione, permettendo una gestione a distanza, pertanto ogni dispositivo può agire sia come cliente che come servitore; per la gestione remota è previsto un sistema di autenticazione e un sistema di crittografia che garantisce la protezione dei dati.

Analisi Del Dominio

Vocabolario

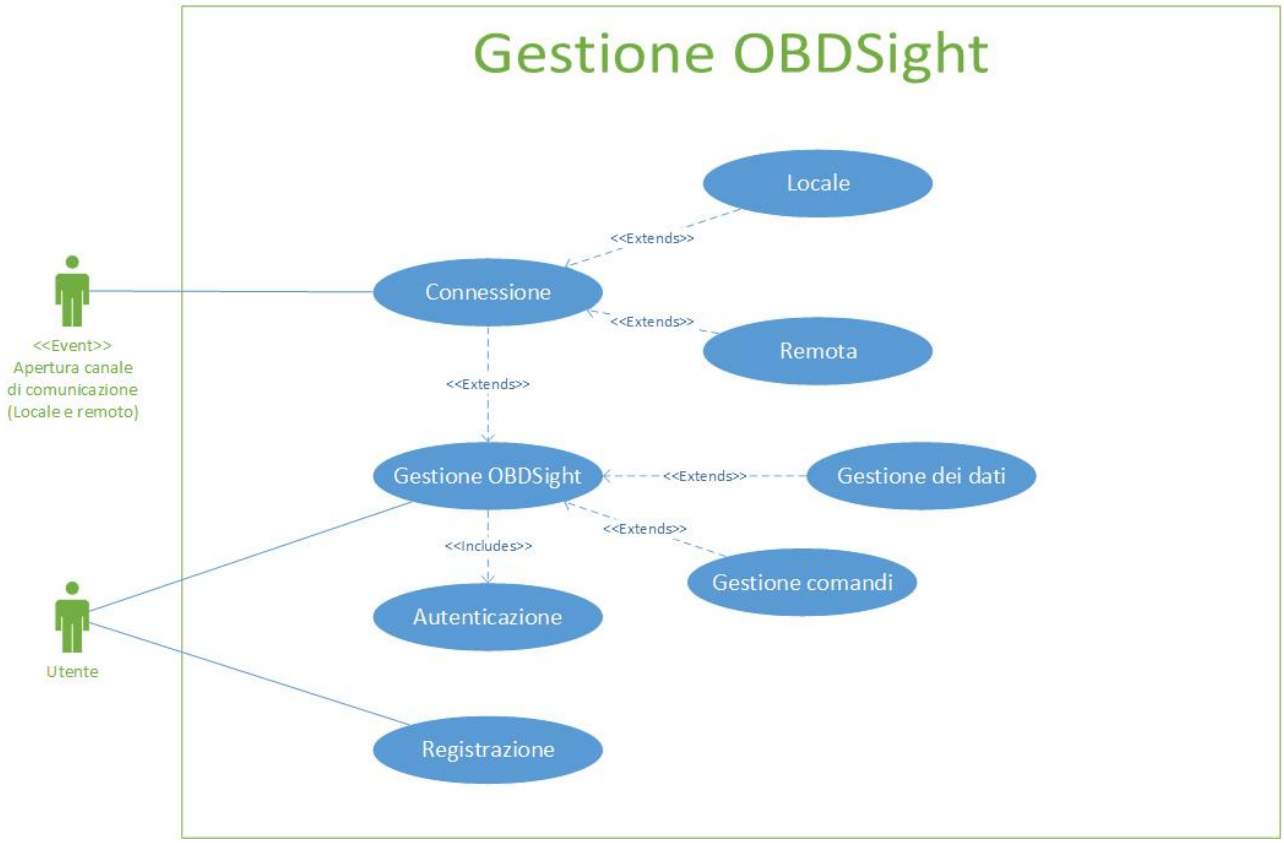
Voce	Definizione	Sinonimi
OBD-II	Standard che permette di collegarsi al sistema di diagnostica degli autoveicoli	
Interfaccia	Mezzo fisico che collega il computer al veicolo	Cavo OBD, Cavo
Dispositivo	Calcolatore sul quale è in esecuzione il software	Computer, PC
Unità remota	Computer connesso all'autoveicolo, gestito a distanza	
Applicativo	Software in esecuzione sul computer	Programma
Risposta	Risultato prodotto dall'auto dopo aver ricevuto un comando appartenente allo standard OBD-II	Dati ottenuti, risultato
Analisi	Invio di uno o più comandi al mezzo	Diagnosi, Diagnostica
Veicolo	Mezzo di trasporto	Automobile, Auto, Mezzo, Autoveicolo
Storico	Insieme delle risposte ottenute dalle analisi	Cronologia
Client	Utilizzatore dell'applicativo connesso da remoto	Utente
Server	Utilizzatore dell'applicativo connesso fisicamente all'interfaccia OBD-II	
Privilegi	Insieme di azioni che l'utente può compiere	Autorizzazioni
Credenziali	Coppia di Username e Password necessari per l'autenticazione nel sistema	
Username	Stringa alfanumerica	
Password	Stringa alfanumerica	
Informazioni chiave	Dati (quali data e ora, identificativo veicolo, …) che associano una risposta ad un determinato veicolo	Identificatori

Sistemi Esterni

Il software interagisce con un sistema esterno: l'interfaccia OBD-II standard.
Essa viene collegata all'autoveicolo e ad un PC rispettivamente attraverso un'interfaccia seriale OBD-II e una porta USB, e gestita da driver proprietari.

Analisi dei Requisiti

Casi d'uso



Scenari

Titolo	Gestione OBDSight
Descrizione	Gestione dell'interfaccia OBD-II e delle sue funzionalità
Attori	Utente
Relazioni	Connessione, Autenticazione, Gestione comandi, Gestione dei dati
Precondizioni	L'utente deve effettuare l'autenticazione
Postcondizioni	
Scenario principale	● L'utente deve autenticarsi con il proprio account per accedere alla funzioni di suo interesse: ○ Connessione locale all'interfaccia OBD-II collegata alla porta COM del PC ○ Connessione remota a un server ○ Lettura parametri ed errori del veicolo ○ Importazione/Esportazione dei dati
Scenari alternativi	
Requisiti non funzionali	
Punti aperti	

Titolo	Autenticazione
Descrizione	Autenticazione dell'utente nel sistema
Attori	Utente
Relazioni	Gestione OBDSight
Precondizioni	L’utente deve essere registrato nel sistema
Postcondizioni	Deve essere effettuato l’accesso all’account dell’utente
Scenario principale	1. Vengono inseriti nell’apposito modulo, nome utente e password 2. Viene verificato che i dati inseriti siano corretti, altrimenti viene visualizzato un messaggio di errore 3. Viene effettuato l’accesso all’account
Scenari alternativi	
Requisiti non funzionali	
Punti aperti	

Titolo	Registrazione
Descrizione	Registrazione dell'utente nel sistema
Attori	Utente
Relazioni	
Precondizioni	● L'utente deve possedere un indirizzo email con cui registrarsi ● L'utente non deve essere già registrato nel sistema
Postcondizioni	Deve essere aggiunto un nuovo account utente nel sistema
Scenario principale	1. Una volta scelti dall'utente, vengono inseriti nell'apposito modulo, nome utente, email e password 2. Viene verificato che non esistano nomi utente o email già utilizzati 3. Viene creato il nuovo account
Scenari alternativi	1. Utente già registrato, si invita l'utente ad effettuare l'autenticazione 2. Dati immessi errati o già presenti nel sistema, si invita l'utente ad inserire nuovamente i dati.
Requisiti non funzionali	
Punti aperti	

Titolo	Gestione comandi
Descrizione	Visualizzazione dei comandi disponibili e relativa manipolazione
Attori	Utente
Relazioni	Gestione OBDSight
Precondizioni	● L'utente deve essere autenticato ● Il PC deve essere collegato all'interfaccia oppure ad un Server remoto
Postcondizioni	● Deve essere notificato all'utente l'invio del comando ● Deve essere visualizzata la risposta prodotta dal comando
Scenario principale	1. L'utente seleziona un comando 2. Il comando viene inviato 3. Viene visualizzata la risposta
Scenari alternativi	Il comando produce una risposta inattesa, viene visualizzato un errore
Requisiti non funzionali	
Punti aperti	

Titolo	Gestione dei dati
Descrizione	Accesso al File System del dispositivo locale
Attori	Utente
Relazioni	Gestione OBDSight
Precondizioni	● L’utente deve aver effettuato l’accesso nel sistema
Postcondizioni	Possibilità di accedere al File System
Scenario principale	1. L’Utente può importare e/o esportare dei file di log
Scenari alternativi	
Requisiti non funzionali	
Punti aperti	

Titolo	Connessione
Descrizione	Creazione di una connessione valida
Attori	Utente
Relazioni	Gestione OBDSight, Locale, Remoto
Precondizioni	● Nel caso di connessione remota deve essere in esecuzione un Server ● Nel caso di connessione locale l'interfaccia OBD-II deve essere collegata al computer tramite porta COM
Postcondizioni	Viene realizzato il collegamento all'interfaccia, in locale o in remoto
Scenario principale	1. L'utente seleziona dal menù a tendina la modalità di connessione 2. Nel caso di connessione remota inserisce indirizzo IP e porta del Server; nel caso di connessione locale seleziona la porta COM 3. Appare un messaggio di avvenuta connessione
Scenari alternativi	Non è possibile effettuare la connessione, viene restituito un messaggio di errore
Requisiti non funzionali	
Punti aperti	

Titolo	Locale
Descrizione	Creazione di una connessione all'interfaccia
Attori	Utente
Relazioni	Connessione
Precondizioni	● L'Utente deve essere autenticato nel sistema ● L'interfaccia OBD-II deve essere collegata al computer tramite porta COM
Postcondizioni	Viene realizzato il collegamento all'interfaccia
Scenario principale	1. L'utente seleziona la COM dell'interfaccia OBD da un menù a tendina 2. Il sistema stabilisce la connessione all'interfaccia
Scenari alternativi	1. Nessuna COM presente, l'utente può aggiornare la lista mediante opportuno tasto Refresh 2. Impossibile collegarsi all'interfaccia mediante la COM selezionata, viene visualizzato un errore
Requisiti non funzionali	
Punti aperti	

Titolo	Remoto
Descrizione	Collegamento ad una interfaccia OBD mediante connessione remota
Attori	Utente
Relazioni	Connessione
Precondizioni	● Il software OBDSight deve essere in funzione sia sul dispositivo Server che sul dispositivo Client ● L'Utente deve essere autenticato nel sistema ● Deve essere in esecuzione un Server ● L'Utente deve essere in possesso di un IP di un Server in ascolto ● Il Server in ascolto deve essere collegato ad un'interfaccia OBD
Postcondizioni	Il collegamento remoto è stabilito con successo
Scenario principale	1. L'utente specifica di voler avviare una connessione remota con ruolo di server o client 2. Il sistema crea un socket di rete per poter permettere lo scambio di informazioni
Scenari alternativi	1. L'utente specifica di voler avviare una connessione remota con ruolo di server: si stabilisce la connessione all'interfaccia OBD e viene creato un socket di rete sul quale porsi in ascolto 2. L'utente specifica di voler avviare una connessione remota con ruolo di client: il sistema fa immettere all'utente un IP e una porta relativi ad un server in ascolto e tenta di effettuare una connessione, se l'esito è negativo viene notificato all'utente
Requisiti non funzionali	La comunicazione deve essere sincrona e non bloccante.
Punti aperti	

Analisi Del Rischio

Valutazione dei Beni

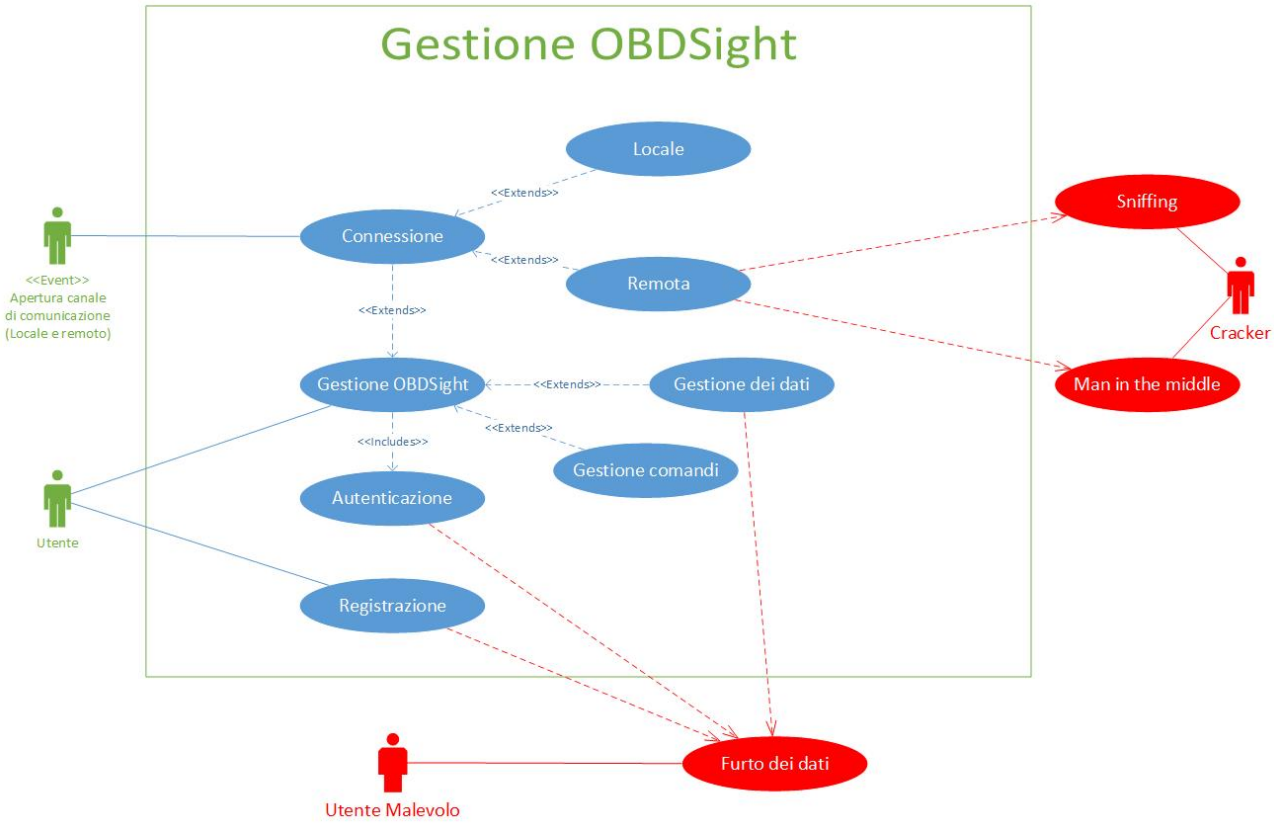
Bene	Valore	Esposizione
Interfaccia	Alto Scambio di informazioni con il veicolo	Alto Mancata rilevazione di risposte significative
Risposta	Medio Dati ottenuti da un'analisi	Bassa Perdita di dati relativi ad un'analisi, che può essere ripetuta
Sistema informativo	Alto Supporto all'autenticazione degli utenti	Alto Rischio di furto d'identità e compromissione degli utenti
Credenziali degli utenti	Medio Informazioni relative agli utenti del sistema, tra cui username e password, che permettono di autenticarsi	Media Perdita d'immagine se l'utente viene compromesso

Analisi minacce e controlli

Minaccia	Probabilità	Controllo	Fattibilità
Furto d'identità utente	Alta	Log degli accessi	Basso costo e trasparente
Alterazione dei dati da e verso unità remota (e attacchi Man in the middle)	Bassa	Uso di canale sicuro SSL	Costo medio, assicura l'integrità dei dati scambiati
Attacchi DDoS verso il database	Media	Controllo e limitazione degli accessi	Costo basso: non è possibile prevenire questo tipo di attacchi

Analisi Tecnologica della Sicurezza

Tecnologia	Vulnerabilità
Architettura Client/Server	● Attacco Man in the Middle ● Intercettazione (sniffing) delle comunicazioni ● Attacco DoS
Sistema di identificazione	● L'utente rivela le sue credenziali volontariamente ● L'utente sceglie credenziali facili da indovinare ● Vengono sottratte le credenziali all'utente mediante phishing
Cifratura delle comunicazioni	Bisogna utilizzare sia una cifratura simmetrica che asimmetrica, ognuna di queste comporta delle vulnerabilità: Cifratura simmetrica: ● Lunghezza della chiave: chiavi corte possono essere scovate con brute force ● Memorizzazione della chiave: la chiave deve essere memorizzata in maniera sicura ● Tempo di vita della chiave: la chiave deve essere sostituita periodicamente, se non si sostituisce e si cifrano troppe informazioni con la stessa chiave un attaccante può calcolarla. Cifratura asimmetrica: ● La chiave privata deve essere salvata in maniera sicura e non accessibile ● Falsificazione della chiave pubblica: la chiave pubblica può essere falsificata in quanto, in questo progetto, si è deciso di non utilizzare una Certificate Authority (CA) ● Una chiave corta può essere decifrata tramite brute force


Gestione OBDSight
Locale
<<Extends>>
Connessione
<<Extends>>
Remota
<<Extends>>
Gestione OBDSight
<<Extends>>
Gestione dei dati
<<Extends>>
<<Includes>>
Gestione comandi
Autenticazione
Registrazione
<<Event>>
Apertura canale
di comunicazione
(Locale e remoto)
Utente
Sniffing
Cracker
Man in the middle
Furto dei dati
Utente Malevolo

Analisi del problema

Analisi Documento dei Requisiti

Analisi delle Funzionalità

Tabella funzionalità

Funzionalità	**Tipo**	**Grado Complessità**
Autenticazione	Gestione e memorizzazione dati	Complessa
Gestione Comandi	Gestione e memorizzazione dati	Semplice
Gestione dei dati	Gestione e memorizzazione dati	Semplice
Connessione	Gestione e memorizzazione dati, interazione con l'esterno	Complessa

Autenticazione: Tabella Informazioni/flusso

Informazione	**Tipo**	**Livello protezione/privacy**	**Input/Output**	**Vincoli**
Email	Semplice	Alta	Input	Non più di 80 caratteri, deve contenere il carattere "@" e il carattere "."
Username	Semplice	Molto alta	input	Non più di 20 caratteri
Password	Semplice	Molto alta	input	Non più di 80 caratteri

Gestione comandi: Tabella Informazioni/flusso

Informazione	**Tipo**	**Livello protezione/privacy**	**Input/Output**	**Vincoli**
Nome	Semplice	Bassa	Output	
Testo del comando	Semplice	Bassa	Output	
Esito del comando	Semplice	Media	Output	Non più di 80 caratteri

Gestione dei dati: Tabella Informazioni/flusso

Informazione	**Tipo**	**Livello protezione/privacy**	**Input/Output**	**Vincoli**
Diagnosi	Composto	Alta	Input	
Data	Semplice	Media	Input	Nel formato dd/mm/yyyy
Ora	Semplice	Media	Input	Nel formato hh:mm:ss
Username	Semplice	Alta	Input	

Connessione: Tabella Informazioni/Flusso

Informazione	**Tipo**	**Livello protezione/privacy**	**Input/Output**	**Vincoli**
Porta COM	Semplice	Bassa	Input	Deve essere una porta COM valida
IP	Semplice	Media	Input	Deve essere un indirizzo IP valido

Analisi dei vincoli

Tabella dei vincoli

Requisito	Categoria	Impatto	Funzionalità
Integrità dei dati	Integrità	Peggiora la velocità di scrittura e di trasmissione ma garantisce migliore protezione e qualità superiore dei dati	Gestione OBDSight, Gestione dei dati, Esportazione dei dati, Importazione dei dati, Gestione comandi, Autenticazione, Registrazione, Connessione remota, Connessione locale
Sicurezza dei dati	Sicurezza	Peggiora il tempo di risposta ma migliora la privacy dei dati memorizzati	Registrazione, Autenticazione, Connessione remota, Connessione locale
Controllo Accessi	Sicurezza	Peggiora il tempo di risposta ma migliora la privacy dei dati	Autenticazione, Connessione
Facilità di navigazione delle schermate	Usabilità	Migliorare l'interfaccia utente	Gestione OBDSight, Autenticazione, Gestione comandi, Gestione dei dati, Connessione locale, Connessione remota
Rapidità di lettura e scrittura dati	Tempo di risposta	Migliorare il tempo di risposta	Gestione comandi
Efficienza analisi	Efficienza	Migliorare la qualità e l'affidabilità del risultato	Gestione comandi
Tempo di risposta	Tempo di inoltramento	Utilizzare protocolli di comunicazione migliori	Gestione comandi

Analisi delle interazioni

Tabella maschere

Maschera	Informazioni	Funzionalità
View OBDSight	Log in tempo reale contenente i risultati ottenuti dall'elaborazione dei dati in ingresso, possibilità di di navigazione verso le altre schermate	Gestione OBDSight
View OBDAuth	Input per username, password	Autenticazione
View OBDReg	Input per email, username, password	Registrazione
View OBDCommands	Menù a tendina contenente lista comandi	Gestione comandi
View OBDData	Bottoni per importazione ed esportazione file *.txt	Gestione dei dati
View OBDLocal	Connessione alla porta seriale COM	Connessione locale
View OBDRemote	Input per indirizzo IP, porta	Connessione remota

Tabella sistemi esterni

Sistema	Descrizione	Protocollo di interazione	Livello di sicurezza
Interfaccia OBD-II	Sistema hardware che invia/riceve flussi di informazioni dal veicolo al computer e viceversa	Il sensore invia tramite porta seriale un flusso continuo di informazioni al computer; può ricevere comandi dal computer relativi a funzioni per la lettura/scrittura dei dati del veicolo. Il sistema è in ascolto su tale interfaccia per effettuare la lettura	Medio. L'interfaccia OBD-II non è attaccabile informaticamente, ma fisicamente scollegabile

Tabella dei Ruoli

Ruolo	Responsabilità	Maschere	Riservatezza	Numerosità
Utente Lato Server	Invio informazioni al Client	OBDSight, OBDAuth, OBDReg, OBDCommands, OBDData, OBDRemote, OBDLocal	Richiesto un alto grado di riservatezza	Singola
Utente Lato Client	Gestione delle informazioni dal Server	OBDSight, OBDAuth, OBDReg, OBDCommands, OBDData, OBDRemote, OBDLocal	Richiesto un alto grado di riservatezza	Non definibile a priori

Utente Lato Client: Tabella Ruolo-Informazioni

Informazione	Tipo di accesso
Username	Lettura/Scrittura
Password	Scrittura
Nome dispositivo	Lettura
Log delle informazioni	Lettura
Informazioni lette dal dispositivo	Lettura/Scrittura

Utente Lato Server: Tabella Ruolo-Informazioni

Informazione	Tipo di accesso
Username	Lettura/Scrittura
Password	Scrittura
Account Connessi	Lettura
Log delle informazioni	Lettura
Informazioni lette dal dispositivo	Lettura/Scrittura

Scomposizione del problema

Tabella Scomposizione Funzionalità

Funzionalità	Scomposizione
Gestione OBD Sight	Invio e ricezione dei dati
Gestione dei dati raccolti	Esportazione dei dati in locale Importazione dei dati da locale

Gestione OBD Sight

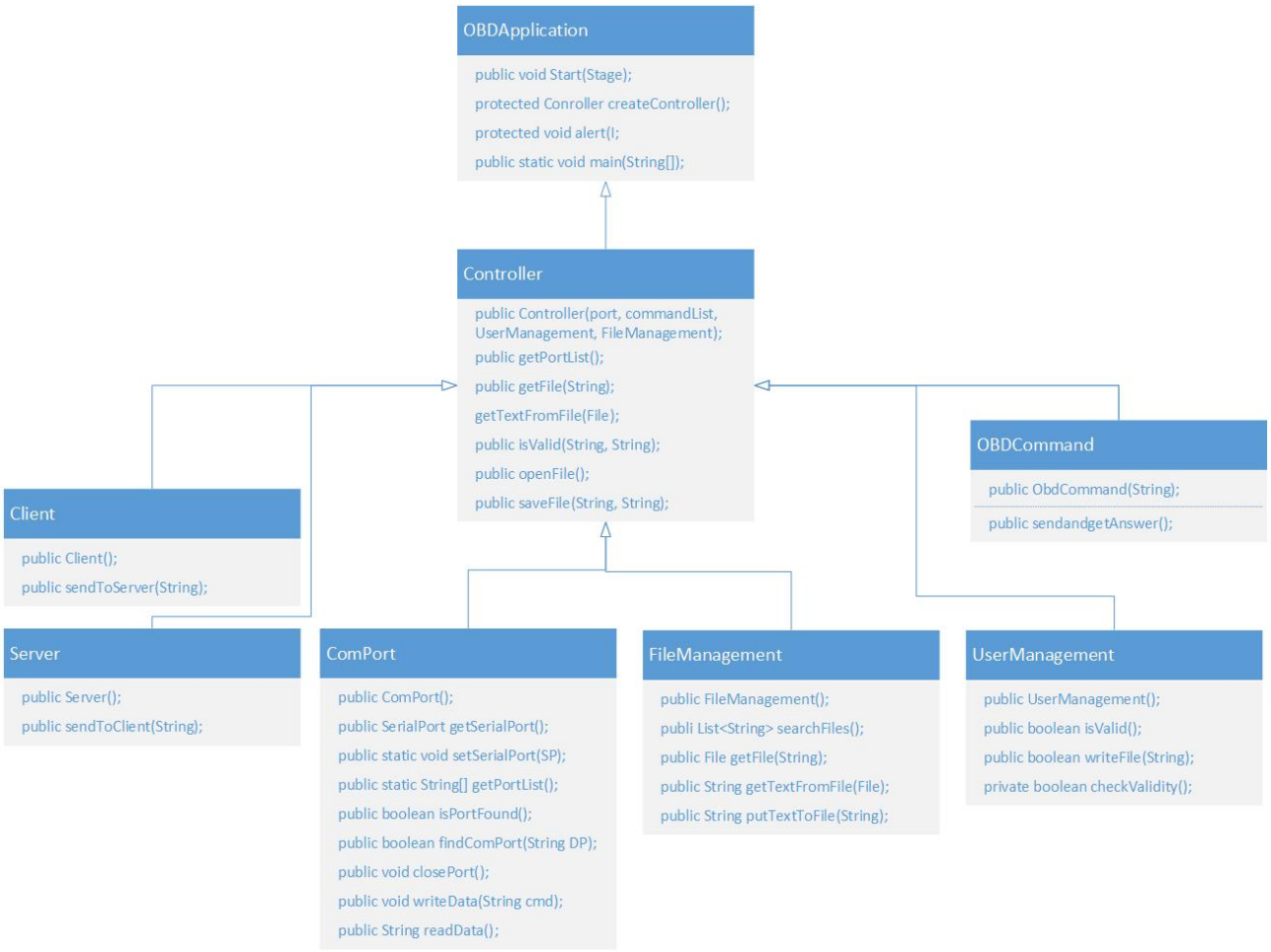
Sotto-Funzionalità	Sotto-Funzionalità	Legame	Informazioni
Ricezione dei dati	Invio dei dati	L'invio si basa sui dati ricevuti	Utente, Codici identificativi, Valori dei dati

Gestione dei dati raccolti

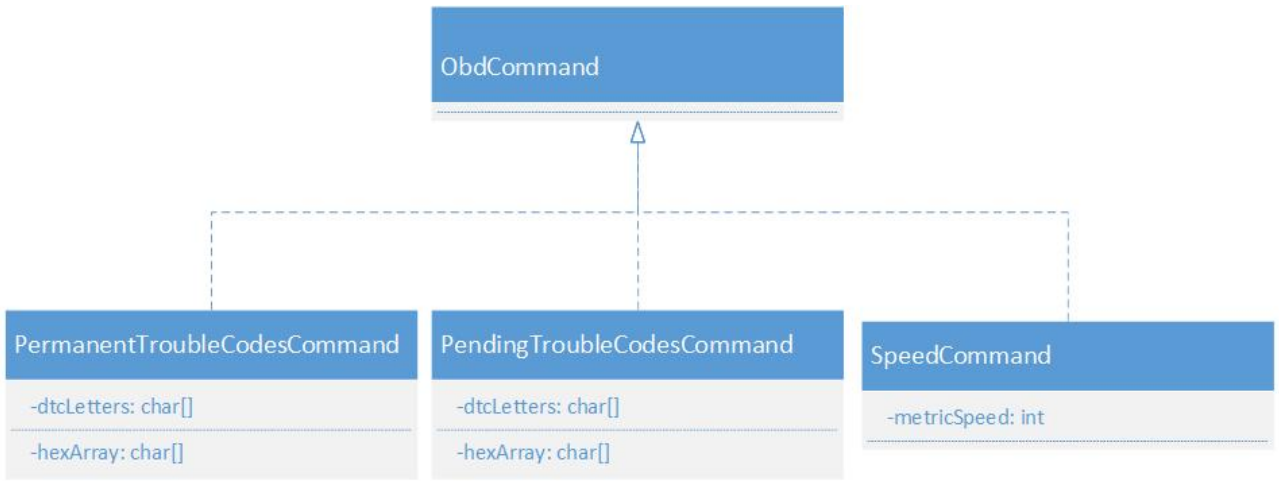
Sotto-Funzionalità	Sotto-Funzionalità	Legame	Informazioni
Importazione dei dati dal locale	Esportazione dei dati dal locale	Importazione ed esportazione si basano sui dati ricevuti al momento dell'analisi	Utente, Codici identificativi, Valori dei dati

Modello del dominio

Il seguente diagramma mostra, in forma sintetica, il diagramma del dominio dell'applicativo.



Il seguente diagramma mostra la parte di modello del dominio relativa al funzionamento dei comandi OBD.



Architettura logica: Struttura

Diagramma dei package

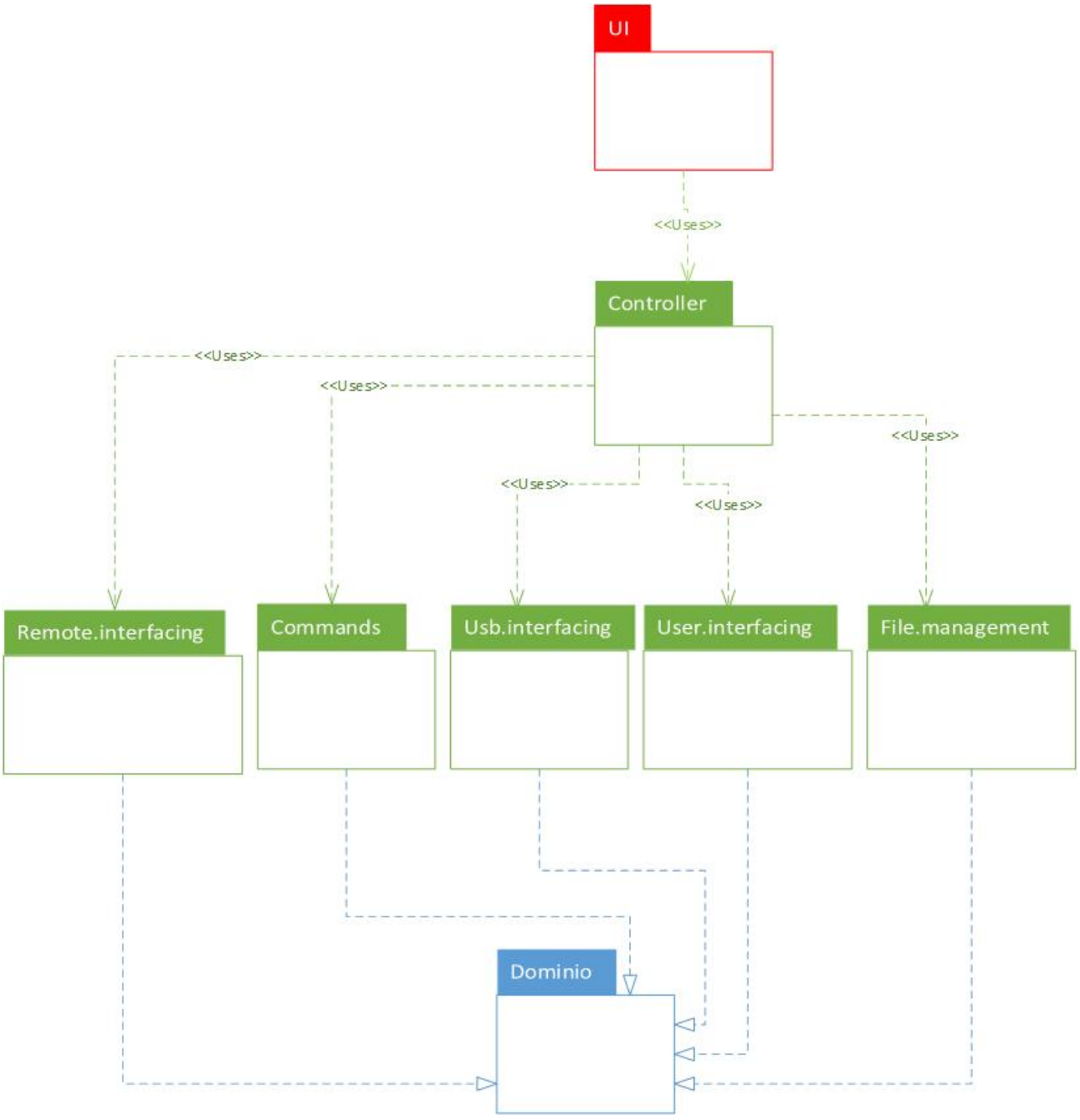


Diagramma delle classi: UI

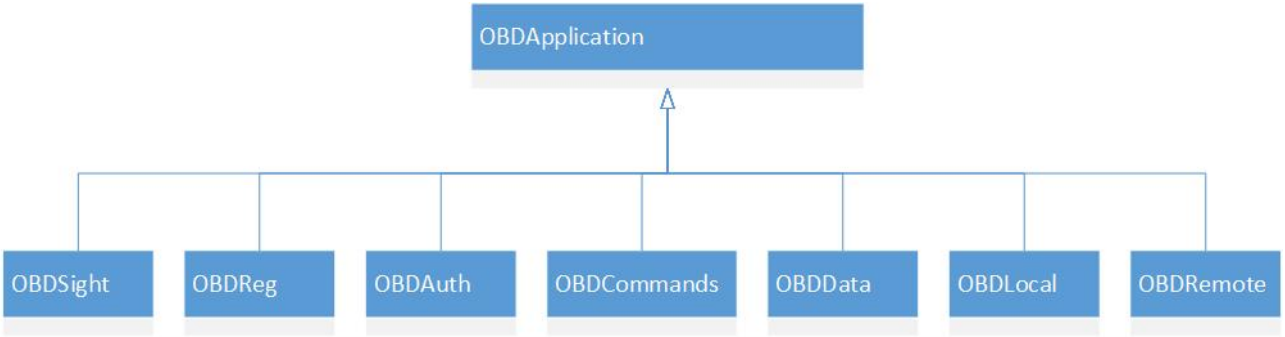


Diagramma delle classi: exceptions

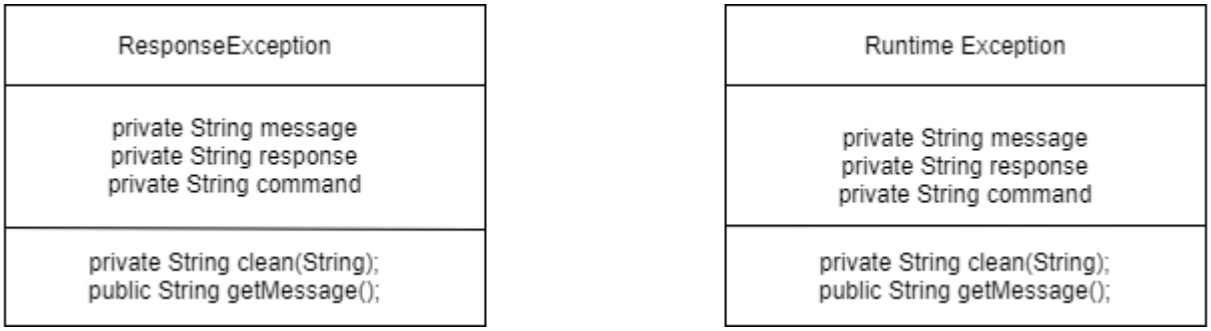


Diagramma delle classi: user.interfacing

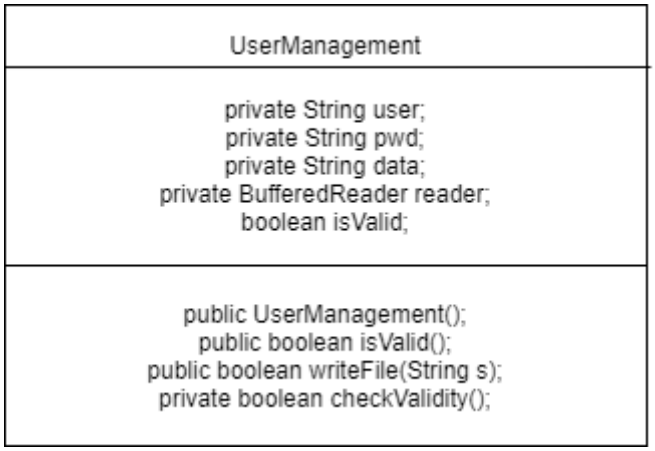


Diagramma delle classi: usb.Interfacing

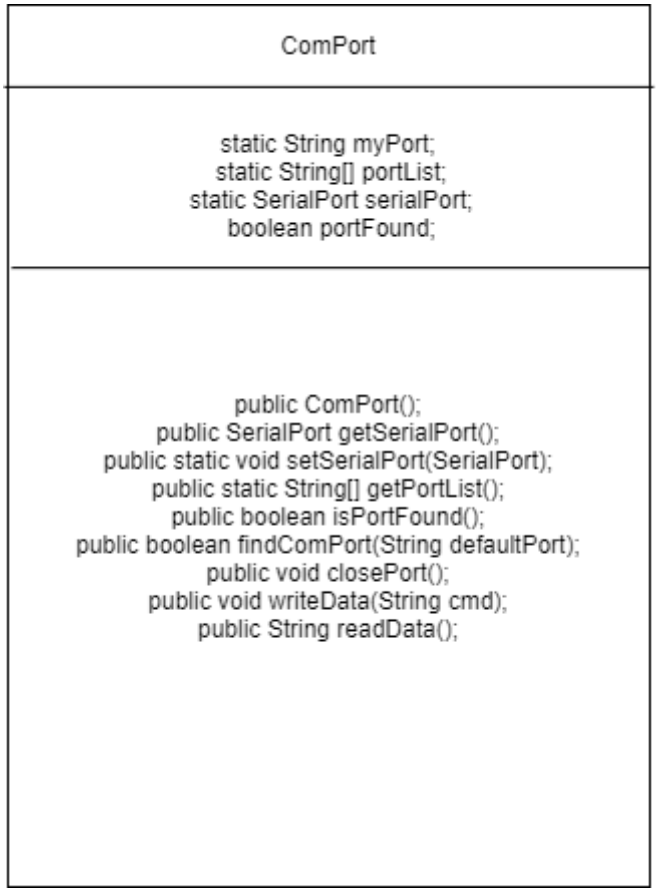
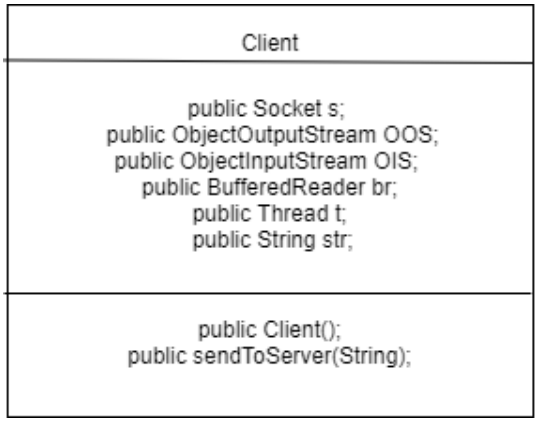


Diagramma delle classi: remote.Interfacing



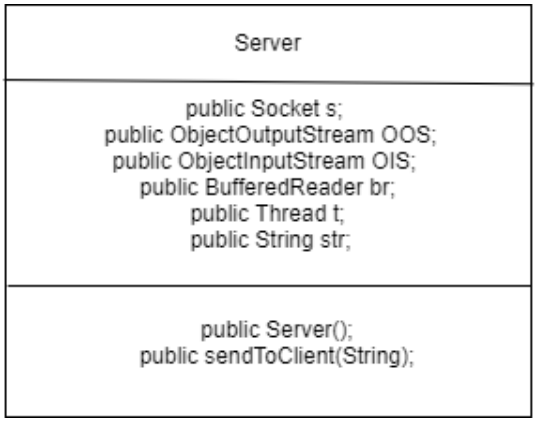


Diagramma delle classi: controller

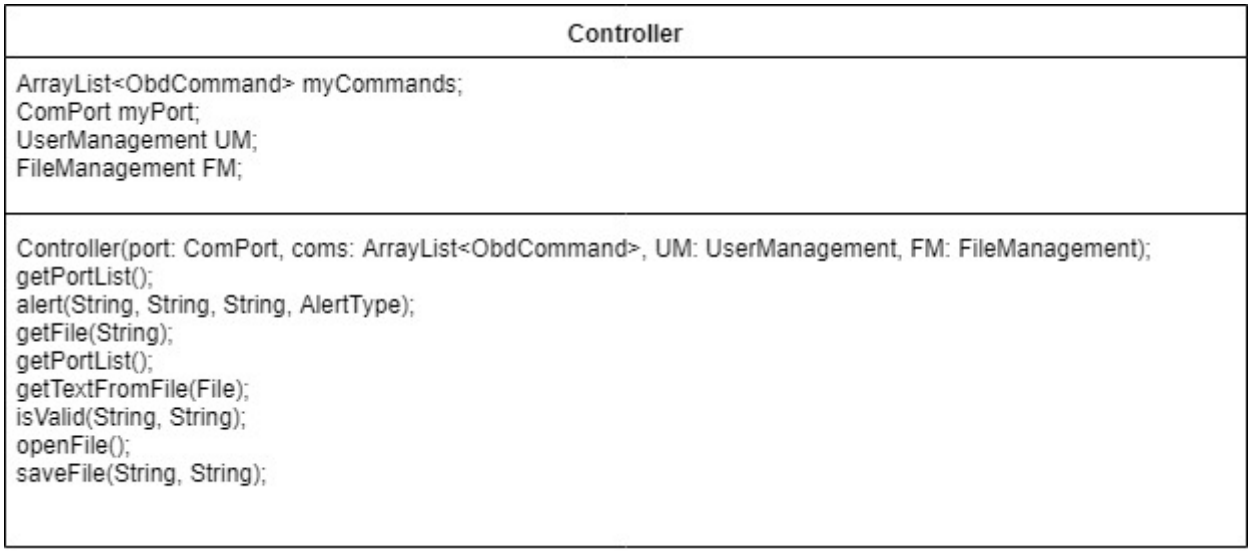


Diagramma delle classi: obd.Commands

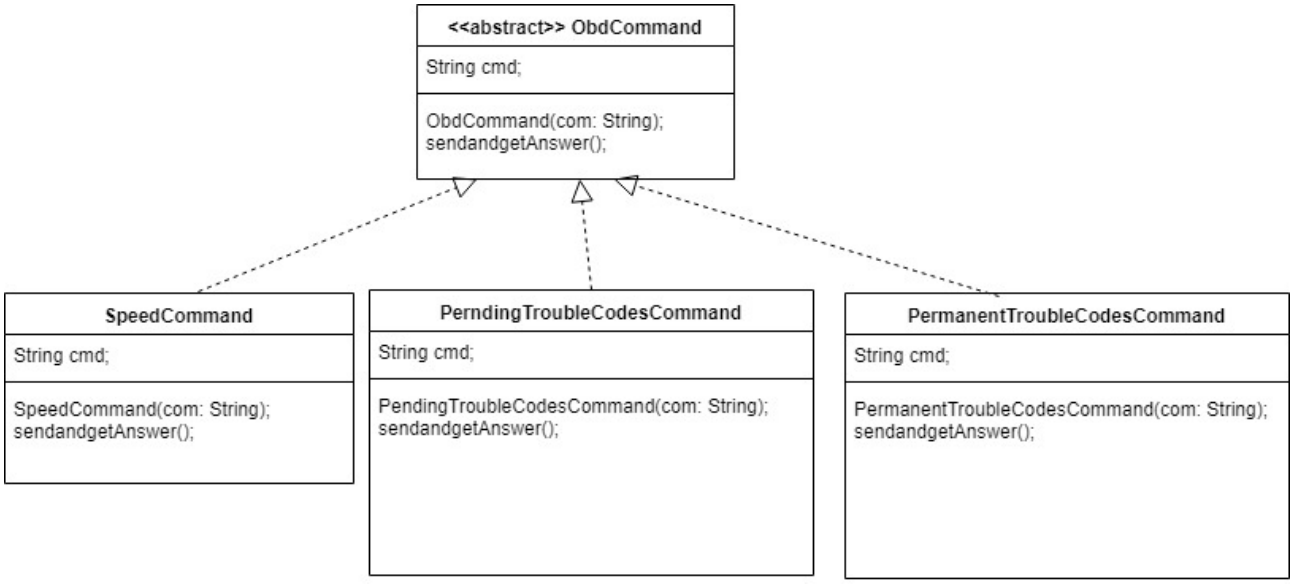
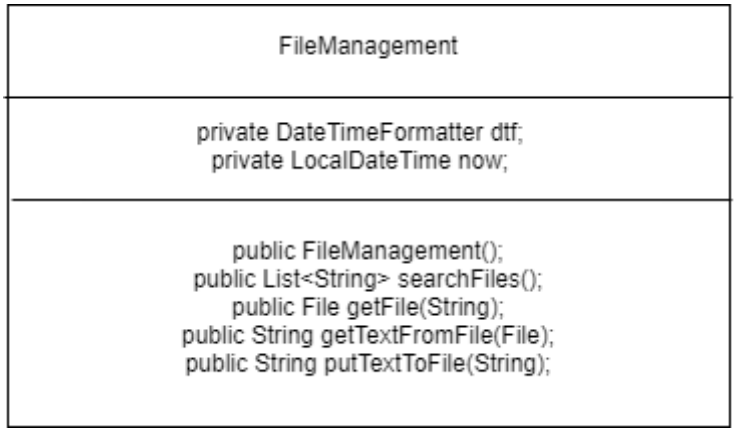


Diagramma delle classi: file.Management



Architettura logica: Interazione

Interazione

Diagramma di sequenza: registrazione

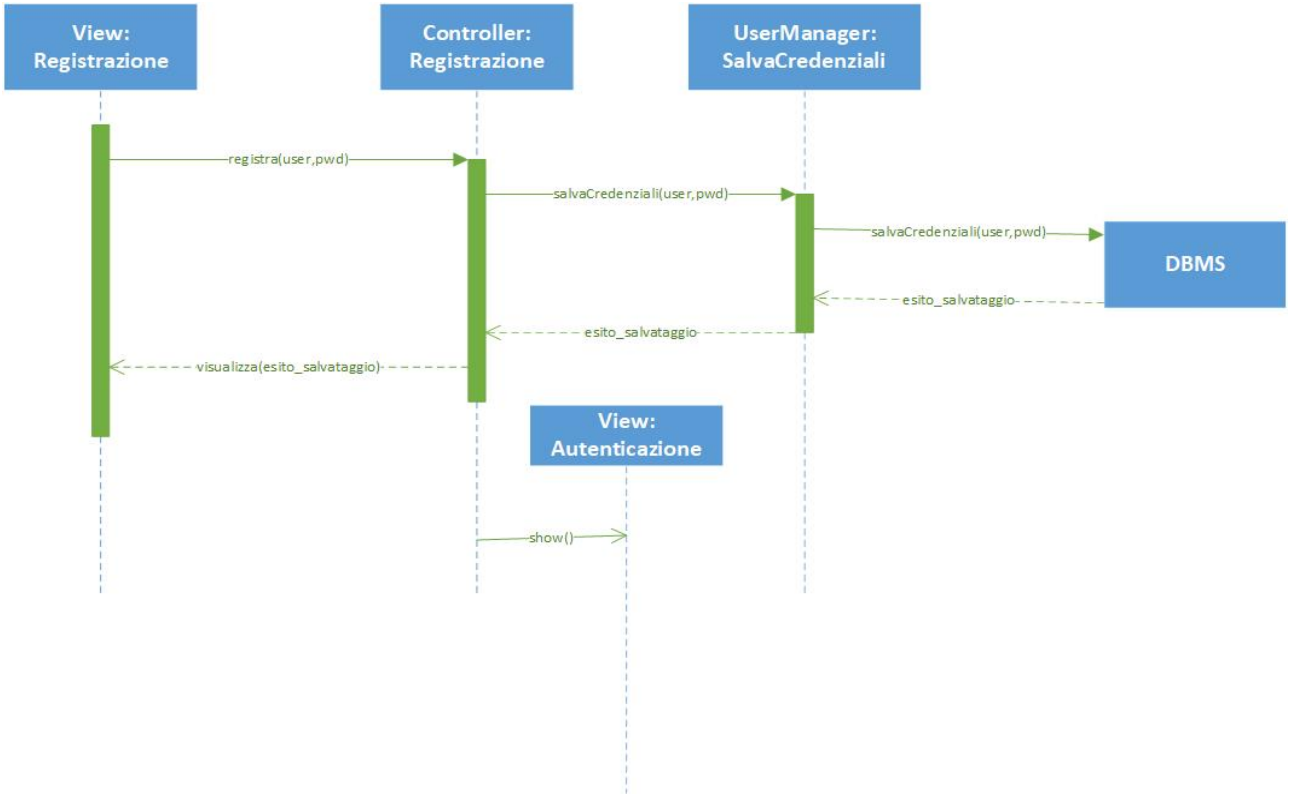


Diagramma di sequenza: autenticazione

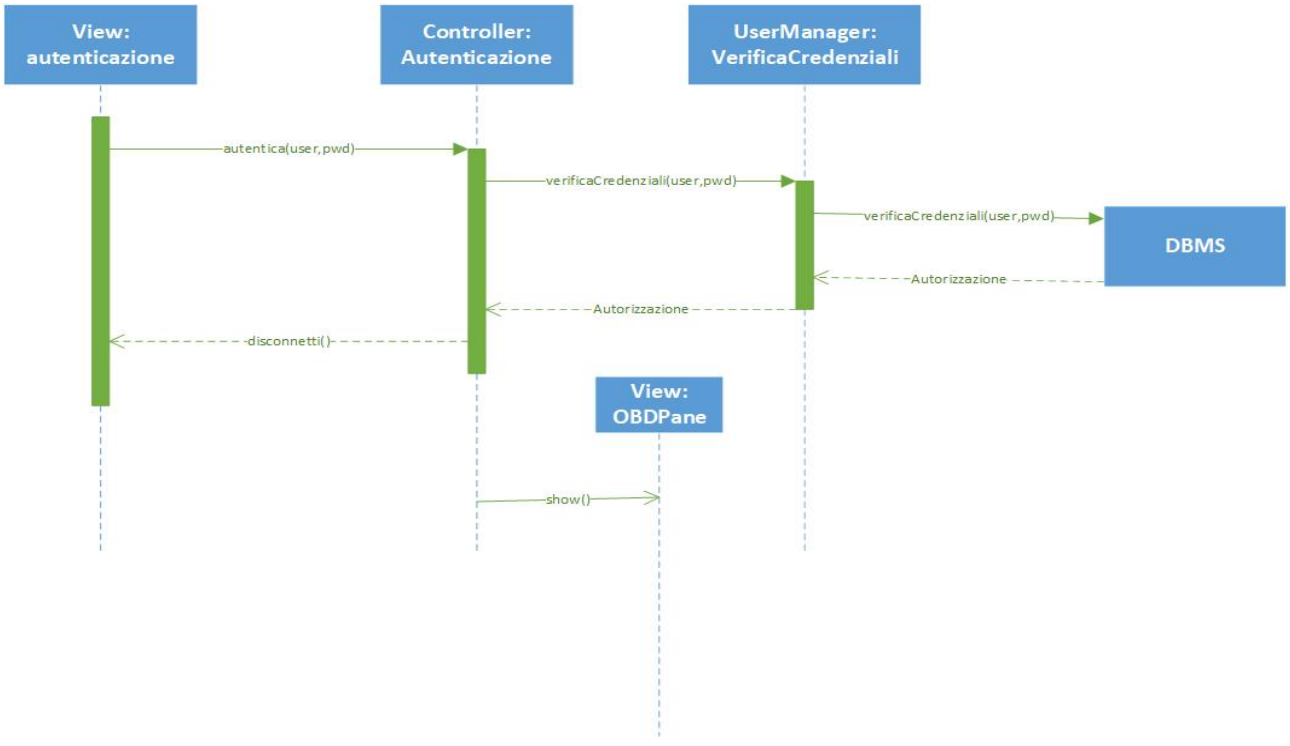


Diagramma di sequenza: invio di un comando in locale

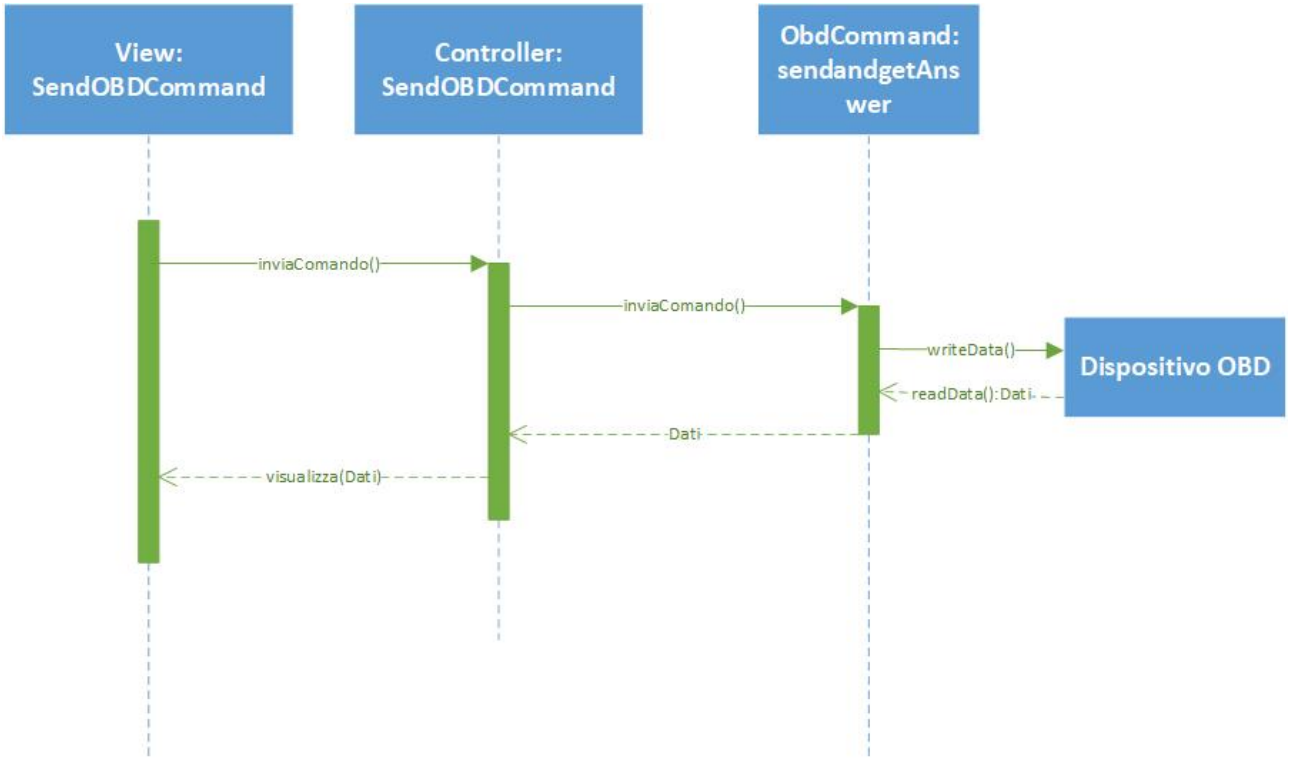


Diagramma di sequenza: invio di un comando in remoto

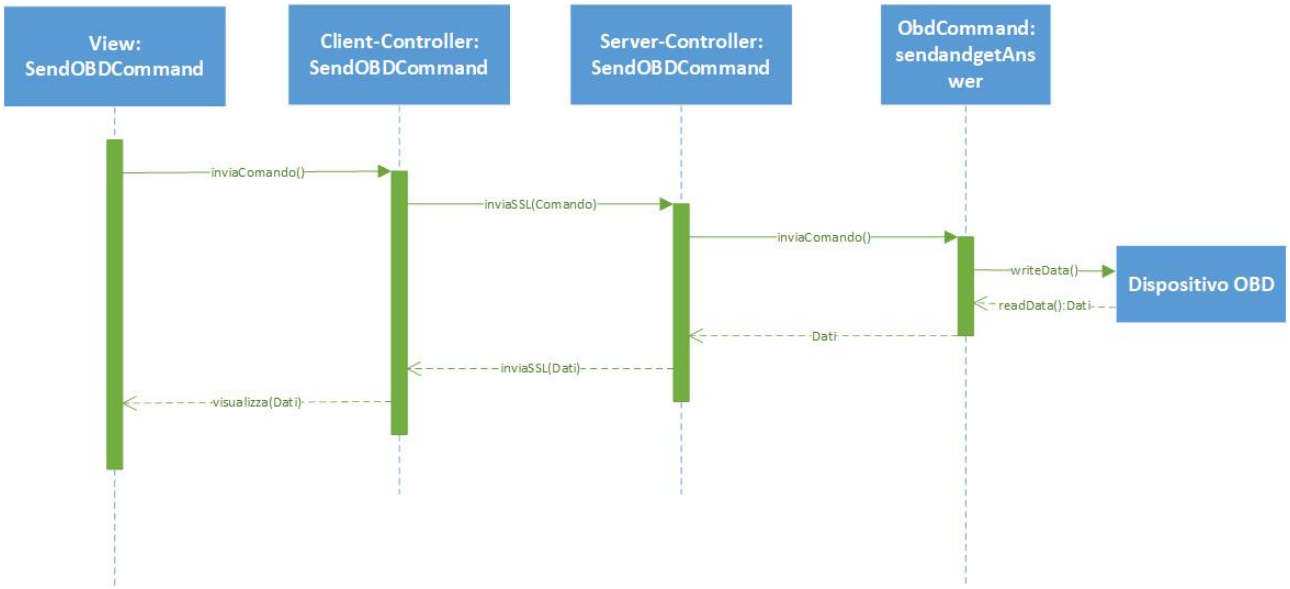


Diagramma di sequenza: importazione di un log

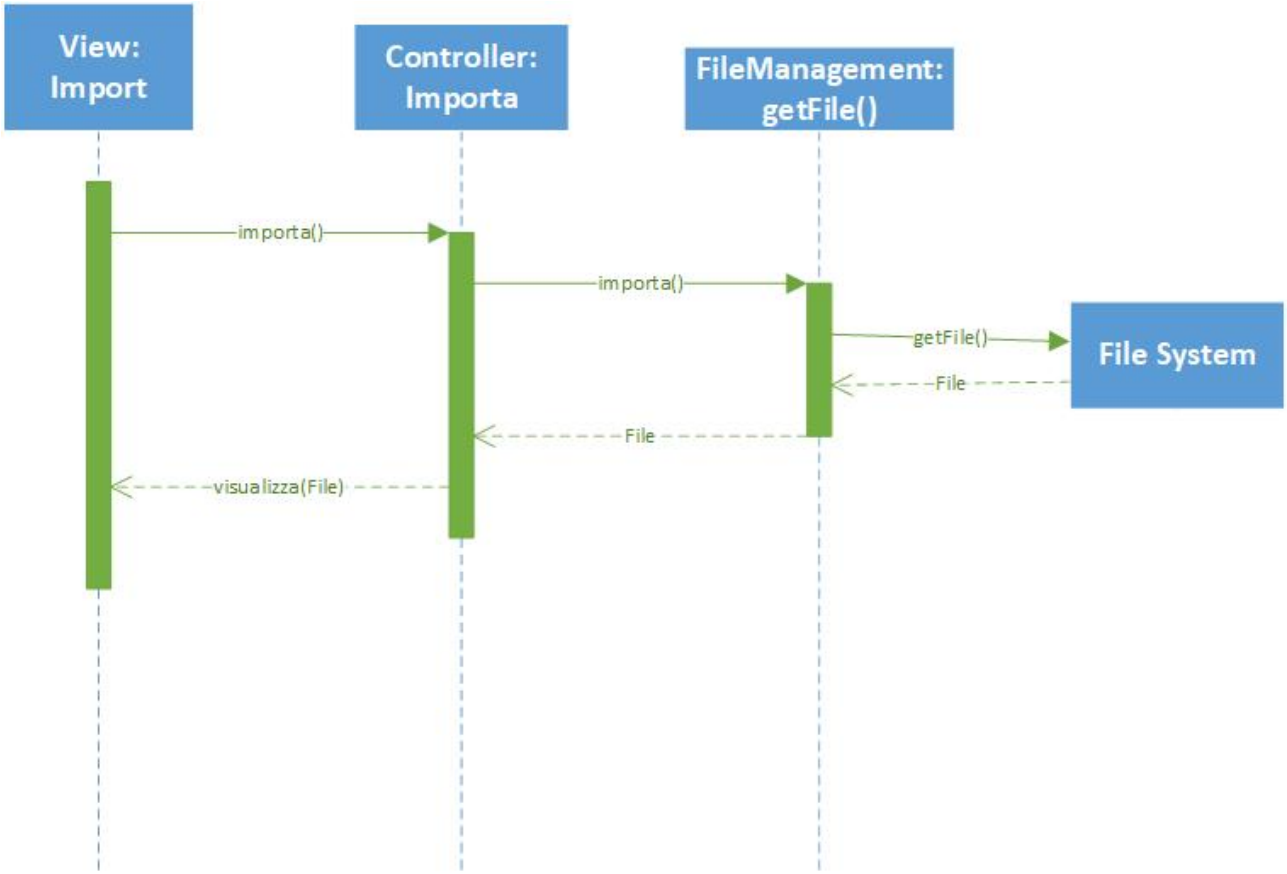
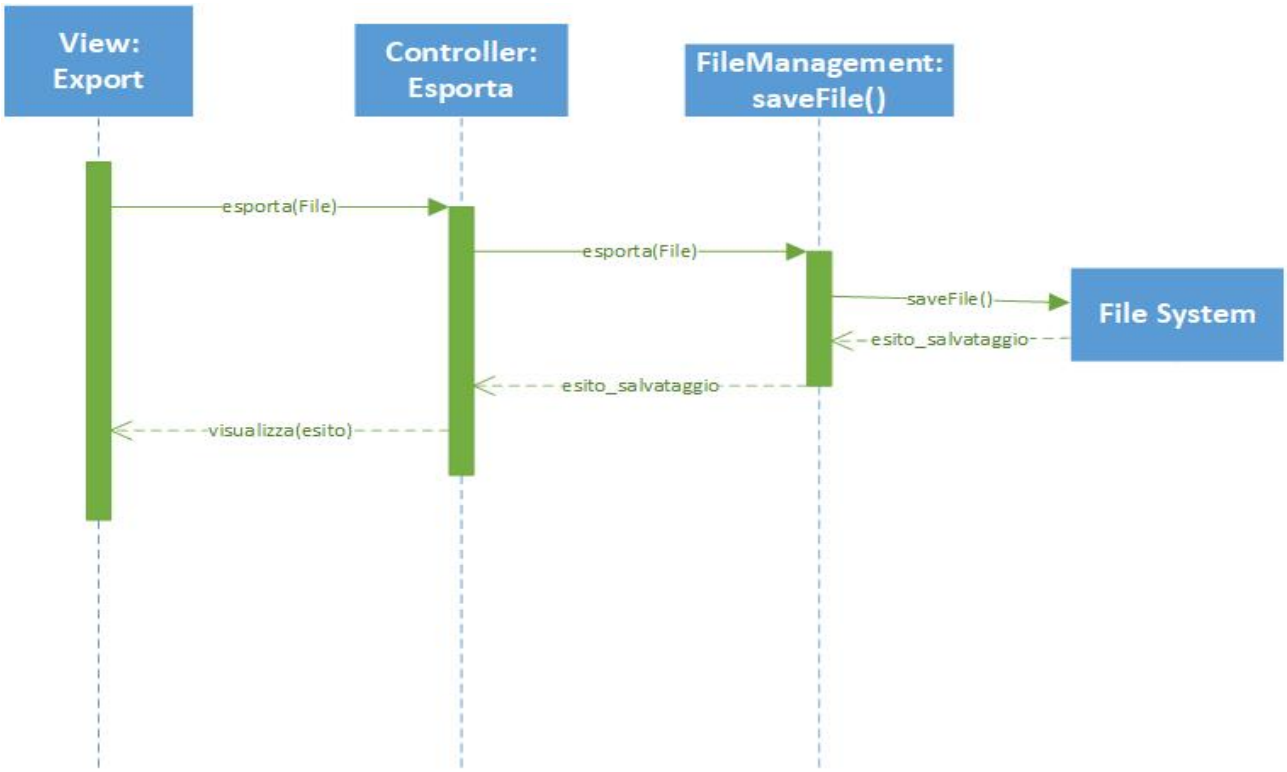


Diagramma di sequenza: esportazione di un log



Architettura logica: Comportamento

Diagramma delle attività: Server

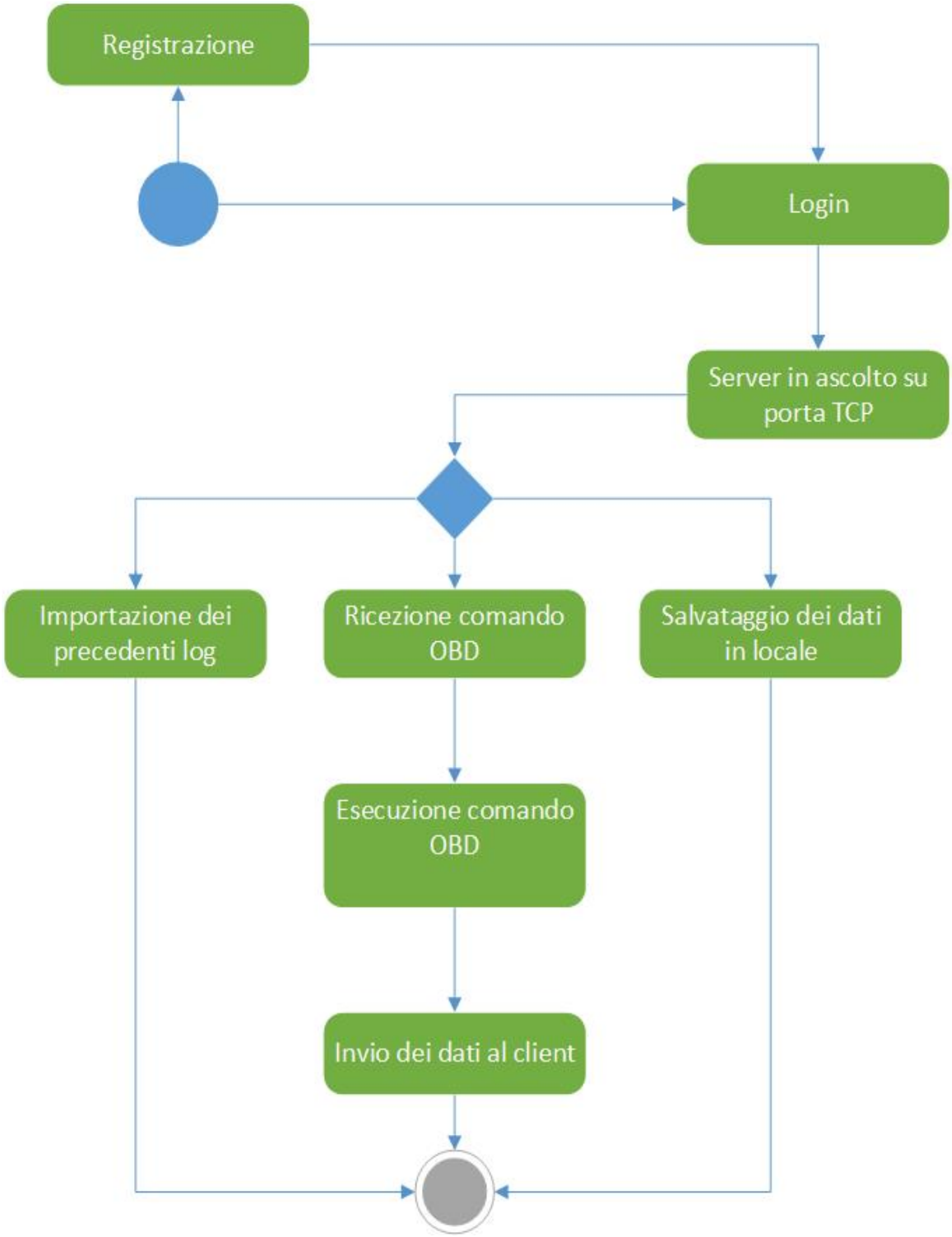
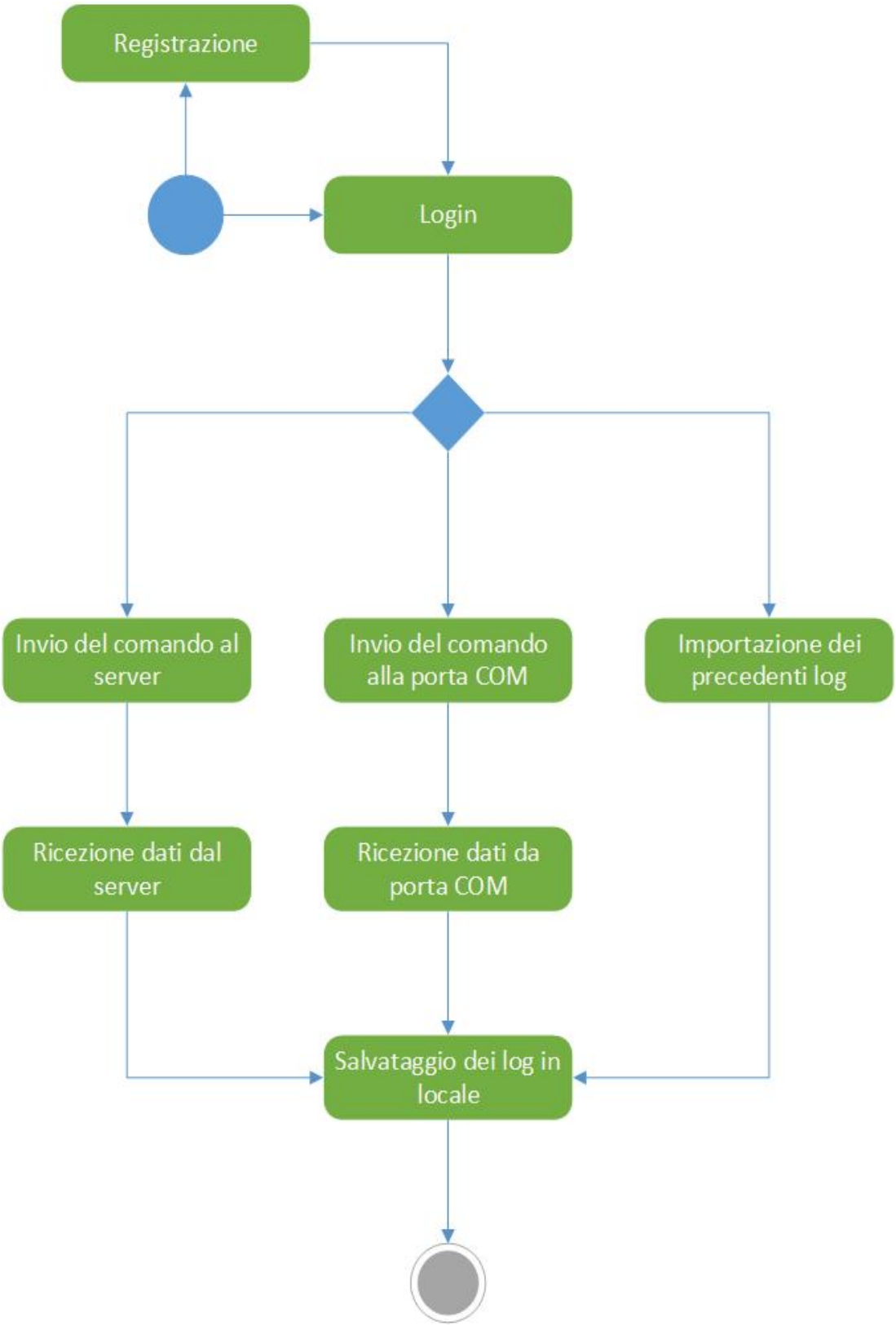


Diagramma delle attività: Client



Piano di Lavoro

Package	**Progetto**	**Sviluppo**
Commands	Daniel, Luca, Matteo	Daniel, Luca, Matteo
Controller	Daniel, Luca, Matteo	Daniel, Luca, Matteo
Enums	Daniel, Luca, Matteo	Daniel, Luca, Matteo
Exceptions	Daniel, Luca, Matteo	Daniel, Luca, Matteo
File.management	Daniel, Luca, Matteo	Matteo
Remote.interfacing	Daniel, Luca, Matteo	Luca
Test	Daniel, Luca, Matteo	Daniel
UI	Daniel, Luca, Matteo	Daniel
Usb.interfacing	Daniel, Luca, Matteo	Luca
User.interfacing	Daniel, Luca, Matteo	Matteo

Il prototipo da consegnare dovrà realizzare le funzionalità di: autenticazione, rilevazione dell'interfaccia, importazione ed esportazione delle diagnosi e avvio di comunicazione remota.

Tempi di rilascio

• Progettazione entro 30 giorni dal 2 Agosto 2020

• Sviluppo delle singole parti con collaudo unitario entro 10 giorni rispetto al fine della progettazione

• Integrazione e test dell'intero sistema entro 2 settimane rispetto alla fine dello sviluppo

Sviluppi futuri: la possibile aggiunta di nuovi comandi del tipo ObdCommand all'applicativo per poter espandere le funzionalità offerte in fase di diagnosi, implementazione di un'interfaccia di registrazione utente.

Piano del Collaudo

Sono state realizzate opportune classi di test per verificare il funzionamento di ogni singolo componente del Model e del Controller, si riporta ad esempio i test della classe Controller, che contribuisce alla realizzazione della View.

```
package obd.tests;

import static org.junit.jupiter.api.Assertions.*;

class MyControllerTest {

	@Test
	void startTest() {
		ComPort cp = new ComPort();
		ArrayList<ObdCommand> list = new ArrayList<ObdCommand>();
		UserManagement UM = new UserManagement();
		FileManagement FM = new FileManagement();
		Controller C = new Controller(cp, list, UM, FM);
		assertTrue(C.getMyCommands().isEmpty());
	}
	@Test
	void myPortTest() {
		ComPort cp = new ComPort();
		ArrayList<ObdCommand> list = new ArrayList<ObdCommand>();
		UserManagement UM = new UserManagement();
		FileManagement FM = new FileManagement();
		Controller C = new Controller(cp, list, UM, FM);
		assertTrue(C.getMyPort().equals(cp));
	}
	@Test
	void getCommandsTest() {
		ComPort cp = new ComPort();
		ArrayList<ObdCommand> list = new ArrayList<ObdCommand>();
		UserManagement UM = new UserManagement();
		FileManagement FM = new FileManagement();
		Controller C = new Controller(cp, list, UM, FM);
		assertTrue(C.getMyCommands().equals(list));
	}
```

```
	@Test
	void getPortListTest() {
		ComPort cp = new ComPort();
		ArrayList<ObdCommand> list = new ArrayList<ObdCommand>();
		UserManagement UM = new UserManagement();
		FileManagement FM = new FileManagement();
		Controller C = new Controller(cp, list, UM, FM);
		assertTrue(C.getPortList().equals(new ArrayList<String>(Arrays.asList(cp.getPortList()))));
	}

	@Test
	void getFileTest(){
		ComPort cp = new ComPort();
		ArrayList<ObdCommand> list = new ArrayList<ObdCommand>();
		UserManagement UM = new UserManagement();
		FileManagement FM = new FileManagement();
		Controller C = new Controller(cp, list, UM, FM);
		String nomeFile = "provaTest";
		assertTrue(C.getFile(nomeFile).equals(FM.getFile(nomeFile)));
	}
}
```

Progetto

PROGETTAZIONE ARCHITETTURALE

Requisiti non funzionali

Dall'analisi dei requisiti sono emersi alcuni vincoli non funzionali, tra cui:

- Sicurezza ed integrità dei dati
- Facilità di utilizzo

L'integrità e la sicurezza dei dati sono due aspetti molto importanti, soprattutto nell'ambito della comunicazione remota. L'utilizzo del protocollo SSL per crittografare la comunicazione permette di rispettare questi requisiti.

Per facilitare l'utilizzo del programma all'utente si è optato per una grafica semplice ed intuitiva con dati precaricati dal Controller, in questo modo l'utente dovrà immettere manualmente solo un sottoinsieme ristretto dei dati (Username, Password ed eventuale indirizzo IP per la connessione remota).

Scelta dell'architettura

Dal punto di vista dell'architettura la scelta più idonea per questo tipo di sistema è un'architettura MVC (Model View Controller).

Model:
Il Model gestisce i dati e le entità in gioco quali:

- Utente
- Server
- File
- Interfaccia OBD

View:
La view mostra i dati e le azioni eseguite dal software OBDSight dalla seguente maschera:

- View OBDSight

Controller:
Il controller gestisce le seguenti funzionalità:

- Comunicazione con l'interfaccia OBD-II
- Comunicazione con Server
- Gestione dei dati
- Gestione autenticazione

Scelte tecnologiche

Il linguaggio di programmazione scelto è il Java in quanto i computer di bordo delle autovetture, i lettori multimediali e stereo sono per la maggior parte basati su software compilato per Java Virtual Machine. Pertanto in un'ottica di sviluppo si prevede che in futuro il software possa essere installato nei sistemi a bordo delle vetture senza la necessità di hardware esterni.

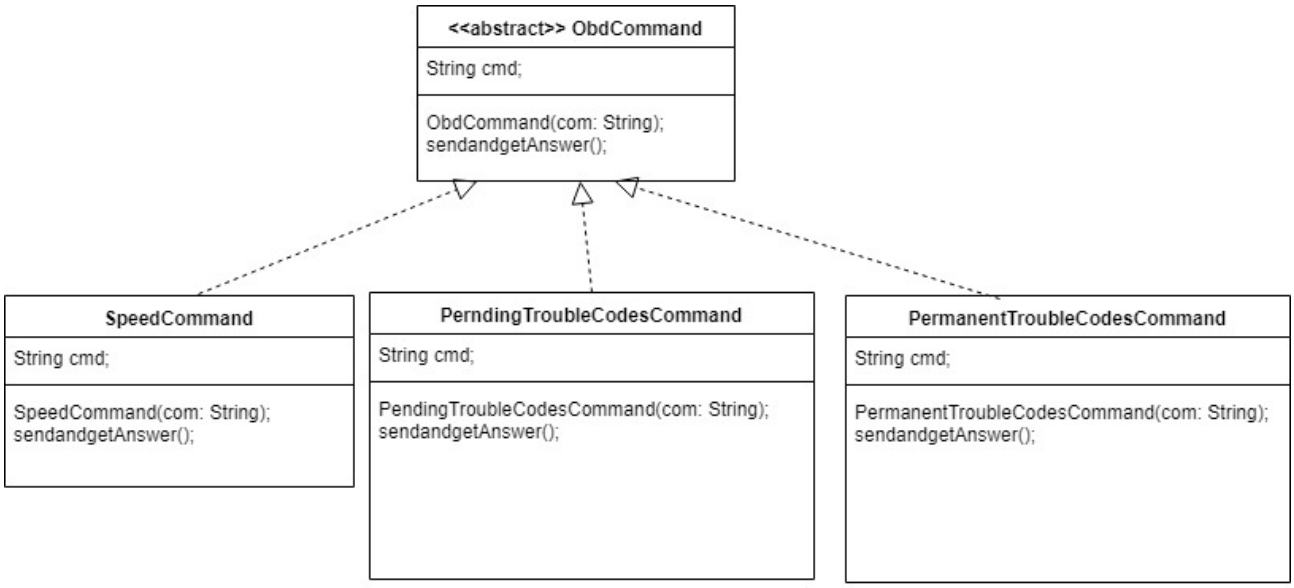
PROGETTAZIONE DI DETTAGLIO

Il dominio realizzato in fase di analisi è stato rispettato, la struttura del sistema richiama il modello MVC (Model-View-Controller) e si rimanda alla parte di analisi per il diagramma delle classi.

Struttura

In fase di progettazione le classi sono rimaste invariate rispetto all'analisi, ad eccezione delle classi relative all'interfaccia grafica; le si riportano di seguito con una breve descrizione testuale.

package obd.commands



Nel modello sopra riportato è rappresentata la struttura dei vari comandi che è possibile inviare al sistema, ogni comando è rappresentato da una stringa e ad ognuno è associato un metodo sendandgetAnswer() che permette di inviare un comando tramite l'interfaccia seriale e ottenere la relativa risposta.

La classe astratta ObdCommand implementa i metodi base e può essere estesa per integrare ulteriori comandi di tipo OBD.

package obd.controller

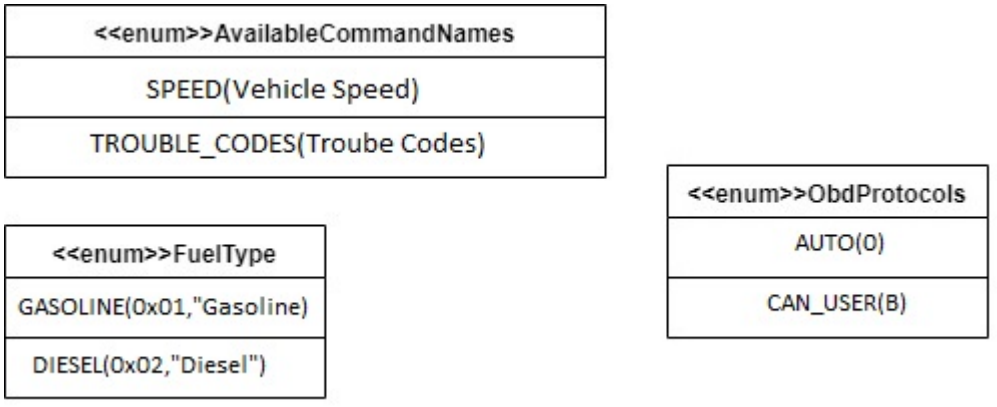
Controller
ArrayList<ObdCommand> myCommands; ComPort myPort; UserManagement UM; FileManagement FM;

Controller(port: ComPort, coms: ArrayList<ObdCommand>, UM: UserManagement, FM: FileManagement);
getPortList();
alert(String, String, String, AlertType);
getFile(String);
getPortList();
getTextFromFile(File);
isValid(String, String);
openFile();
saveFile(String, String);

La classe Controller nel nostro modello di progettazione si occupa di gestire l'interazione da parte dell'utente con la view ed accedere ai dati prodotti dal model.

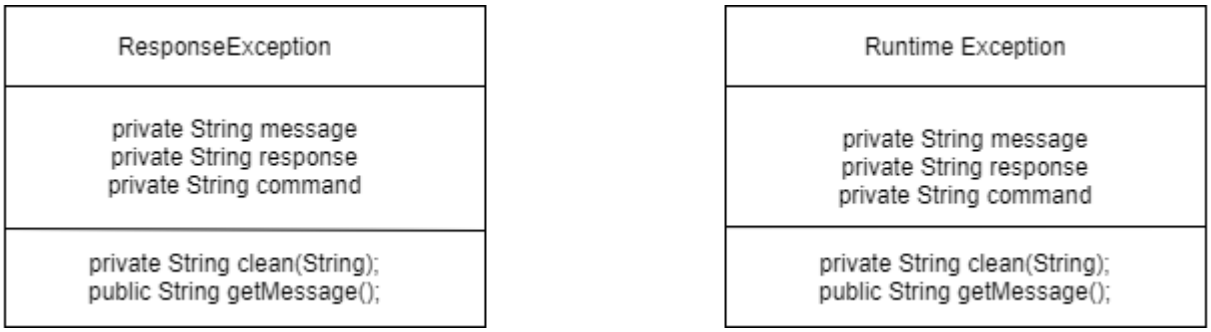
La classe offre tutti i metodi necessari per poter gestire la view e viene inizializzata fornendo la porta COM a cui collegarsi, un array di comandi OBD, l'oggetto UserManagement che permette di gestire l'identificazione e la registrazione dell'utente e l'oggetto FileManagement che permette di gestire la persistenza.

package obd.enums



Le seguenti classi di tipo ENUM contengono delle costanti che rappresentano i nomi dei comandi disponibili, i tipi di carburante, il livello di di carburante e i vari tipi di protocolli fruibili per la connessione all'interfaccia OBD.

package obd.exceptions



Le due classi sopra rappresentate sono le principali implementazioni per rappresentare le possibili eccezioni che possono essere generate nel sistema; la prima rappresenta le eccezioni che si possono presentare in response mentre la seconda le eccezioni che si possono verificare in runtime.

Le seguenti classi sono implementazione di eccezioni di risposta:

public class BusInitException;

public class MisunderstoodCommandException;

public class NoDataException;

public class ResponseException;

public class StoppedException;

public class UnableToConnectException;

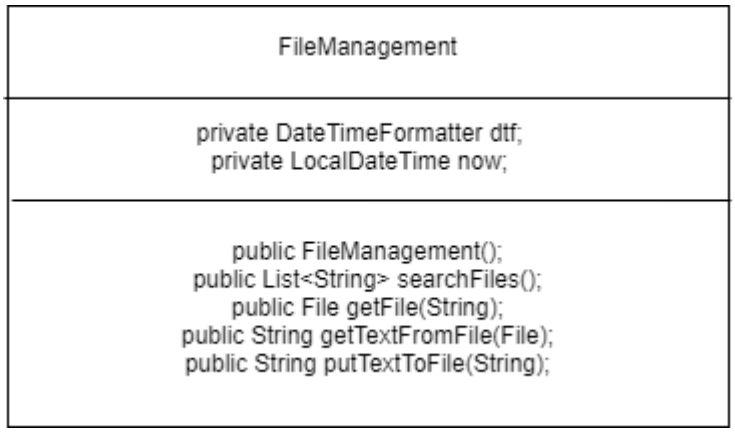
public class UnknownErrorException;

public class UnsupportedCommandException;

Mentre la classe di eccezione durante l'esecuzione viene implementata con la seguente classe:

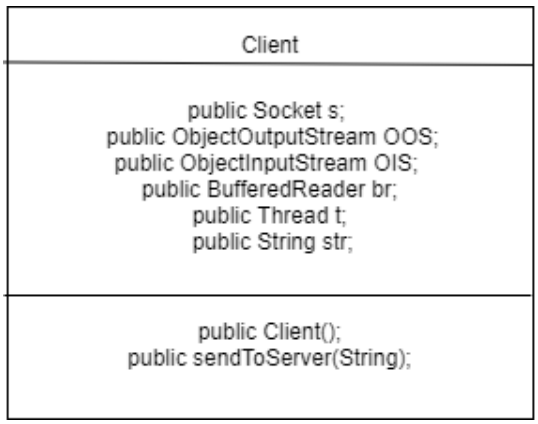
public class NonNumericResponseException;

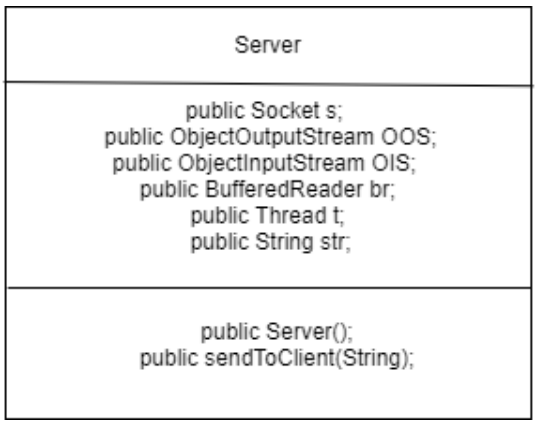
package obd.file.management



La classe sopra rappresentata offre funzionalità per gestire la persistenza, offre metodi per cercare i files, accedere al loro contenuto e modificarli.

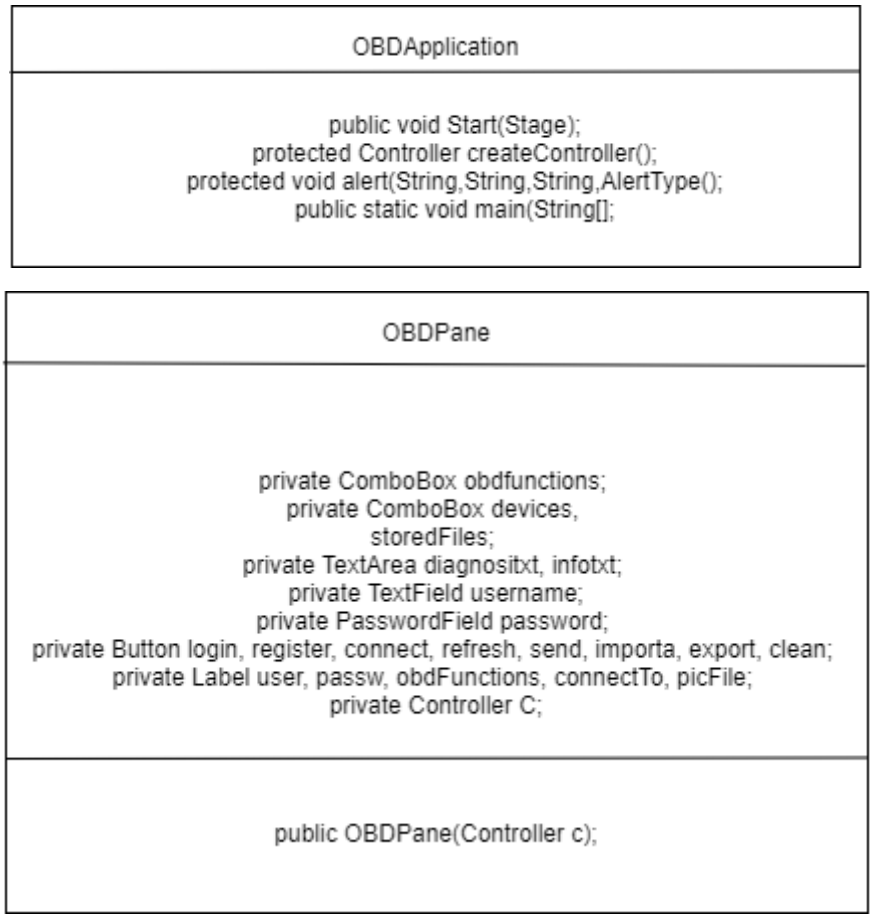
package obd.remote.interfacing





Le classi Client e Server hanno il ruolo di rappresentare dei Thread in esecuzione in background che una volta inizializzati pongono il sistema in ascolto su una porta TCP predefinita; ognuna delle due classi offre un metodo per inviare una stringa (sendToServer e sendToClient) che verrà ricevuta dal destinatario, che resta in ascolto fino a quando la connessione non viene chiusa.

package obd.ui

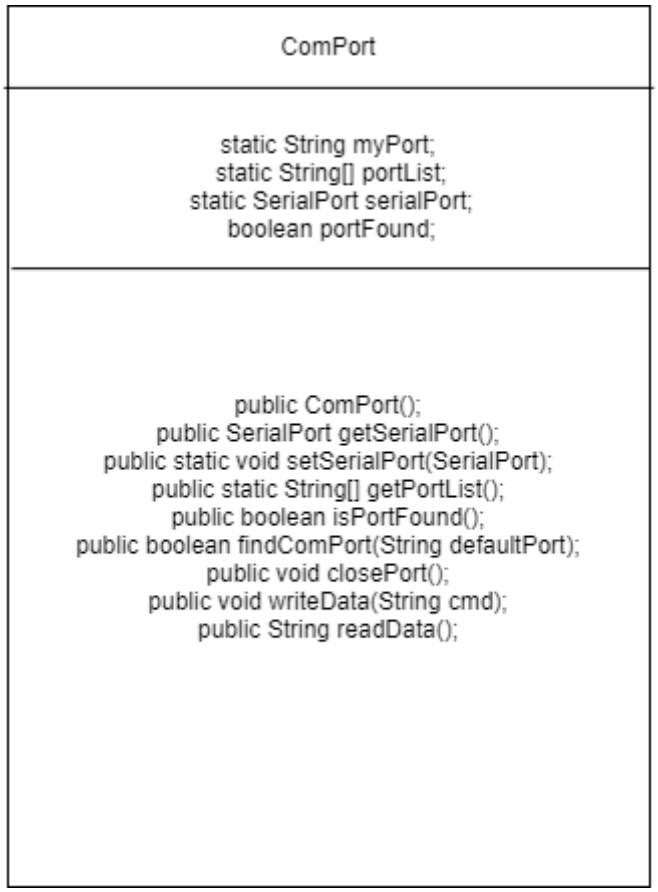


Rispetto all'analisi, abbiamo deciso di implementare un'unica view dinamica all'interno del software che permette di gestire tutte le funzionalità del programma.

La classe OBDPane si occupa della gestione della grafica, inizializzando i componenti grafici presenti nell'applicazione e richiamando i listener di eventi tramite il controller alle interazioni dell'utente.

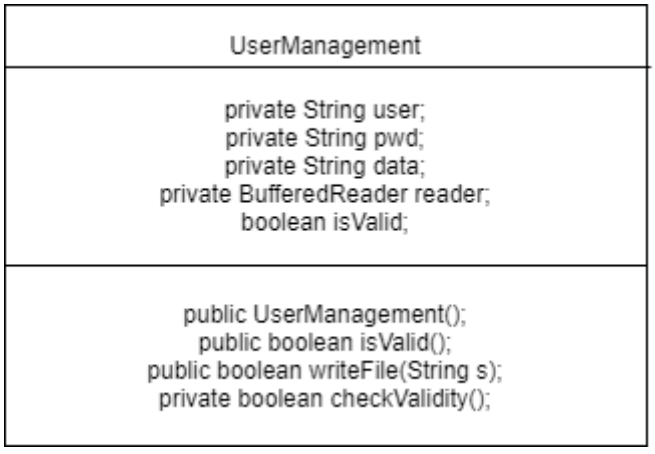
La classe OBDApplication è la principale del sistema e si occupa di inizializzare il Controller e la grafica, ed istanzia le classi necessarie del model.

package obd.usb.interfacing



La classe ComPort permette di rilevare i dispositivi connessi alle porte seriali (nel nostro caso tramite USB) e di aprire una connessione sulla porta selezionata, realizzando appositi buffer di lettura e scrittura.

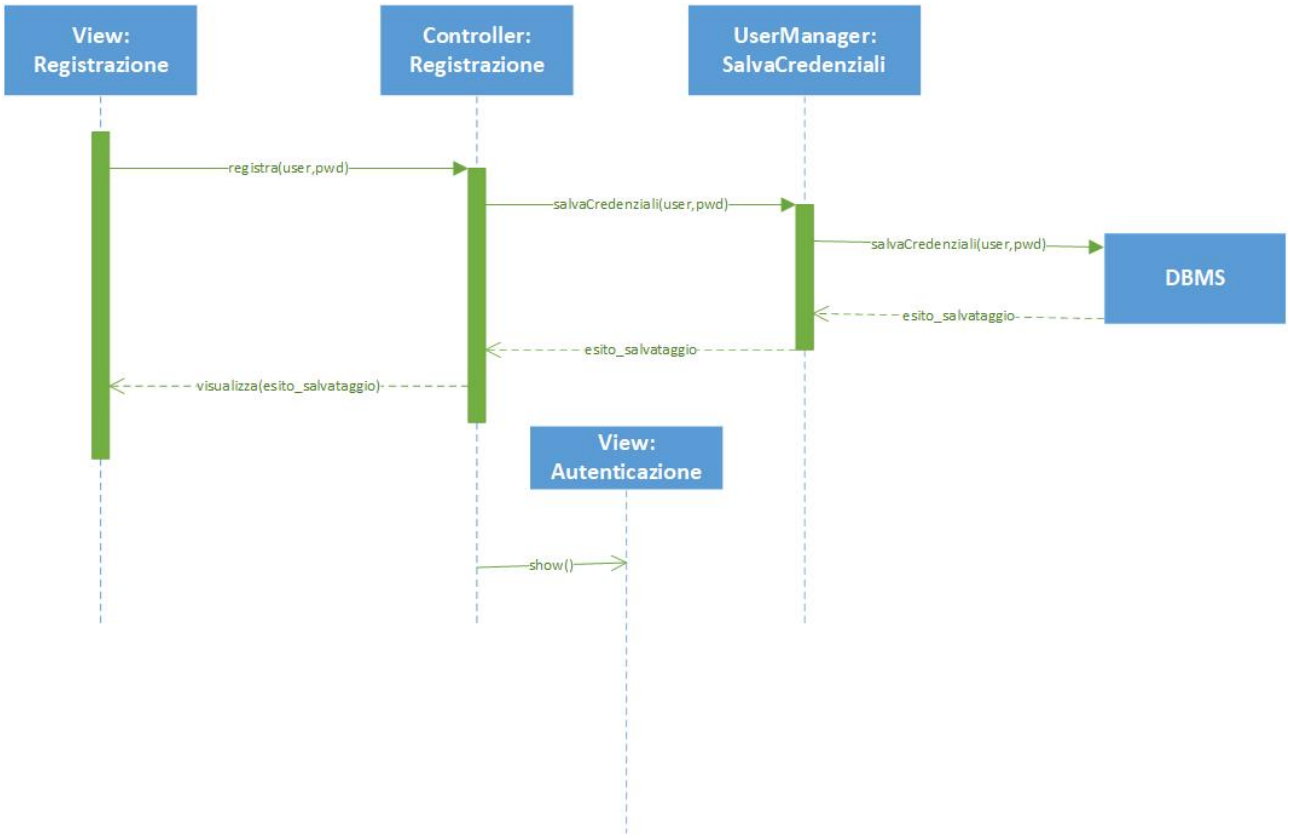
package obd.user.management



La classe UserManagement gestisce l'accesso degli utenti registrati e permette l'inserimento di nuove utenze.

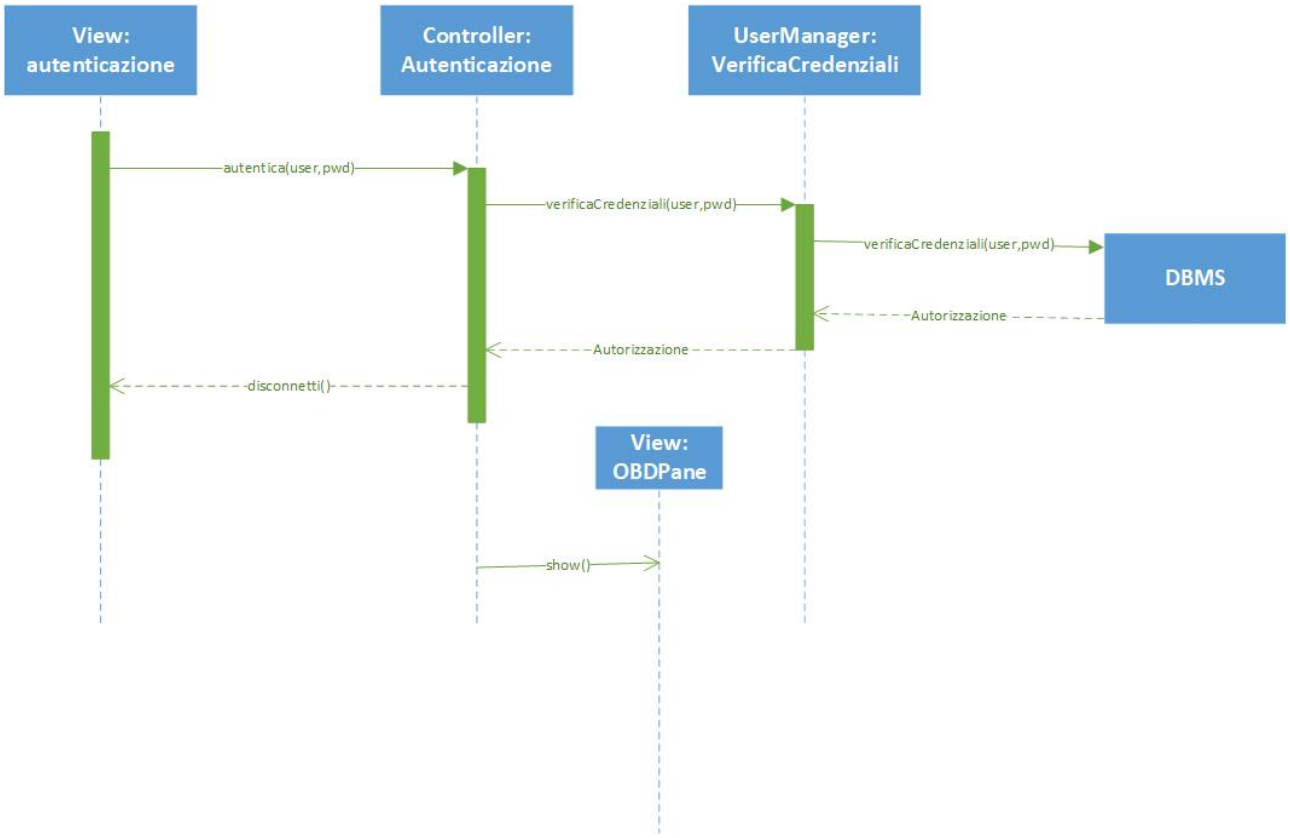
Interazione

Diagramma di sequenza: registrazione



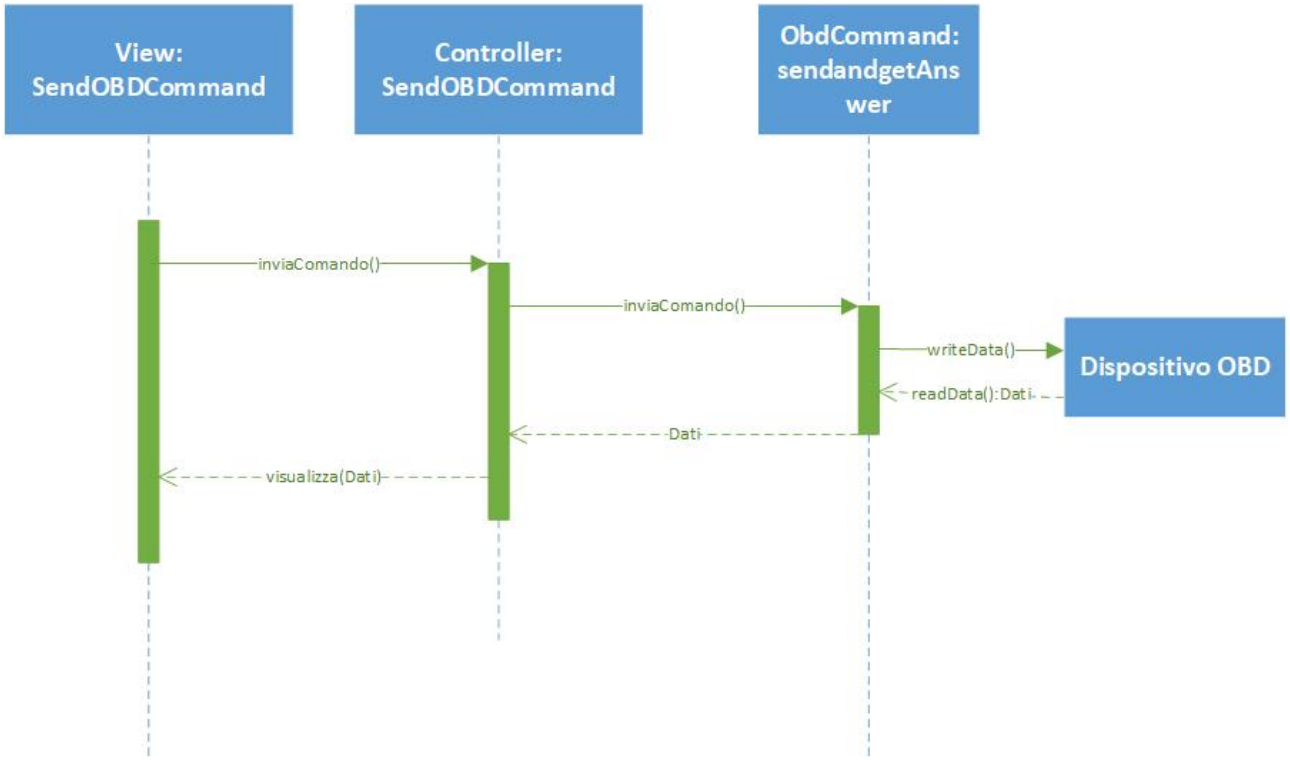
Il sistema permette la registrazione nella pagina principale; l'azione di registrazione, se avvenuta con esito positivo, mostrerà la schermata di autenticazione.

Diagramma di sequenza: autenticazione



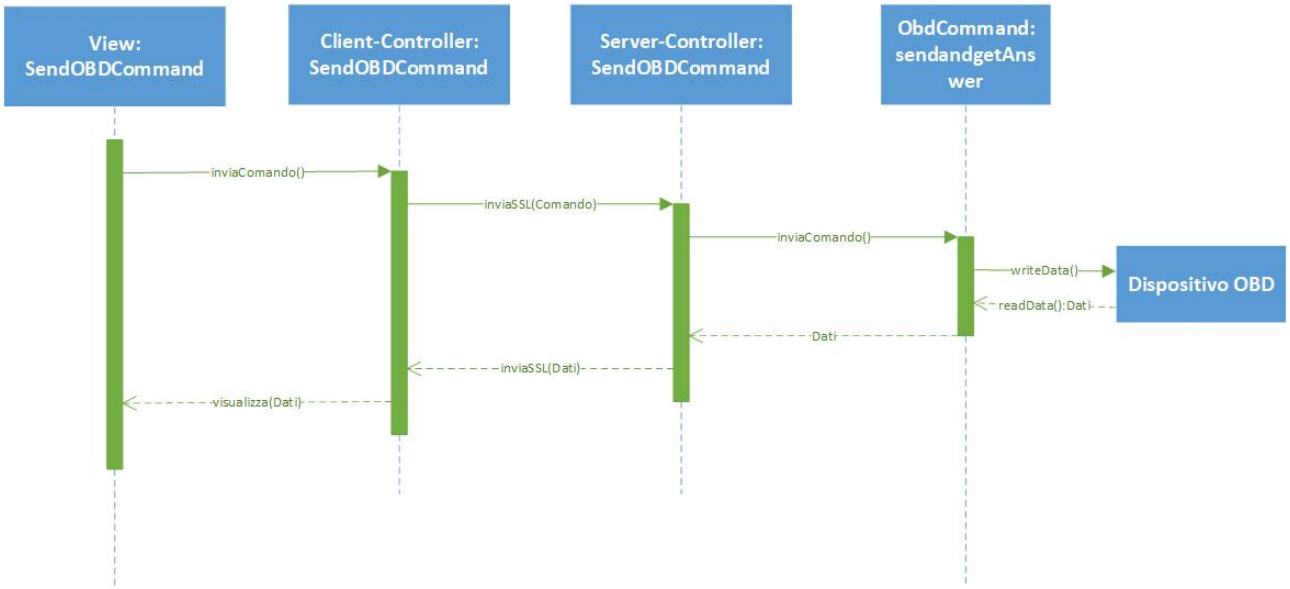
Il sistema permette l'autenticazione nella pagina principale dove è già possibile visualizzare l'interfaccia completa del software disabilitata; l'azione di autenticazione abilita le funzionalità dell'interfaccia; la funzione disconnetti disabilita le funzionalità non fruibili senza autenticazione.

Diagramma di sequenza: invio di un comando in locale



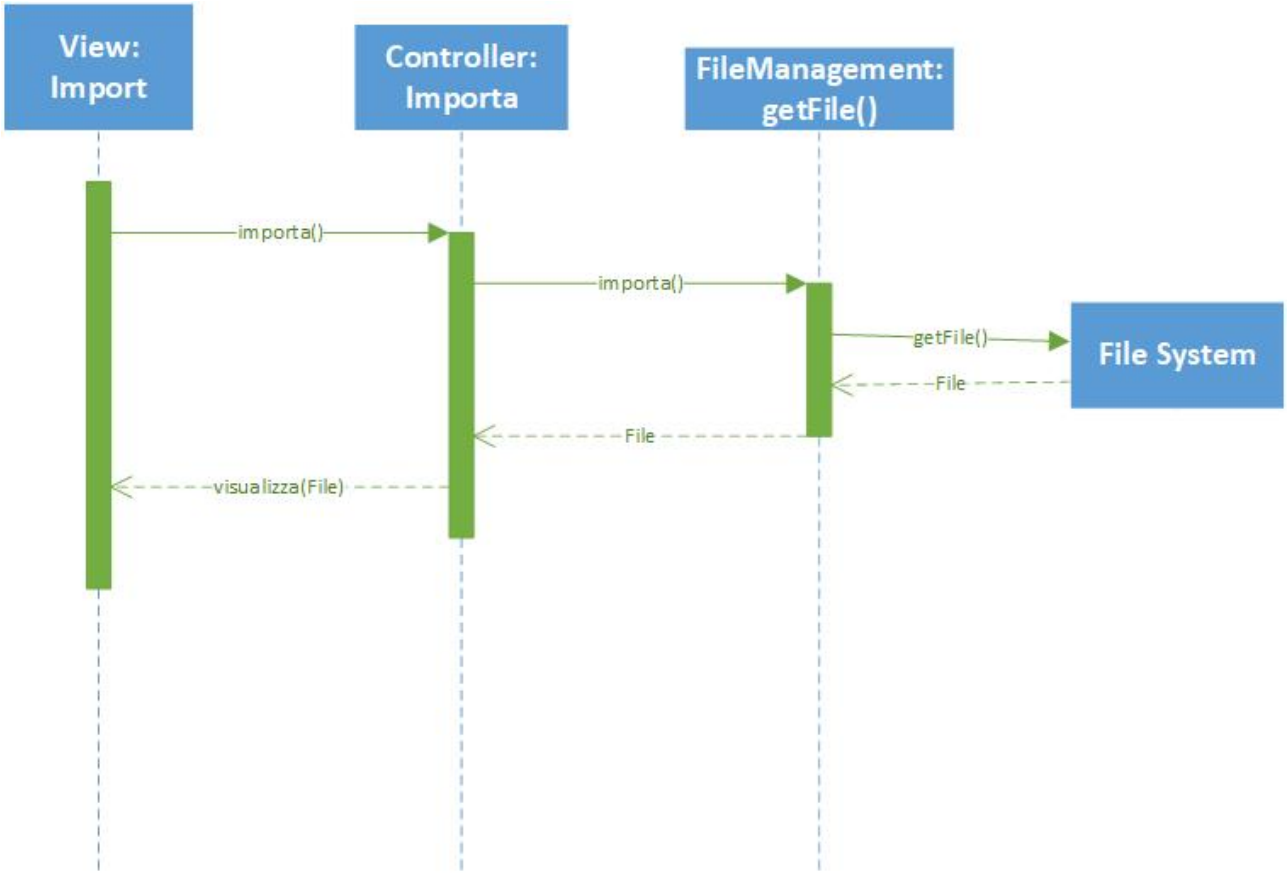
L'utente tramite la view seleziona un comando da inviare; il controller lo trasmette al gestore dell'interfaccia USB il quale scrive il comando sulla porta seriale COM e successivamente legge i dati in risposta che vengono elaborati e mostrati dal controller all'interno della view.

Diagramma di sequenza: invio di un comando in remoto



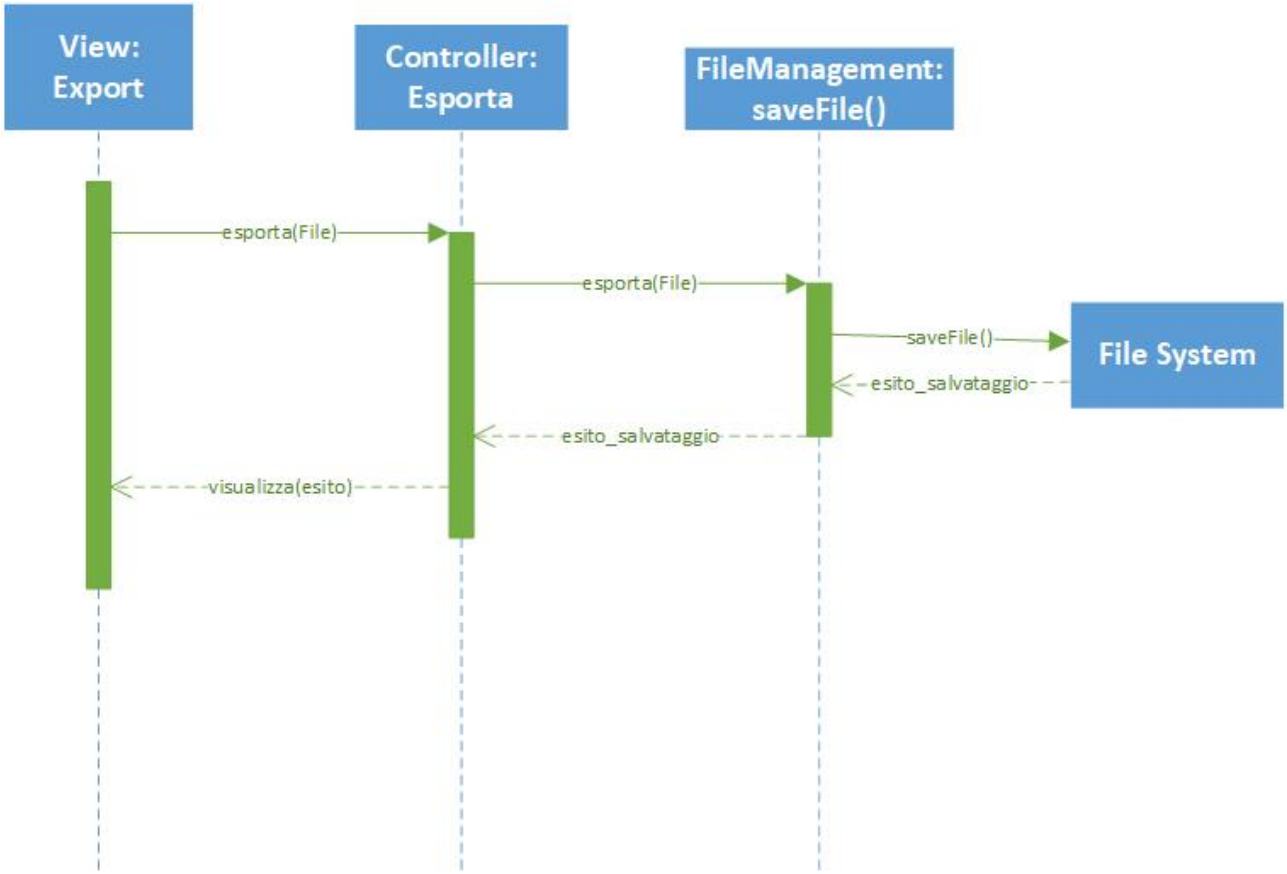
L'utente lato server seleziona tramite interfaccia un comando da inviare lato client, la comunicazione avviene tramite canale SSL e il controller lato server invia il comando tramite il gestore dell'interfaccia USB al dispositivo OBD.

Diagramma di sequenza: importazione di un log



Tramite apposita funzione getFile viene restituito il contenuto di un file presente nel File System.

Diagramma di sequenza: esportazione di un log



Tramite la funzione saveFile viene realizzato un file di log del nel File System.

Comportamento

Risultano sufficienti i diagrammi delle attività sviluppati in fase di analisi del problema, pertanto non ne sono stati aggiunti altri.

PROGETTAZIONE DELLA PERSISTENZA

Per la persistenza è stato scelto di utilizzare dei file di log in formato txt in cui salvare i dati acquisiti.

Esportazione di un file:

```
public boolean saveFile(String toSave, String owner)  {
		now = LocalDateTime.now();
		String nomeFile = owner+dtf.format(now)+".txt";
		BufferedWriter writer;
		try {
			writer = new BufferedWriter(new FileWriter(nomeFile,true));
			writer.write(toSave);
			writer.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return true;
	}
```

Importazione di un file:

```
public File getFile(String value) {
		File curDir = new File(".");
	File[] filesList = curDir.listFiles();
		for(File f : filesList){
			if(f.isFile() && f.getName().equals(value)){
				return f;
	  }
	}
		return new File("empty");
	  }
```

Formato file di log

Formato file di log per lo storico:

- DataOra Utente DatiInAnalisi

PROGETTAZIONE DEL COLLAUDO

Sono stati sviluppati i seguenti test per verificare il corretto funzionamento delle varie parti del sistema:

```
public class MyControllerTest {

    @Test
    void startTest() {
        ComPort cp = new ComPort();
        ArrayList<ObdCommand> list = new ArrayList<ObdCommand>();
        UserManagement UM = new UserManagement();
        FileManagement FM = new FileManagement();
        Controller C = new Controller(cp, list, UM, FM);
        assertTrue(C.getMyCommands().isEmpty());
    }

    @Test
    void myPortTest() {
        ComPort cp = new ComPort();
        ArrayList<ObdCommand> list = new ArrayList<ObdCommand>();
        UserManagement UM = new UserManagement();
        FileManagement FM = new FileManagement();
        Controller C = new Controller(cp, list, UM, FM);
        assertTrue(C.getMyPort().equals(cp));
    }
```

```
	@Test
	void getCommandsTest() {
		ComPort cp = new ComPort();
		ArrayList<ObdCommand> list = new ArrayList<ObdCommand>();
		UserManagement UM = new UserManagement();
		FileManagement FM = new FileManagement();
		Controller C = new Controller(cp, list, UM, FM);
		assertTrue(C.getMyCommands().equals(list));
	}
	@Test
	void getPortListTest() {
		ComPort cp = new ComPort();
		ArrayList<ObdCommand> list = new ArrayList<ObdCommand>();
		UserManagement UM = new UserManagement();
		FileManagement FM = new FileManagement();
		Controller C = new Controller(cp, list, UM, FM);
		assertTrue(C.getPortList().equals(new
ArrayList<String>(Arrays.asList(cp.getPortList()))));
	}
	@Test
	void getFileTest(){
		ComPort cp = new ComPort();
		ArrayList<ObdCommand> list = new ArrayList<ObdCommand>();
		UserManagement UM = new UserManagement();
		FileManagement FM = new FileManagement();
		Controller C = new Controller(cp, list, UM, FM);
		String nomeFile = "provaTest";
		assertTrue(C.getFile(nomeFile).equals(FM.getFile(nomeFile)));
	}
}
```

```
public class MyUsbInterfacingTest {

	@Test
	public void startTest() {
		ComPort myPort = new ComPort();
		//assertTrue(myPort.getSerialPort().equals(null));
		assertFalse(myPort.isPortFound());
	}

	@Test
	public void getPortTest() {
		ComPort myPort = new ComPort();
		assertFalse(myPort.findComPort("test"));

	}

	@Test
	public void getListTest() {
		ComPort myPort = new ComPort();
		String[] devs = myPort.getPortList();
		String [] internal = devs.clone();
		devs = myPort.getPortList();
		assertEquals(devs.length, internal.length);
	}
}
```

```
public class MyUserManagementTest {

	@Test
	void startTest() {
		UserManagement UM = new UserManagement();
		assertTrue(UM.getData().equals("users.txt"));
	}

	@Test
	void isValidTest() {
		UserManagement UM = new UserManagement();
		assertFalse(UM.isValid("prova", ""));
		assertFalse(UM.isValid("", "prova"));
		assertFalse(UM.isValid("", ""));
		assertFalse(UM.isValid("prova", "prova"));
	}

	@Test
	void writeFileTest() {
		String s = "user, password";
		UserManagement UM = new UserManagement();
		assertTrue(UM.writeFile(s));
	}
}
```

```
public class MyFileManagementTest {

	@Test
	void startTest() {
		FileManagement FM = new FileManagement();
		assertTrue(FM.getFile("prova").getName().equals("empty"));
	}

	@Test
	void mySearchFilesTest() {
		FileManagement FM = new FileManagement();
		FileManagement FM1 = new FileManagement();
		List<String> res = FM.searchFiles();
		List<String> res1 = FM1.searchFiles();
		assertTrue(res.equals(res1));
	}

	@Test
	void saveFileTest() {
		FileManagement FM = new FileManagement();
		String text = "provaprova";
		String owner = "programmer";
		assertTrue(FM.saveFile(text, owner));
	}
}
```

PROGETTAZIONE DEL DEPLOYMENT

Tutte le impostazioni e le funzioni del sistema sono accessibili attraverso il software OBDSight.

Avendo sviluppato il software come modello stand-alone, che pertanto è in grado di funzionare senza dipendenze esterne se non una Java Virtual Machine; si è deciso di fornire gli aggiornamenti del software, utili all'integrazione di nuovi comandi e nuove funzionalità in ambito di comunicazione client-server previsti per la versione 2.0, rilasciando il file jar eseguibile tramite servizio di hosting web.

Implementazione

Problematiche

- Per garantire una comunicazione client-server in tempo reale e trasparente è stato necessario eseguire le istanze di client e di server su dei thread separati. Abbiamo riscontrato problemi nel momento in cui il thread esterno (client o server) aveva necessità di modificare o mostrare contenuti all'interno del thread principale con la grafica realizzata in JavaFX. In seguito ad analisi e studio della manualistica Java abbiamo ovviato al problema integrando i thread come Task nel nostro applicativo.
- Si prevede di permettere la registrazione di una nuova utenza attraverso modulo web previsto per il rilascio nella futura versione 2.0 del software.
- Il rilevamento della porta COM, per poter avviare una comunicazione seriale con l'interfaccia OBD, è stato realizzando implementando la libreria open-source jssc alla classe ComPort; la quale offre metodi personalizzati per la lettura e la scrittura attraverso una porta COM selezionata.

Variazioni

In fase di sviluppo non è stato necessario implementare le classi enum descritte nella progettazione di dettaglio in quanto si è deciso di integrare le varie costanti necessarie all'interno di ogni singola classe rappresentante un comando.

L'autenticazione e la registrazione di un utente, da effettuare con mail all'interno di un servizio web, è stata temporaneamente implementata in locale e verrà resa disponibile in fase 2.

Collaudo

Il collaudo è stato effettuato sulle seguenti classi di test JUnit con esito positivo:

- MyCommandTest
- MyControllerTest
- MyFileManagementTest
- MyUsbInterfacingTest
- MyUserManagementTest
- MyClientServerTest

Nello specifico, nella classe di test MyCommandTest sono stati simulati l'istanziamento dei comandi, ognuno rappresentato da una specifica classe, ed i metodi ad essi associati.

La classe MyControllerTest permette di verificare il corretto istanziamento delle classi del model, come ad esempio la porta COM.

La classe MyFileManagementTest permette di verificare la corretta creazione e scrittura di file secondo le specifiche, oltre che la lettura in fase successiva.

La classe MyUsbInterfacingTest permette di verificare il corretto istanziamento dell'oggetto ComPort, l'avvenuta connessione alla porta COM selezionata e la comunicazione (read/write) con quest'ultima.

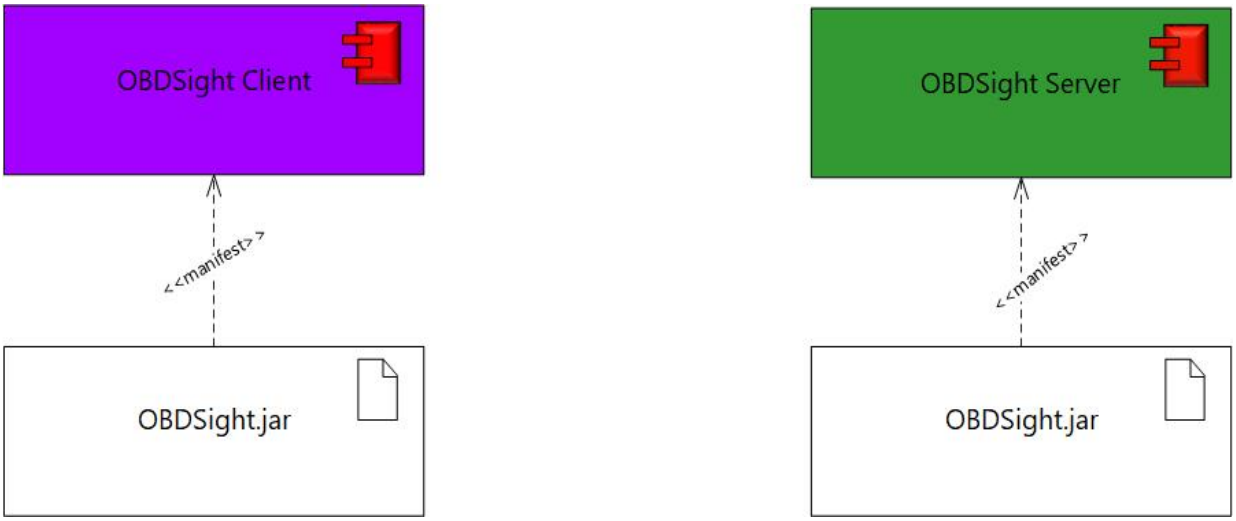
La classe MyUserManagementTest esegue controlli per garantire la corretta procedura di identificazione dell'utente e la verifica dei dati immessi.

La classe MyClientServerTest esegue le istanze di server e client, simulando una comunicazione attraverso un canale in rete locale e verificando la correttezza dei dati trasmessi.

Deployment

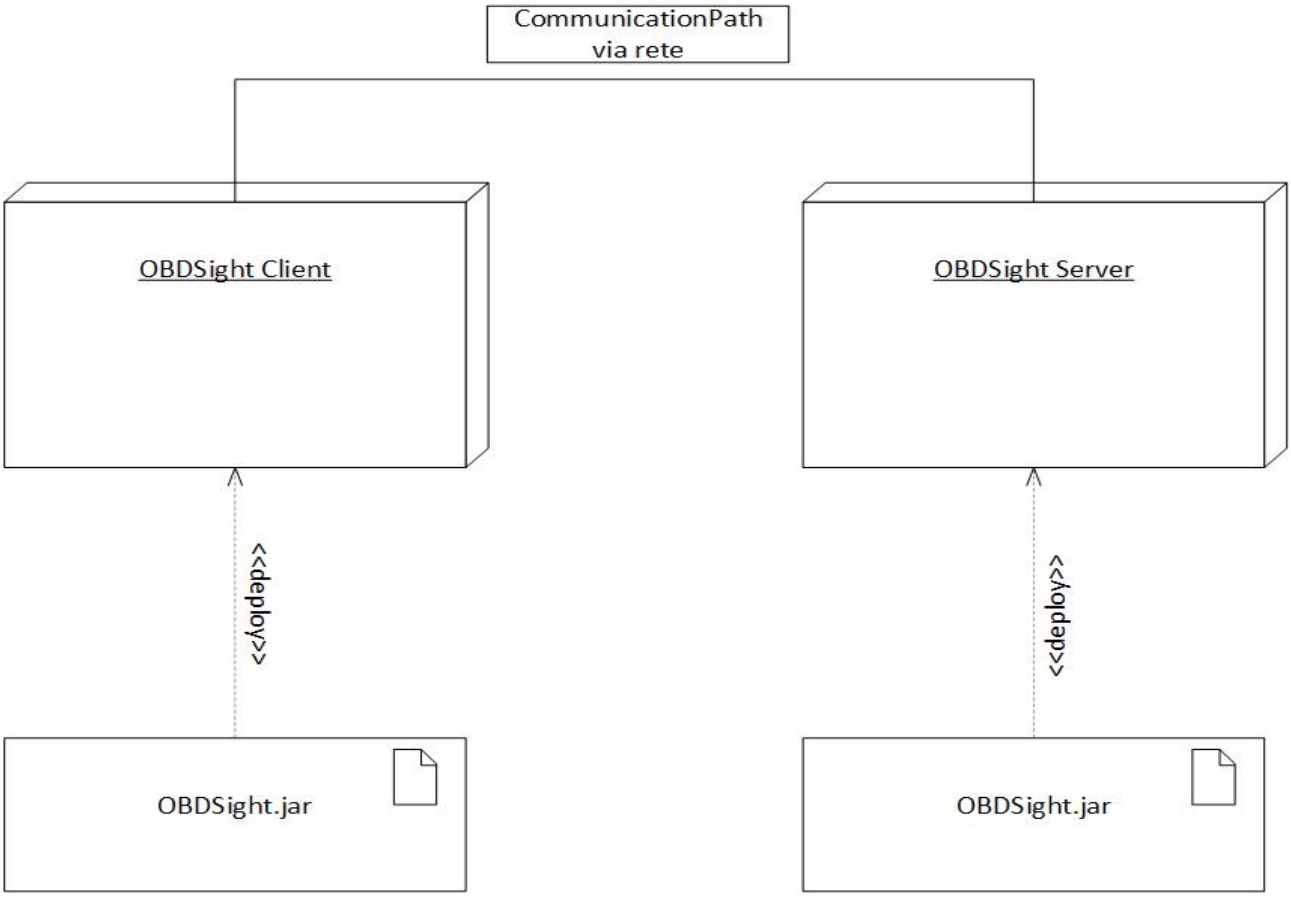
Artefatti

Il sistema OBDSight racchiude i componenti cliente e servitore generando un unico artefatto.



Deployment Type-Level

L'applicazione è auto-contenuta e pertanto è stato realizzato un unico applicativo per poter usufruire sia delle funzionalità server che delle funzionalità client.


CommunicationPath
via rete
OBDSight Client
OBDSight Server
<<deploy>>
<<deploy>>
OBDSight.jar
OBDSight.jar