Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Progettazione e Sviluppo di un Ambiente di Test Remoto per Protocollo UDS su CAN Bus.

Tesi di Laurea

Relatore

Prof. De Giovanni Luigi

Laureando

Brunello Marco

Matricola 2110997

Anno Accademico 2025-2026

"L'avversità produce perseveranza; la perseveranza una virtù provata"

—cf. Romani 5:3-4

Ringraziamenti

Desidero esprimere la mia gratitudine al professor De Giovanni Luigi, mio relatore, per l'aiuto e il sostegno che mi ha dato durante la stesura dell'elaborato.

Vorrei anche ringraziare, con affetto, i miei genitori per il loro sostegno, il grande aiuto e la loro presenza in ogni momento durante gli anni di studio.

Desidero poi ringraziare i miei amici per i bellissimi anni trascorsi insieme e le mille avventure vissute.

Padova, Luglio 2026 *Brunello Marco*

Sommario

La tesi descrive il lavoro svolto durante il periodo di stage presso l'azienda **Bluewind S.r.l.**, situata a Castelfranco Veneto (TV), nel periodo che intercorre dal 04/05/2026 al 26/06/2026 per un totale di 300 ore.
Lo scopo dello stage è stato quello di progettare e sviluppare un ambiente di testing remoto per verificare la sicurezza e la robustezza del sistema diagnostico UDSoCan interno dell'azienda.
Il lancio da remoto è stato gestito tramite una API REST esposta dal dispositivo Raspberry Pi, collegato all'hardware da testare e configurato per le risposte UDS; l'API viene successivamente chiamata da una pagina web (single page application) sviluppata con Svelte, framework Javascript, per visualizzare i risultati dei test e configurarne i parametri per poi inviare il comando di esecuzione.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda Bluewind S.r.l.

L'azienda Bluewind S.r.l. è un'azienda che fornisce soluzioni innovative a problemi in ambiti di elettronica, efficenza energetica e dispositivi distribuiti.

L'azienda, il cui logo è mostrato in Figura 1.1, fu fondata nel 1998 per servire industrie in tutta Europa e negli Stati Uniti, offrendo una vasta gamma di competenze in materia Automotive, Industriale e Medica.

Figura 1.1: Logo Bluewind S.r.l.

Al momento Bluewind S.r.l. conta più di 20 persone, comprese di staff e collaboratori, che provengono da ambiti diversi, creando così un pluralismo scientifico che dà all'azienda l'esperienza e la competenza che necessita per approcciare i clienti nei settori più disparati.

Per offrire servizi di qualità Bluewind S.r.l. collabora con aziende provenienti da tutto il mondo: Infineon, ST Microeletronics, NXP, Hightec Pls, Hitex, AWS e molte altre.

Tra i clienti che si affidano alla Bluewind per sviluppare le loro soluzioni ci sono: Illy, Osram, Euronda e molte altre [1].

1.2 Organizzazione del testo

Il resto della tesi è organizzato come segue:

Il secondo capitolo descrive il progetto di tirocinio e gli obiettivi prefissati.

Il terzo capitolo approfondisce invece i casi d'uso del prodotto con i relativi requisiti, suddivisi per importanza (obbligatori o non) e tipologia (funzionali, qualità e vincolo).

Il quarto capitolo approfondisce le attività di progettazione e codifica, presentando le architetture scelte ed i design pattern utilizzati.

Il quinto capitolo approfondisce le tecniche di verifica e validazione utilizzate nel progetto.

Il sesto capitolo raccoglie i risultati e mostra il consuntivo delle attività svolte e gli obiettivi raggiunti, con una riflessione personale accompagnata da una valutazione complessiva dell'esperienza.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *Nome del termine*$_G$;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Descrizione dello stage

2.1 Introduzione al progetto

L'introduzione di di nuove normative sulla sicurezza informatica per i prodotti connessi sta trasformando il modo in cui i sistemi *embedded*$_G$ vengono sviluppati e testati. Ai produttori, infatti, viene richiesto di fornire prove oggettive, strutturate e ripetibili che dimostrino l'implementazione e la validazione di adeguate misure di sicurezza informatica.

L'obiettivo del progetto è proprio quello di progettare e prototipare un ambiente di test di sicurezza informatica orientato ai dispositivi embedded impostati per rispondere a *frame*$_G$ costruiti secondo il protocollo UDSonCAN [3].

L'ambiente di test esegue su di un dispositivo Raspberry Pi; quest'ultimo espone un'*Application Program Interface (API)*$_G$ per permettere richieste di esecuzione dei test da remoto con parametri configurati dall'utente, oltre che ad una *Single Page Application (SPA)*$_G$, sviluppata usando Svelte [20], così da permettere una visualizzazione chiara e ordinata dei risultati dei test eseguiti.

L'argomento di studio principale del progetto di stage è quello di individuare un'architettura software, sia frontend sia backend, oltre che a consentire la corretta esecuzione dei test di sicurezza.

2.2 Obiettivi

In questa sezione vengono descritti gli obiettivi del progetto di stage, così come sono stati concordati l'azienda Bluewind S.r.l.

2.2.1 Notazione

La notazione scelta per la priorità degli obiettivi del tirocinio consiste:

- **O**: Obbligatori, in quanto obiettivo primario richiesto dall'azienda;
- **D**: Desiderabili, non vincolanti o strettamente necessari, ma danno un importante valore aggiunto al prodotto;
- **F**: Facoltativi, obiettivi il cui scopo è solo quello di dare valore aggiunto.

Le iniziali precedenti saranno seguite da un numero incrementale, relativo alla priorità, identificando univocamente l'obiettivo.

2.2.2 Obiettivi fissati

Gli obiettivi dello stage sono riportati nella Tabella 2.1.

Tabella 2.1: Tabella obiettivi stage

Codice	Descrizione
O1	Scelta delle tecnologie, setup dell'ambiente di sviluppo e definizione dei requisiti minimi
O2	*Setup* del canale di comunicazione (CAN) ed esecuzione manuale di test
O3	Definizione di una struttura per descrivere i test e implementazione di un test runner
O4	Sviluppo di API per l'esecuzione di test da remoto
D1	*Refactoring*$_G$ e modularizzazione del sistema di test

Continua nella prossima pagina...

Tabella 2.1 – Continuo della tabella

Codice	Descrizione
D2	Refactoring test esistenti
D3	Documentazione e sviluppo di un'applicazione dimostrativa
D4	Concatenazione di test multipli in un *workflow*$_G$, con possibilità di utilizzare il riusltato di un test come *input* per uno successivo
F1	*Proof of Concept (PoC)*$_G$ dell'estendibilità del sistema tramite setup di un altro canale di comunicazione (es.UART)
F2	Sviluppo di uno storico che permetta all'utente di visualizzare i test eseguiti in precedenza, insieme alle relative configurazioni, così da poterli rieseguire facilmente

Facciamo notare che, rispetto agli obiettivi riportati nel piano di lavoro, sono state concordate, all'inizio dello stage, alcune piccole variazioni. In particolare, sono stati aggiunti l'obiettivo desiderabile D4 l'obiettivo facoltativo F2.

2.3 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi a cui si potrà andare incontro. Si è quindi proceduto a elaborare delle possibili soluzioni per far fronte ai rischi di seguito elencati.

1. Inesperienza

Descrizione: Lavorando in un ambiente nuovo, quale l'embedded, è facile commettere errori dovuti alla mancanza di conoscenza ed esperienza.

Soluzione: Studio personale e supporto da parte del tutor aziendale.

2. Errori di valutazione temporale

Descrizione: È possibile la sottostima o sovrastima della durata di alcuni compiti.

Soluzione: Pianificazione anticipata di macrocompiti da portare a termine nelle varie settimane per tutta la durata del tirocinio.

2.4 Pianificazione

Come da piano di lavoro, il progetto è stato previsto durare 8 settimane, per un totale di 300 ore. La distribuzione delle ore è mostrata nella Tabella 2.2.

Tabella 2.2: Tabella che illustra le attività previste nel piano di lavoro.

Settimana	Attività	Ore pianificate
Prima settimana	Scelta delle tecnologie, setup dell'ambiente di sviluppo e definizione dei requisiti minimi	40
Seconda settimana	Setup del canale di comunicazione (CAN) ed esecuzione manuale di test	40
Terza settimana	Definizione di una struttura per descrivere i test e implementazione di un test runner	40
Quarta settimana	Sviluppo di API per l'esecuzione di test da remoto	40
Quinta settimana	Refactoring test esistenti	40
Sesta settimana	Refactoring e modularizzazione del sistema di test	40
Settima settimana	PoC dell'estendibilità del sistema tramite setup di un altro canale di comunicazione (es. UART)	40
Ottava settimana	Documentazione e sviluppo di un'applicazione dimostrativa	20

2.5 Tecnologie utilizzate

In questa sezione vengono descritte le tecnologie impiegate nel progetto, la motivazione della scelta e il loro utilizzo.

2.5.1 CAN

CAN *bus*$_G$ è un sistema di comunicazione, usato solitamente in veicoli, che permette a diverse *Electronic Control Units (ECU)*$_G$ di comunicare tra di loro senza un computer *host* [7].

Fisicamente parlando, tutte le ECU sono collegate da un bus composto da una coppia di cavi intrecciati: CAN-high e CAN-low, come si può osservare in Figura 2.1.

Figura 2.1: Coppia di cavi CAN-high e CAN-low [2]

In data odierna, CAN rappresenta lo standard di comunicazione per ambienti in ambito automotive ed è largamente usata in applicazioni industriali embedded, dove viene richiesto un alto livello di immunità ai disturbi del segnale.

CAN, infatti, è stato espressamente progettato per funzionare in ambienti ove le interferenze, dovute alle onde elettromagnetiche, rendono difficile la comunicazione utilizzando dispositivi di uso quotidiano.

Frame CAN

CAN è composto da una componente fisica (livello 1 *stack ISO-OSI*$_G$) e da una componente data-link (livello 2 stack ISO-OSI), per il progetto tuttavia la parte rilevante è stata quella data-link.

Un frame CAN standard è composto da diversi campi, tra cui un identificativo di 11 bit, vedi Figura 2.2. La versione estesa dell'identificativo lungo 29 bit viene usata per veicoli pesanti.

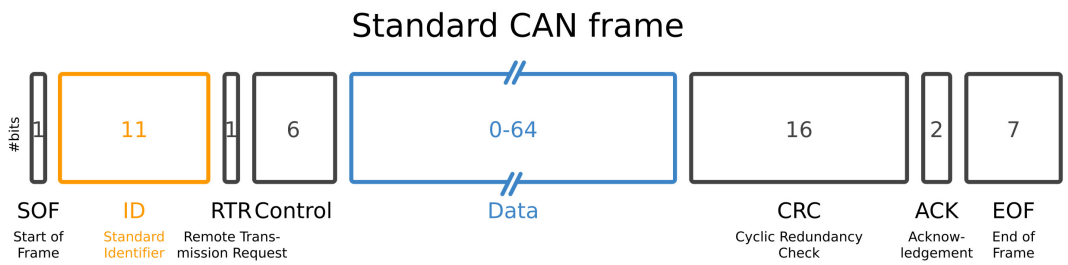

Figura 2.2: Frame Can standard [2]

I campi del frame CAN sono illustrati nella Figura 2.2, che ne riporta la lunghezza in numero di bit. I campi sono i seguenti [2]:

- **SOF**: bit di inizio frame, viene posto a 0 per indicare agli altri nodi che si intende comunicare;

- **ID**: identificativo del frame, più il valore è basso, più il frame ha priorità alta;

- **RTR**: richiesta di trasmissione remota, indica se il mittente sta inviando dati oppure li sta richiedendo da un altro nodo;

- **Controllo**: i bit di controllo contengono il *Bit aggiuntivo di identificazione (IDE)*$_G$ che deve essere 0 per i frame CAN standard, un bit riservato a distinguere i frame standard da quelli estesi e 4 bit per indicare la lunghezza dei dati che verranno trasmessi (0-8 bytes);

- **Dati**: contiene i byte di dati significativi che devono essere trasmessi al nodo ricevente;

- **CRC**: controllo di parità a ridondanza;

- **ACK**: indica se il nodo ha ricevuto e riconosciuto i dati correttamente, il primo bit ha valore 1 se è il mittente ad inviare il messaggio, 0 se

è il ricevente, mentre il secondo rappresenta un delimitatore ed ha valore 1;

- **EOF**: indica la fine di un frame.

Con questa struttura del frame esistono quattro tipi di varianti di CAN frame [2]:

- **Data frame**: illustrato sopra, trasporta dati da un nodo mittente ad un nodo ricevente;

- **Error frame**: usato per comunicare l'individuazione di un errore. Questo frame contiene delimitatori di errore e delle *flag* di errore. Quest'ultime si suddividono in *flag* di errore attivo, trasmessi da un nodo che ha rilevato un errore e si trova nello stato di *error active*, e di errore passivo, trasmessi da un nodo che ha rilevato la presenza sulla rete di una flag di errore attiva, passando allo stato *error passive*;
- **Remote frame**: questo frame può essere usato per richiedere certi dati da un nodo CAN ed è molto simile al data frame. Manca però il campo "Dati" e il campo "RTR" è posto ad 1;

- **Overload frame**: può essere usato per fornire un ritardo maggiore tra altri frame CAN, questo se qualche nodo CAN richiede del tempo aggiuntivo per il calcolo, in pratica però non sono usati quasi mai.

2.5.2 ISO-TP

ISO-TP è uno standard internazionale per inviare pacchetti di dati attraverso il CAN bus [10]. Viene usato quando si ha la necessità di inviare dati che superano gli 8 byte, limite massimo di dati trasportabili da un frame CAN standard.

ISO-TP aggiunge metadati all'inizio della sezione "Dati" di un frame CAN, riducendo di fatto la quantità di dati che un singolo frame può trasportare, i metadati si chiamano *Protocol Control Information (PCI)*$_G$. Il campo iniziale è composto da 4 bit e indica il tipo di frame, descrivendo implicitamente anche la lunghezza del PCI.

ISO-TP definisce 4 frame [23]:

- **Frame singolo (FS)**: codice PCI 0, il singolo frame che viene inviato contiene un massimo di 7 byte per il payload (campo "Dati");
- **Primo frame (PF)**: codice PCI 1, indica il primo frame di un pacchetto segmentato in più frame, questo quando si vogliono comunicare un numero di byte superiore a 7. Il primo frame contiene la lunghezza dell'intero pacchetto segmentato e i dati iniziali di quest'ultimo;
- **Frame consecutivo (FC)**: codice PCI 2, un frame che contiene i dati successivi al PF;
- **Frame di controllo di flusso (CF)**: codice PCI 3, questo frame viene inviato in risposta alla ricezione del PF. Viene usato per gestire la cadenza di invio dei frame consecutivi.

Dunque, come riportato nella Tabella 2.3, il primo byte contiene il PCI, mentre i successivi 7 byte contengono i dati significativi.

Tabella 2.3: ISO-TP Header [23]

	Byte 0		Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
	Nibble alto	Nibble basso	Byte Pieno (0x00-0x-FF)	Byte Pieno (0x00-0x-FF)	"	"	"	"	"
Singolo	0x0	Lungezza (0x1...0x7)	Dato A	Dato B	Dato C	...	...	...	...
Primo	0x1	Lunghezza Messaggio (0x008...0xFFF)		Dato A	Dato B	...	...	...	...
Consecutivo	0x2	Indice (0x0...0xF)	Dato A	Dato B	Dato C	...	...	...	...
Controllo Flusso	0x3	FC Flag (0x0, 0x1, 0x2)	Grandezza blocco	STmin	...	...	...	...	...

Come mostrato dalla Tabella 2.3, il frame di controllo di flusso contiene alcuni campi aggiuntivi che permettono al CF di comunicare al mittente se e cosa inviare una volta inviato il PF. I campi aggiuntivi sono:

- **Nibble basso**: si intendono i bit meno significativi del byte, nonostante ciò, indicano al mittente se continuare a inviare dati (0x0), aspettare un determinato lasso di tempo (0x1) oppure smettere di inviare a causa di un errore di tipo buffer overflow, oppure la da parte del ricevente di interrompere la trasmissione;

- **Grandezza blocco**: se il valore di questo campo è 0, allora il ricevente sta segnalando al mittente di inviare i frame rimanenti senza che quest'ultimo aspetti un altro frame di controllo oppure un *delay*;

- **STmin**: si intende il tempo di separazione minimo che intercorre un tra un frame e il suo successivo.

2.5.3 UDS

Unified Diagnostic Services (UDS)$_G$ è un protocollo di comunicazione usato per la diagnostica, aggiornamenti *firmware*, testing *runtime* e molto altro [3].

Nella pratica, la comunicazione UDS avviene tramite un'architettura di tipo client-server, dove il client è il dispositivo che effettua i test mentre il server è il dispositivo ECU. A differenza di CAN e ISO-TP, che stanno rispettivamente ai livelli 2 e 3 dello stack ISO-OSI, UDS si piazza nel livello 7 dello stack, in quanto può essere implementato da vari protocolli di comunicazione.

Dato che il contesto dello stage prevede l'utilizzo di CAN al livello 2 e di ISO-TP come protocollo di trasporto, si procederà a descrivere UDSonCAN [9].

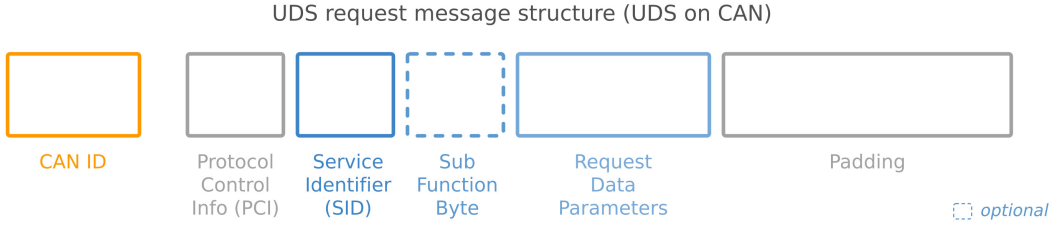

Figura 2.3: Frame UDSonCAN [3]

Come si può vedere dalla Figura 2.3, il PCI, di ISO-TP, viene mantenuto.

Il **SID**, o identificatore di servizio, è il campo dove inserire il codice del servizio UDS di cui si vuole usufruire. Ad esempio, il servizio 0x10 controlla quali servizi UDS, oltre al 0x10, sono disponibili in una ECU. I SID si dividono in due categorie: richiesta e risposta. Questo perché i SID risposta aggiungono un 0x40 al SID che viene richiesto. Ad esempio, se l'utente invia il SID 0x10, la risposta avrà come SID: 0x50 [8].

I byte di sottofunzione (opzionali) sono usati in alcuni frame di richiesta UDS, infatti non tutti i servizi prevedono delle sottofunzioni, per visualizzare i SID e le relative sottofunzioni si rimanda a ISO-14229.

Infine ci sono i parametri della richiesta, o *Identificatori Di Dati (DID)*$_G$. Nella maggior parte dei servizi UDS di richiesta ci sono diversi tipi parametri, usati per fornire ulteriori informazioni alla richiesta oltre al SID e alle sottofunzioni. Gli identificatori di dati possono essere proprietari e, in quanto tali, solo i costruttori delle ECU ne conoscono i dettagli, nonstante ciò ce ne sono alcuni di standardizzati. Ad esempio 0xF190 con SID 0x22 richiede il numero identificativo del veicolo.

Come detto in precedenza, in caso di risposta postiva, il ricevente risponderà al SID di richiesta con lo stesso SID più 0x40, seguito dai dati effettivamente richiesti.

In alcuni casi però, una ECU deve inviare una risposta negativa, ad esempio un servizio non è supportato. Una risposta negativa è costruita come segue:

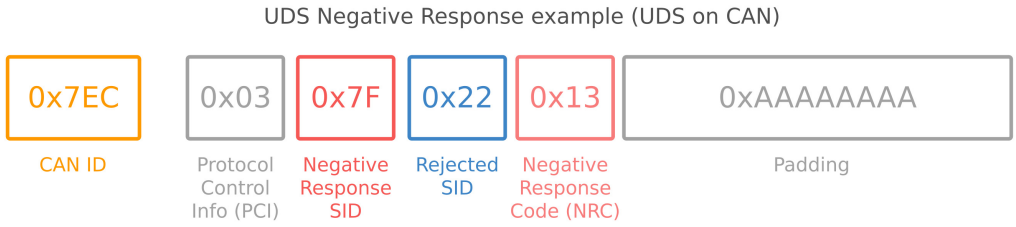

Figura 2.4: Esempio di risposta negativa UDS per SID 0x22 [3]

Come si può evincere dalla Figura 2.4:

- **1 Byte**: il primo byte è il PCI;

- **2 Byte**: 0x7F è il SID riservato all'errore;

- **3 Byte**: è il SID della richiesta che ha generato l'errore;
- **4 Byte**: è il codice di risposta negativa (NRC), è molto simile al campo DID, solo che invece che identificare i dati, fornisce maggiori dettagli sulla causa dell'errore.

2.5.4 Linguaggi e framework di sviluppo

Per lo sviluppo del progetto sono stati utilizzati diversi linguaggi di programmazione e framework. In questa sezione vengono descritte le loro caratteristiche e la motivazione che ne ha portato alla scelta.

Python

Python [16] è stato utilizzato come linguaggio principale per lo sviluppo del backend del progetto. La scelta di questa tecnologia non è stata effettuata direttamente in fase progettuale, ma è derivata dall'ambiente di sviluppo già adottato dall'azienda ospitante: i test di sicurezza preesistenti, con cui il progetto doveva integrarsi, erano infatti già implementati in Python. Per garantire compatibilità, continuità e riutilizzo del codice esistente, il linguaggio è stato quindi mantenuto anche per lo sviluppo dell'API e dei nuovi strumenti di testing remoto. In particolare, Python è stato impiegato per la gestione delle comunicazioni relative ai protocolli CAN e UDSonCAN.

La disponibilità di librerie dedicate e la natura modulare del linguaggio hanno inoltre facilitato lo sviluppo e l'integrazione delle diverse componenti software del sistema.

Svelte

Svelte [20] è un framework JavaScript/TypeScript per la realizzazione di interfacce utente dinamiche, basato su un approccio di compilazione che genera

codice ottimizzato in fase di build, riducendo il carico di esecuzione lato browser. Proprio per queste caratteristiche, Svelte è stato utilizzato per lo sviluppo dell'interfaccia grafica dell'applicativo. Poiché l'ambiente di esecuzione è destinato a un dispositivo Raspberry Pi, la ridotta complessità del codice generato e il basso impatto prestazionale rappresentano un vantaggio significativo, consentendo di ottimizzare l'uso delle risorse hardware disponibili e di riservare maggiore capacità di calcolo all'esecuzione dei test.

SQLite

SQLite [19] è un sistema di gestione di database relazionali leggero e integrato, che non richiede un server separato per funzionare.

In questo progetto, SQLite è stato utilizzato per memorizzare i risultati dei test eseguiti, le configurazioni associate e lo storico delle esecuzioni. Il motivo principale per cui SQLite è stato scelto è la sua semplicità di integrazione e la sua leggerezza, che lo rendono ideale per ambienti embedded.

Vite

Vite [21] è uno strumento di build che rappresenta lo standard per lo sviluppo di applicazioni frontend che adottano framework Javascript/Typescript. Nel contesto del progetto, Vite è stato utilizzato in combinazione con Svelte, essendo lo strumento di build raccomandato per la gestione della compilazione e del testing di questo framework.

Pytest

Pytest [14] è un framework di test python che rappresenta lo standard per i test di unità e di integrazione.

Nel progetto, pytest è stato impiegato per la scrittura dei test di unità facendo largo uso delle `fixture` ed è stato scelto proprio perché rappresenta lo standard per la verifica del codice scritto in python.

Vitest

Vitest [22] è il framework di testing integrato con Vite. Poiché il progetto si basa sull'ecosistema Svelte, l'adozione di Vitest è risultata la scelta ottimale per la scrittura dei test di unità, consentendo di sfruttare la stessa pipeline di configurazione del codice sorgente.

2.6 Dispositivi hardware

In questa sezione vengono descritti i dispositivi hardware forniti da Bluewind S.r.l. per lo sviluppo del progetto.

2.6.1 Raspberry Pi 4

Le schede Raspberry Pi [17] sono delle sehede programmabili a basso costo, con un'architettura ARM, che offrono una vasta gamma di funzionalità e connettività. Il Raspberry Pi 4, utilizzato nel progetto, è dotato di un processore quad-core a 64 bit, fino a 8 GB di RAM, porte USB 3.0.

La scheda Raspberry Pi è stata utilizzata come piattaforma principale per l'esecuzione dei test, esponendo l'API per consentire il lancio da remoto dei test e la visualizzazione dei risultati.

2.6.2 Scheda Infosystem

La scheda Infosystem è una scheda di sviluppo progettata per facilitare la comunicazione e l'interfacciamento con dispositivi che utilizzano il protocollo CAN. Questa scheda è dotata di un microcontrollore che gestisce le comunicazioni CAN, consentendo di inviare e ricevere messaggi attraverso il bus CAN.

Nel contesto del progetto, la scheda Infosystem è stata utilizzata come target ECU per i test di sicurezza che veninvano eseguiti nel Raspberry Pi.

Capitolo 3

Analisi dei requisiti

Lo stage mira allo sviluppo di un sistema che, tramite un'interfaccia web, consente all'utente di selezionare, configurare ed eseguire un test di sicurezza su una ECU collegata fisicamente ad un Raspberry Pi remoto. Una volta terminata l'esecuzione del test, il sistema ne restituisce il risultato che verrà visualizzato dall'utente tramite la stessa l'interfaccia web utilizzata per l'invio della richiesta.

In questo capitolo vengono descritti i casi d'uso individuati, i requisiti funzionali e non funzionali del sistema. L'analisi dei requisiti è stata condotta attraverso uno studio dei casi d'uso e delle funzionalità richieste dal sistema. Sia i casi d'uso che i requisiti sono stati approvati dal tutor aziendale.

3.1 Casi d'uso

I casi d'uso sono descrizioni di scenari che rappresentano le interazioni tra gli attori e il sistema. Nel progetto è stato individuato un attore principale: l'utente che attraverso l'interfaccia web può interagire con il sistema e cambiarne lo stato. Per questo motivo i casi d'uso sono stati descritti in relazione a questo singolo attore.

Ogni caso d'uso è identificato in modo univoco tramite la convenzione UC_X, dove X è un numero progressivo. Per ogni caso d'uso sono stati definiti i seguenti campi:

- **Attori Principali**: gli attori che interagiscono con il sistema;

- **Descrizione**: una breve descrizione del caso d'uso;
- **Precondizioni**: le condizioni che devono essere soddisfatte perché il caso d'uso venga soddisfatto;
- **Postcondizioni**: le condizioni che il caso d'uso rispetterà una volta soddisfatto;
- **Inclusioni** (opzionale): riferimenti ai casi d'uso il cui comportamento è inserito obbligatoriamente dentro al caso d'uso principale;
- **Estensioni** (opzionale): riferimenti ai casi d'uso che costituiscono uno scenario alternativo rispetto al caso d'uso principale;
- **Generalizzazioni** (opzionale): riferimenti ai casi d'uso che specializzano il caso d'uso principale.

Di seguito, vengono riportati tutti i casi d'uso individuati.

UC_1: Visualizzazione test implementati

Attori Principali: Utente.
Descrizione: L'utente viene a conoscenza dei test implementati e può visualizzarli.
Precondizioni: Il sistema è connesso e configurato correttamente.
Postcondizioni: L'utente viene a conoscenza dei test eseguibili.
Inclusioni: UC_1.1

UC_1.1: Visualizzazione lista dei test implementati

Attori Principali: Utente.
Descrizione: L'utente visualizza la lista dei test implementati, dunque eseguibili.
Precondizioni: Il sistema è connesso e configurato correttamente.
Postcondizioni: L'utente visualizza la lista dei test eseguibili.
Inclusioni: UC_1.1.1

UC_1.1.1: Visualizza singolo test

Attori Principali: Utente.

Descrizione: L'utente visualizza il singolo test da eseguire.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il singolo test.

Inclusioni: UC_1.1.1.1, UC_1.1.1.2, UC_1.1.1.3, UC_1.1.1.4, UC_1.1.1.5, UC_1.1.1.6, UC_1.1.1.7, UC_1.1.1.8, UC_1.1.1.9, UC_1.1.1.10, UC_1.1.1.11, UC_1.1.1.12, UC_1.1.1.13, UC_1.1.1.14

UC_1.1.1.1: Visualizza nome

Attori Principali: Utente.

Descrizione: L'utente visualizza il nome del test.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il nome del test.

UC_1.1.1.2: Visualizza campo "id"

Attori Principali: Utente.

Descrizione: L'utente visualizza campo "id", ovvero l'id del messaggio CAN.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il campo "id".

UC_1.1.1.3: Visualizza campo "dati"

Attori Principali: Utente.

Descrizione: L'utente visualizza campo "dati", ovvero i dati che verranno inviati.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il campo "dati".

UC_1.1.1.4: Visualizza campo "*delay*"

Attori Principali: Utente.

Descrizione: L'utente visualizza campo "*delay*", ovvero il lasso di tempo trascorso tra i messaggi CAN.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il campo "*delay*".

UC_1.1.1.5: Visualizza campo "durata"

Attori Principali: Utente.

Descrizione: L'utente visualizza campo "durata", ovvero la durata massima di esecuzione del test.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il campo "durata".

UC_1.1.1.6: Visualizza campo "esteso"

Attori Principali: Utente.

Descrizione: L'utente visualizza campo "esteso", ovvero un booleano che indica se usare l'id nella forma estesa.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il campo "esteso".

UC_1.1.1.7: Visualizza campo "intervallo_id"

Attori Principali: Utente.

Descrizione: L'utente visualizza campo "intervallo_id", ovvero l'intervallo di ID dei messaggi CAN per cui sarà effettuata una richiesta.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il campo "intervallo_id".

UC_1.1.1.8: Visualizza campo "intervallo_servizi"

Attori Principali: Utente.

Descrizione: L'utente visualizza campo "intervallo_servizi", per ogni servizio compreso in questo intervallo sarà effettuata una richiesta UDS.
Precondizioni: Il sistema è connesso e configurato correttamente per il test.
Postcondizioni: L'untente visualizza il campo "intervallo_servizi".

UC_1.1.1.9: Visualizza campo "*timeout*"

Attori Principali: Utente.
Descrizione: L'utente visualizza campo "*timeout*", ovvero il tempo massimo di attesa per una risposta UDS.
Precondizioni: Il sistema è connesso e configurato correttamente per il test.
Postcondizioni: L'untente visualizza il campo "*timeout*".

UC_1.1.1.10: Visualizza campo "intervallo_did"

Attori Principali: Utente.
Descrizione: L'utente visualizza campo "intervallo_did", per ogni DID compreso in questo intervallo sarà effettuata una richiesta UDS.
Precondizioni: Il sistema è connesso e configurato correttamente per il test.
Postcondizioni: L'untente visualizza il campo "intervallo_did".

UC_1.1.1.11: Visualizza campo "cambia_sessione"

Attori Principali: Utente.
Descrizione: L'utente visualizza campo "cambia_sessione", ovvero un booleano che indica se cambiare la sessione UDS.
Precondizioni: Il sistema è connesso e configurato correttamente per il test.
Postcondizioni: L'untente visualizza il campo "cambia_sessione".

UC_1.1.1.12: Visualizza campo "sessioni_supportate"

Attori Principali: Utente.
Descrizione: L'utente visualizza campo "sessioni_supportate", ovvero un elen-

co delle sessioni UDS supportate.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il campo "sessioni_supportate".

UC_1.1.1.13: Visualizza campo "livelli_supportati"

Attori Principali: Utente.

Descrizione: L'utente visualizza campo "livelli_supportati", ovvero un elenco dei livelli di sicurezza UDS supportati.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il campo "livelli_supportati".

UC_1.1.1.14: Visualizza inserimento file di configurazione .yml

Attori Principali: Utente.

Descrizione: L'Utente visualizza il campo di inserimento del file di configurazione .yml.

Precondizioni: Il sistema è connesso e configurato correttamente per il test.

Postcondizioni: L'untente visualizza il campo di inserimento del file di configurazione .yml.

UC_2: Configurazione test

Attori Principali: Utente.

Descrizione: L'utente configura il test da eseguire.

Precondizioni: L'utente visualizza i parametri del test.

Postcondizioni: Il test è pronto per essere eseguito.

UC_3: Configurazione manuale dei parametri

Attori Principali: Utente.

Descrizione: L'utente inserisce il valore nei campi dei parametri indicati dal test.

Precondizioni: L'utente visualizza i campi dei parametri del test.

Postcondizioni: Il test è pronto per essere eseguito.

Inclusioni: UC_3.1, UC_3.2, UC_3.3, UC_3.4, UC_3.5, UC_3.6, UC_3.7, UC_3.8, UC_3.9, UC_3.10, UC_3.11, UC_3.12

Generalizzazione: UC_2

UC_3.1: Configura parametro "id"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "id".

Precondizioni: L'utente visualizza il campo "id".

Postcondizioni: L'utente ha configurato il campo "id".

UC_3.2: Configura parametro "dati"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "dati", ovvero i dati che verranno inviati.

Precondizioni: L'utente visualizza il campo "dati".

Postcondizioni: L'utente ha configurato il campo "dati".

UC_3.3: Configura parametro "*delay*"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "*delay*", ovvero il lasso di tempo trascorso tra una richiesta e l'altra.

Precondizioni: L'utente visualizza il campo "*delay*".

Postcondizioni: L'utente ha configurato il campo "*delay*".

UC_3.4: Configura parametro "durata"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "durata", ovvero la durata massima

di esecuzione del test.

Precondizioni: L'utente visualizza il campo "durata".

Postcondizioni: L'utente ha configurato il campo "durata".

UC_3.5: Configura parametro "esteso"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "esteso", ovvero un booleano se l'id nella forma estesa.

Precondizioni: L'utente visualizza il campo "esteso".

Postcondizioni: L'utente ha configurato il campo "esteso".

UC_3.6: Configura parametro "intervallo_id"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "intervallo_id", per ogni di in questo intervallo verrà effettuata una richiesta.

Precondizioni: L'utente visualizza il campo "intervallo_id".

Postcondizioni: L'utente ha configurato il campo "intervallo_id".

Estensioni: UC_3.6.1

UC_3.6.1: Errore configurazione parametro "intervallo_id"

Attori Principali: Utente.

Descrizione: L'utente inserisce un intervallo non valido, ovvero l'estremo inferiore dell'intervallo è maggiore di quello superiore.

Precondizioni: L'utente ha inserito un intervallo non valido nel campo "intervallo_id".

Postcondizioni: L'utente visualizza l'errore "Intervallo id non valido".

UC_3.7: Configura parametro "intervallo_servizi"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "intervallo_servizi", per ogni servizio in questo intervallo verrà inviata una richiesta.
Precondizioni: L'utente visualizza il campo "intervallo_servizi".
Postcondizioni: L'utente ha configurato il campo "intervallo_servizi".
Estensioni: UC_3.7.1

UC_3.7.1: Errore configurazione parametro "intervallo_servizi"

Attori Principali: Utente.
Descrizione: L'utente inserisce un intervallo non valido, ovvero l'estremo inferiore dell'intervallo è maggiore di quello superiore.
Precondizioni: L'utente ha inserito un intervallo non valido nel campo "intervallo_id".
Postcondizioni: L'utente visualizza l'errore "Intervallo servizi non valido".

UC_3.8: Configura campo "*timeout*"

Attori Principali: Utente.
Descrizione: L'utente configura il campo "*timeout*", ovvero il tempo massimo di attesa per una risposta ECU.
Precondizioni: L'utente visualizza il campo "*timeout*".
Postcondizioni: L'utente ha configurato il campo "*timeout*".

UC_3.9: Configura campo "intervallo_did"

Attori Principali: Utente.
Descrizione: L'utente configura il campo "intervallo_did", per ogni *data identifier* in questo intervallo verrà inviata una richiesta.
Precondizioni: L'utente visualizza il campo "intervallo_did".
Postcondizioni: L'utente ha configurato il campo "intervallo_did".
Estensioni: UC_3.9.1

UC_3.9.1: Errore configurazione parametro "intervallo_did"

Attori Principali: Utente.

Descrizione: L'utente inserisce un intervallo non valido, ovvero l'estremo inferiore dell'intervallo è maggiore di quello superiore.

Precondizioni: L'utente ha inserito un intervallo non valido nel campo "intervallo_did".

Postcondizioni: L'utente visualizza l'errore "Intervallo *data identifier* non valido".

UC_3.10: Configura campo "cambia_sessione"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "cambia_sessione", ovvero un booleano che indica se cambiare la sessione.

Precondizioni: L'utente visualizza il campo "cambia_sessione".

Postcondizioni: L'utente ha configurato il campo "cambia_sessione".

UC_3.11: Configura campo "sessioni_supportate"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "sessioni_supportate", ovvero un elenco delle sessioni UDS supportate.

Precondizioni: L'utente visualizza il campo "sessioni_supportate".

Postcondizioni: L'utente ha configurato il campo "sessioni_supportate".

UC_3.12: Configura campo "livelli_supportati"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "livelli_supportati", ovvero un elenco dei livelli di sicurezza UDS supportati.

Precondizioni: L'utente visualizza il campo "livelli_supportati".

Postcondizioni: L'untente ha configurato il campo "livelli_supportati".

UC_4: Configurazione parametri via file di configurazione

Attori Principali: Utente.

Descrizione: L'utente fornisce al sistema un file di configurazione .yml contenente tutti i parametri del test con i relativi valori.

Precondizioni: L'utente visualizza il campo di caricamento del file.

Postcondizioni: Il test è pronto per essere eseguito.

Generalizzazione: UC_2

UC_5: Esecuzione test

Attori Principali: Utente.

Descrizione: L'utente invia al sistema il test da eseguire, con i parametri configurati.

Precondizioni: L'utente ha impostato i parametri correttamente.

Postcondizioni: Il test viene eseguito.

UC_6: Visualizzazione risultato

Attori Principali: Utente.

Descrizione: L'utente visualizza il risultato del test che ha eseguito.

Precondizioni: L'utente ha eseguito un test e quest'ultimo ha terminato la sua esecuzione.

Postcondizioni: L'utente visualizza il risultato del test.

Inclusioni: UC_6.1

UC_6.1: Visualizzazione stato del test

Attori Principali: Utente.

Descrizione: L'utene visualizza lo stato del test.

Precondizioni: L'utente ha eseguito un test e quest'ultimo ha terminato la sua esecuzione.

Postcondizioni: L'utente visualizza lo stato del test.

UC_7: Visualizzazione impostazioni di sessione

Attori Principali: Utente.

Descrizione: L'utente visualizza una finestra di dialogo dove può configurare le impostazioni di Can bus e ISO-TP.

Precondizioni: L'utente ha eseguito un test e quest'ultimo ha terminato la sua esecuzione.

Postcondizioni: L'utente visualizza una finestra di dialogo con le impostazioni di sessione.

Inclusioni: UC_7.1

UC_7.1: Visualizzazione configurazione Can bus

Attori Principali: Utente.

Descrizione: L'utente visualizza i campi di configurazione del Can bus.

Precondizioni: Il sistema è connesso e configurato correttamente.

Postcondizioni: L'utente visualizza la configurazione del Can bus.

Inclusioni: UC_7.1.1, UC_7.1.2, UC_7.1.3, UC_7.1.4

UC_7.1.1: Visualizzazione campo "canale"

Attori Principali: Utente.

Descrizione: L'utente visualizza il campo "canale" del bus, che indica la porta da impiegare.

Precondizioni: Il sistema è connesso e correttamente configurato.

Postcondizioni: L'utente visualizza il campo "canale.

UC_7.1.2: Visualizzazione campo "interfaccia"

Attori Principali: Utente.

Descrizione: Utente visualizza il campo "interfaccia" del bus, indica l'interfaccia di accesso del bus all'hardware.

Precondizioni: Il sistema è connesso e correttamente configurato.

Postcondizioni: L'utente visualizza il campo "interfaccia".

UC_7.1.3: Visualizzazione campo "*bitrate*"

Attori Principali: Utente.

Descrizione: L'utente visualizza il campo "*bitrate*", definisce la velocità di scambio dei dati all'interno del bus.

Precondizioni: Il sistema è connesso e correttamente configurato.

Postcondizioni: L'utente visualizza il campo "*bitrate*".

UC_7.1.4: Visualizzazione inserimento file configurazione .yml

Attori Principali: Utente.

Descrizione: L'utente visualizza il campo di inserimento del file di configurazione .yml per le configurazioni del Can bus.

Precondizioni: Il sistema è connesso e correttamente configurato.

Postcondizioni: L'utente visualizza il campo di inserimento del file.

UC_7.2: Visualizzazione configurazione ISO-TP

Attori Principali: Utente.

Descrizione: L'utente visualizza i campi di configurazione ISO-TP.

Precondizioni: Il sistema è connesso e correttamente configurato.

Postcondizioni: L'utente visualizza la configurazione ISO-TP.

Inclusioni: UC_7.2.1, UC_7.2.2, UC_7.2.3

UC_7.2.1: Visualizzazione campo "id_richiesta"

Attori Principali: Utente.

Descrizione: L'utente visualizza il campo "id_richiesta", ovvero l'id con cui verrà inviato il frame Can.

Precondizioni: Il sistema è connesso e correttamente configurato.

Postcondizioni: L'utente visualizza il campo "id_richiesta".

UC_7.2.2: Visualizzazione campo "id_risposta"

Attori Principali: Utente.

Descrizione: L'utente visualizza il campo "id_risposta", ovvero l'id del frame di risposta.

Precondizioni: Il sistema è connesso e correttamente configurato.

Postcondizioni: L'utente visualizza il campo "id_risposta".

UC_7.2.3: Visualizzazione campo "esteso"

Attori Principali: Utente.

Descrizione: L'utente visualizza il campo "esteso", ovvero un booleano che indica se il frame utilizza un id esteso a 29 bit.

Precondizioni: Il sistema è connesso e correttamente configurato.

Postcondizioni: L'utente visualizza il campo "esteso".

UC_7.2.4: Visualizzazione inserimento file configurazione .yml

Attori Principali: Utente.

Descrizione: L'utente visualizza il campo di inserimento del file di configurazione .yml per le configurazioni del frame ISO-TP.

Precondizioni: Il sistema è connesso e correttamente configurato.

Postcondizioni: L'utente visualizza il campo di inserimento del file.

UC_8: Configurazione impostazioni di sessione

Attori Principali: Utente.

Descrizione: L'utente inserisce i valori dei parametri di configurazione Can bus e ISO-TP.

Precondizioni: L'utente visualizza i campi di configurazione della sessione.

Postcondizioni: Can bus e ISO-TP correttamente configurati e pronti per essere utilizzati.

Inclusioni: UC_8.1, UC_8.2, UC_8.3, UC_8.4

UC_8.1: Configurazione Can bus

Attori Principali: Utente.

Descrizione: L'utente inserisce i valori di configurazione del Can bus.

Precondizioni: L'utente visualizza i campi di configurazione del Can bus.

Postcondizioni: Il bus è pronto per essere utilizzato.

Inclusioni: UC_8.1.1, UC_8.1.2, UC_8.1.3

UC_8.1.1: Configurazione campo "canale"

Attori Principali: Utente.

Descrizione: L'utente inserisce il valore della porta.

Precondizioni: L'utente visualizza il campo "canale".

Postcondizioni: L'utente ha configurato il campo "canale".

UC_8.1.2: Configurazione campo "interfaccia"

Attori Principali: Utente.

Descrizione: L'utente inserisce il valore dell'interfaccia bus.

Precondizioni: L'utente visualizza il campo "interfaccia".

Postcondizioni: L'utente ha configurato il campo "interfaccia".

UC_8.1.3: Configurazione campo "*bitrate*"

Attori Principali: Utente.

Descrizione: L'utente inserisce il valore del *bitrate* desiderato del bus.

Precondizioni: L'utente visualizza il campo "*bitrate*".

Postcondizioni: L'utente ha configurato il campo "*bitrate*".

UC_8.2: Configurazione Can bus via file .yml

Attori Principali: Utente.

Descrizione: L'utente inserisce il file di configurazione .yml con i parametri del Can bus.

Precondizioni: L'utente visualizza i campi di configurazione del Can bus.

Postcondizioni: Il bus è pronto per essere utilizzato.

UC_8.3: Configurazione ISO-TP

Attori Principali: Utente.

Descrizione: L'utente inserisce i valori di configurazione di ISO-TP.

Precondizioni: L'utente visualizza i campi di configurazione ISO-TP.

Postcondizioni: Il sistema è pronto per eseguire test con i parametri di configurazione dell'ISO-TP.

Inclusioni: UC_8.3.1, UC_8.3.2, UC_8.3.3

UC_8.3.1: Configurazione campo "id_richiesta"

Attori Principali: Utente.

Descrizione: L'utente compila il campo "id_richiesta".

Precondizioni: L'utente visualizza il campo "id_richiesta".

Postcondizioni: L'utente ha configurato il campo "id_richiesta".

UC_8.3.2: Configurazione campo "id_risposta"

Attori Principali: Utente.

Descrizione: L'utente compila il campo "id_risposta".

Precondizioni: L'utente visualizza il campo "id_risposta".

Postcondizioni: L'utente ha configurato il campo "id_risposta".

UC_8.3.3: Configurazione campo "esteso"

Attori Principali: Utente.

Descrizione: L'utente compila il campo "esteso".

Precondizioni: L'utente visualizza il campo "esteso".

Postcondizioni: L'utente ha configurato il campo "esteso".

UC_8.4: Configurazione ISO-TP via file .yml

Attori Principali: Utente.

Descrizione: L'utente inserisce il file di configurazione .yml con i parametri di configurazione dell'ISO-TP.

Precondizioni: L'utente visualizza i campi di configurazione dell'ISO-TP.

Postcondizioni: Il sistema è pronto per eseguire test con i parametri di configurazione dell'ISO-TP.

UC_9: Esportazione risultato test in file .yml

Attori Principali: Utente.

Descrizione: L'utente esporta il risultato del test con la sua configurazione in un file .yml.

Precondizioni: Il test è stato eseguito e il risultato è visualizzato.

Postcondizioni: File .yml completo di risultato e configurazione del test.

UC_10: Esportazione risultato test in file json

Attori Principali: Utente.

Descrizione: L'utente esporta il risultato del test con la sua configurazione in un file .json.

Precondizioni: Il test è stato eseguito e il risultato è visualizzato.

Postcondizioni: File .json completo di risultato e configurazione del test.

UC_11: Creazione workflow

Attori Principali: Utente.

Descrizione: L'utente crea un workflow di test in modo da eseguire più test in sequenza.

Precondizioni: Test presenti nel sistema.

Postcondizioni: Workflow pronto per essere eseguito.

Inclusioni: UC_11.1, UC_11.2, UC_11.3

UC_11.1: Selezione test

Attori Principali: Utente.

Descrizione: L'utente seleziona il test da inserire nel workflow.

Precondizioni: Test presenti nel sistema.

Postcondizioni: Test selezionato.

UC_11.2: Configurazione test

Attori Principali: Utente.

Descrizione: L'utente configura il test selezionato.

Precondizioni: Test selezionato.

Postcondizioni: Test configurato e pronto per essere eseguito nel workflow.

Inclusioni: UC_11.2.1, UC_11.2.2, UC_11.2.3, UC_11.2.4, UC_11.2.5, UC_11.2.6, UC_11.2.7, UC_11.2.8, UC_11.2.9, UC_11.2.10, UC_11.2.11, UC_11.2.12, UC_11.2.13

UC_11.2.1: Configura parametro "id"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "id", ovvero l'id del frame Can.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "id".

UC_11.2.2: Configura parametro "dati"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "dati", ovvero i dati che verranno inviati.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "dati".

UC_11.2.3: Configura parametro "*delay*"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "*delay*", overo il lasso di tempo trascorso tra una richiesta e l'altra.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "*delay*".

UC_11.2.4: Configura parametro "durata"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "durata", ovvero la durata massima di esecuzione del test.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "durata".

UC_11.2.5: Configura parametro "esteso"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "esteso", ovvero un booleano che indica se usare l'id nella forma estesa.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "esteso".

UC_11.2.6: Configura parametro "intervallo_id"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "intervallo_id", per ogni id in questo intervallo verrà effettuata una richiesta.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "intervallo_id".

Estensioni: UC_11.2.6.1

UC_11.2.6.1: Errore configurazione parametro "intervallo_id"

Attori Principali: Utente.

Descrizione: L'utente inserisce un intervallo non valido, ovvero l'estremo inferiore dell'intervallo è maggiore di quello superiore.

Precondizioni: L'utente ha inserito un intervallo non valido nel campo "intervallo_id".

Postcondizioni: L'utente visualizza l'errore "Intervallo id non valido".

UC_11.2.7: Configura parametro "intervallo_servizi"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "intervallo_servizi", per ogni servizio in questo intervallo verrà effettuata una richiesta.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "intervallo_servizi".

Estensioni: UC_11.2.7.1

UC_11.2.7.1: Errore configurazione parametro "intervallo_servizi"

Attori Principali: Utente.

Descrizione: L'utente inserisce un intervallo non valido, ovvero l'estremo inferiore dell'intervallo è maggiore di quello superiore.

Precondizioni: L'utente ha inserito un intervallo non valido nel campo "intervallo_servizi".

Postcondizioni: L'utente visualizza l'errore "Intervallo servizi non valido".

UC_11.2.8: Configura campo "*timeout*"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "*timeout*", ovvero il tempo massimo di attesa per la risposta di una richiesta.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "*timeout*".

UC_11.2.9: configura campo "intervallo_did"

Attori Principali: utente.

Descrizione: l'utente configura il campo "intervallo_did", per ogni did in questo intervallo verrà effettuata una richiesta.

Precondizioni: sistema connesso e configurato correttamente per il test.

Postcondizioni: l'utente ha configurato il campo "intervallo_did".

Estensioni: UC_11.2.9.1

UC_11.2.9.1: Errore configurazione parametro "intervallo_did"

Attori Principali: Utente.

Descrizione: L'utente inserisce un intervallo non valido, ovvero l'estremo inferiore dell'intervallo è maggiore di quello superiore.

Precondizioni: L'utente ha inserito un intervallo non valido nel campo "intervallo_did".

Postcondizioni: L'utente visualizza l'errore "Intervallo did non valido".

UC_11.2.10: Configura campo "cambia_sessione"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "cambia_sessione", ovvero un booleano che indica se cambiare la sessione.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "cambia_sessione".

UC_11.2.11: Configura campo "sessioni_supportate"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "sessioni_supportate", ovvero un elenco di sessioni UDS supportate.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "sessioni_supportate".

UC_11.2.12: Configura campo "livelli_supportati"

Attori Principali: Utente.

Descrizione: L'utente configura il campo "livelli_supportati", ovvero un elenco di livelli di sicurezza supportati.

Precondizioni: Sistema connesso e configurato correttamente per il test.

Postcondizioni: L'utente ha configurato il campo "livelli_supportati".

UC_11.2.13: Utilizzo contesto

Attori Principali: Utente.

Descrizione: L'utente seleziona, tramite una *checkbox*, se utilizzare o meno output di altri test come input di configurazione per parametri del test che stesso.

Precondizioni: Test presenti nel sistema e almeno due test presenti e configurati nel workflow.

Postcondizioni: L'utente ha selezionato se utilizzare o meno il contesto.

UC_11.3: Aggiunta test al workflow

Attori Principali: Utente.

Descrizione: L'utente, una volta configurato il test selezionato, lo aggiunge al workflow.

Precondizioni: Test configurato e pronto per essere aggiunto al workflow.

Postcondizioni: Aggiunto test al workflow.

UC_12: Esecuzione workflow

Attori Principali: Utente.

Descrizione: L'utente, una volta creato il workflow, lo esegue.

Precondizioni: Workflow pronto per essere eseguito.

Postcondizioni: Tutti i test all'interno del workflow vengono eseguiti in sequenza.

UC_13: Visualizzazione risultato workflow

Attori Principali: Utente.

Descrizione: L'utente visualizza i risultati dei test eseguiti all'interno del workflow.

Precondizioni: Workflow ha terminato la sua esecuzione.

Postcondizioni: L'utente visualizza i risultati dei test eseguiti all'interno del workflow.

Inclusioni: UC_13.1

UC_13.1: Visualizza risultato di ogni singolo test del workflow

Attori Principali: Utente.

Descrizione: L'utente visualizza i risultati di ogni singolo test nel workflow.

Precondizioni: Workflow eseguito.

Postcondizioni: Tutti i test del workflow eseguiti.

Inclusioni: UC_13.1.1

UC_13.1.1: Visualizza stato del test

Attori Principali: Utente.

Descrizione: L'utente visualizza lo stato del risultato del test.

Precondizioni: Test del workflow eseguito.

Postcondizioni: L'utente visualizza lo stato del test.

UC_14: Visualizzazione storico

Attori Principali: Utente.

Descrizione: L'utente visualizza lo storico di tutti i test e workflow che ha eseguito.

Precondizioni: Il sistema è connesso.

Postcondizioni: Lista di test e workflow eseguiti.

Inclusioni: UC_14.1, UC_14.2, UC_14.3

UC_14.1: Visualizzazione singolo elemento storico

Attori Principali: Utente.

Descrizione: L'uente visualizza un singolo elemento necessario nello storico.

Precondizioni: È stato eseguito almeno un test o workflow

Postcondizioni: L'utente visualizza il singolo elemento dello storico

Inclusioni: UC_14.1.1

UC_14.1.1: Visualizzazione id univoco dell'esecuzione del test

Attori Principali: Utente.

Descrizione: L'utente visualizza un id univoco usato per identificare il test o workflow.

Precondizioni: Sistema connesso ed esiste un test o workflow nello storico.

Postcondizioni: L'utente visualizza l'id univoco del test o workflow.

UC_14.2: Visualizzazione storico test

Attori Principali: Utente.

Descrizione: L'utente visualizza un test nello storico.

Precondizioni: È stato esetuito almeno un test dall'utente.

Postcondizioni: L'utente visualizza il singolo test.

Inclusioni: UC_14.2.1

Generalizzazione: UC_14.1

UC_14.2.1: Visualizzazione nome del test

Attori Principali: Utente.

Descrizione: L'utente visualizza il nome del test nello storico.

Precondizioni: È stato eseguito almeno un test dall'utente.

Postcondizioni: L'utente visualizza il nome del test.

UC_14.3: Visualizzazione storico workflow

Attori Principali: Utente.

Descrizione: L'utente visualizza un workflow nello storico.

Precondizioni: È stato eseguito almeno un workflow dall'utente.

Postcondizioni: L'utente visualizza il singolo workflow.

Inclusioni: UC_14.3.1

Generalizzazione: UC_14.1

UC_14.3.1: Visualizzazione numero test workflow

Attori Principali: Utente.

Descrizione: L'utente visualizza il numero di test presenti all'interno del workflow.

Precondizioni: È stato eseguito almeno un workflow dall'utente.

Postcondizioni: L'utente visualizza il numero di test presenti all'interno del workflow.

UC_15: Visualizzazione dettaglio test nello storico

Attori Principali: Utente.

Descrizione: L'utente visualizza in dettaglio il test che è stato eseguito.

Precondizioni: Il test è stato selezionato.

Postcondizioni: L'utente visualizza i dettagli dei test.

Inclusioni: UC_15.1, UC_15.2, UC_15.3, UC_15.4, UC_15.5, UC_15.6

UC_15.1: Visualizzazione configurazione del test

Attori Principali: Utente.

Descrizione: L'utente visualizza i parametri di configurazione del test

Precondizioni: Test visualizzato in dettaglio.

Postcondizioni: L'utente visualizza la configurazione del test.

UC_15.2: Visualizzazione risultati test

Attori Principali: Utente.

Descrizione: L'utente visualizza il risultato dell'esecuzione del test.

Precondizioni: Test visualizzato in dettaglio.

Postcondizioni: L'utente visualizza il risultato prodotto dall'esecuzione del test.

Inclusioni: UC_15.2.1

UC_15.2.1: Visualizzazione stato test

Attori Principali: Utente.

Descrizione: L'utente visualizza lo stato del risultato del test.

Precondizioni: Test visualizzato in dettaglio.

Postcondizioni: L'utente visualizza lo stato del test.

UC_15.3: Visualizzazione esportazione risultati test

Attori Principali: Utente.

Descrizione: L'utente visualizza il pulsante per esportare i risultati del test in un file .yml.

Precondizioni: Test visualizzato in dettaglio.

Postcondizioni: File .yml completo delle informazioni del test.

UC_15.4: Esportazione risultati test

Attori Principali: Utente.

Descrizione: L'utente clicca il pulsante per esportare i risultati del in un file .yml.

Precondizioni: L'utente visualizza il pulsante per esportare i risultati del test in .yml.

Postcondizioni: Risultati del test correttamente inseriti in un file .yml.

UC_15.5: Visualizzazione rilancio del test

Attori Principali: Utente.

Descrizione: L'utente visualizza il pulsante per eseguire di nuovo il test con le configurazioni mostrate in dettaglio.

Precondizioni: Test visualizzato in dettaglio.

Postcondizioni: L'utente visualizza il pulsante per eseguire di nuovo il test.

UC_15.6: Rilancio del test

Attori Principali: Utente.

Descrizione: L'utente clicca il pulsante di rilancio del test in modo da eseguire nuovamente il test con la configurazione visualizzata.

Precondizioni: Test visualizzato in dettaglio.

Postcondizioni: Test eseguito nuovamente con la configurazione mostrata.

UC_16: Visualizzazione dettaglio workflow

Attori Principali: Utente.

Descrizione: L'utente visualizza in dettaglio il workflow

Precondizioni: È stato eseguito almeno un workflow.

Postcondizioni: L'utente visualizza il workflow nel dettaglio.

Inclusioni: UC_16.1, UC_16.2, UC_16.3, UC_16.4, UC_16.5, UC_16.6, UC_16.7, UC_16.8, UC_16.9

UC_16.1: Visualizzazione esportazione workflow

Attori Principali: Utente.

Descrizione: L'utente visualizza il pulsante per esportare in .yml i dettagli del workflow.

Precondizioni: Workflow visualizzato in dettaglio.

Postcondizioni: Visualizzazione del pulsante per esportare in .yml i dettagli del workflow.

UC_16.2: Esportazione workflow

Attori Principali: Utente.

Descrizione: L'utente esporta in un file .yml i dettagli del workflow.

Precondizioni: L'utente clicca il pulsante per esportare in .yml i dettagli del workflow.

Postcondizioni: File .yml correttamente riempito dei dettagli del workflow.

UC_16.3: Visualizzazione rilancio del workflow

Attori Principali: Utente.

Descrizione: L'utente visualizza il pulsante per eseguire di nuovo tutti i test del workflow.

Precondizioni: Workflow visualizzato in dettaglio.

Postcondizioni: L'utente visualizza il pulsante di rilancio del workflow.

UC_16.4: Rilancio intero workflow

Attori Principali: Utente.

Descrizione: L'utente esegue di nuovo tutti i test presenti nel workflow.

Precondizioni: Workflow visualizzato in dettaglio.

Postcondizioni: Tutti i test del workflow vengono eseguiti nuovamente.

UC_16.5: Visualizzazione singolo test del workflow

Attori Principali: Utente.

Descrizione: L'utente visualizza il singolo test del workflow in dettaglio.

Precondizioni: Workflow visualizzato in dettaglio.

Postcondizioni: L'utente visualizza il singolo test del workflow in dettaglio.

UC_16.6: Visualizzazione configurazione del test del workflow

Attori Principali: Utente.

Descrizione: L'utente visualizza i dettagli di configurazione dei singoli test presenti nel workflow.

Precondizioni: Workflow visualizzato in dettaglio.

Postcondizioni: L'utente visualizza le configurazioni dei test nel workflow.

UC_16.7: Visualizzazione rilancio singolo test del workflow

Attori Principali: Utente.

Descrizione: L'utente visualizza il pulsante per eseguire di nuovo il test con la configurazione visualizzata.

Precondizioni: L'utente visualizza il singolo test del workflow in dettaglio.

Postcondizioni: L'utente visualizza il il pulsante per il rilancio del singolo test del workflow.

UC_16.8: Rilancio singolo test del workflow

Attori Principali: Utente.

Descrizione: L'utente esegue nuovamente il singolo test del workflow con la configurazione presente.

Precondizioni: Workflow visualizzato in dettaglio.

Postcondizioni: L'utente esegue di nuovo il singolo test del workflow.

UC_16.9: Visualizzazione risultati

Attori Principali: Utente.

Descrizione: L'utente visualizza i risultati dei singoli test del workflow.

Precondizioni: Workflow visualizzato in dettaglio.

Postcondizioni: L'utente visualizza i risultati dei singoli test del workflow.

Inclusioni: UC_16.9.1

UC_16.9.1: Visualizzazione stato

Attori Principali: Utente.

Descrizione: L'utente visualizza lo stato dei singoli test del workflow.

Precondizioni: L'utente visualizza il risultato dei test.

Postcondizioni: L'utente visualizza lo stato dei test.

3.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.

Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.

Il codice dei requisiti, dove ogni requisito è identificato con il carattere **R**, è così strutturato:

F: Funzionale.

Q: Qualitativo.

V: Di vincolo.

O: Obbligatorio (necessario).

D: Desiderabile.

OP: Opzionale.

Nelle Tabelle 3.1, 3.2 e 3.3 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali.

Requisito	Descrizione	Use Case
RFO_1	L'utente deve poter visualizzare la lista dei test eseguibili	UC_1, UC_1.1, UC_1.1.1
RFO_2	L'utente deve poter visualizzare il nome di ogni singolo test	UC_1.1.1.1
Continua nella prossima pagina...		

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFO_3	L'utente deve poter visualizzare il parametro "id" qualora fosse necessario	UC_1.1.1.2
RFO_4	L'utente deve poter visualizzare il parametro "dati" qualora fosse necessario	UC_1.1.1.3
RFO_5	L'utente deve poter visualizzare il parametro "*delay*" qualora fosse necessario	UC_1.1.1.4
RFO_6	L'utente deve poter visualizzare il parametro "durata" qualora fosse necessario	UC_1.1.1.5
RFO_7	L'utente deve poter visualizzare il parametro "esteso" qualora fosse necessario	UC_1.1.1.6
RFO_8	L'utente deve poter visualizzare il parametro "intervallo_id" qualora fosse necessario	UC_1.1.1.7
RFO_9	L'utente deve poter visualizzare il parametro "intervallo_servizi" qualora fosse necessario	UC_1.1.1.8
RFO_10	L'utente deve poter visualizzare il parametro "*timeout*" qualora fosse necessario	UC_1.1.1.9
RFO_11	L'utente deve poter visualizzare il parametro "intervallo_did" qualora fosse necessario	UC_1.1.1.10
RFO_12	L'utente deve poter visualizzare il parametro "cambia_sessione" qualora fosse necessario	UC_1.1.1.11
RFO_13	L'utente deve poter visualizzare il parametro "sessioni_supportate" qualora fosse necessario	UC_1.1.1.12
RFO_14	L'utente deve poter visualizzare il parametro "livelli_supportati" qualora fosse necessario	UC_1.1.1.13

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFO_15	L'utente deve poter configurare il parametro "id" qualora fosse necessario	UC_2, UC_3, UC_3.1
RFO_16	L'utente deve poter configurare il parametro "dati" qualora fosse necessario	UC_2, UC_3, UC_3.2
RFO_17	L'utente deve poter configurare il parametro "*delay*" qualora fosse necessario	UC_2, UC_3, UC_3.3
RFO_18	L'utente deve poter configurare il parametro "durata" qualora fosse necessario	UC_2, UC_3, UC_3.4
RFO_19	L'utente deve poter configurare il parametro "esteso" qualora fosse necessario	UC_2, UC_3, UC_3.5
RFO_20	L'utente deve poter configurare il parametro "intervallo_id" qualora fosse necessario	UC_2, UC_3, UC_3.6
RFO_21	L'utente deve poter visualizzare un messaggio d'errore qualora inserisca un intervallo non valido, ovvero l'estremo inferiore è maggiore dell'estremo superiore, nel parametro "intervallo_id"	UC_2, UC_3, UC_3.6.1
RFO_22	L'utente deve poter configurare il parametro "intervallo_servizi" qualora fosse necessario	UC_2, UC_3, UC_3.7

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFO_23	L'utente deve poter visualizzare un messaggio d'errore qualora inserisca un intervallo non valido, ovvero l'estremo inferiore è maggiore dell'estremo superiore, nel parametro "intervallo_servizi"	UC_2, UC_3, UC_3.7.1
RFO_24	L'utente deve poter configurare il parametro "*timeout*" qualora fosse necessario	UC_2, UC_3, UC_3.8
RFO_25	L'utente deve poter configurare il parametro "intervallo_did" qualora fosse necessario	UC_2, UC_3, UC_3.9
RFO_26	L'utente deve poter visualizzare un messaggio d'errore qualora inserisca un intervallo non valido, ovvero l'estremo inferiore è maggiore dell'estremo superiore, nel parametro "intervallo_did"	UC_2, UC_3, UC_3.9.1
RFO_27	L'utente deve poter configurare il parametro "cambia_sessione" qualora fosse necessario	UC_2, UC_3, UC_3.10
RFO_28	L'utente deve poter configurare il parametro "sessioni_supportate" qualora fosse necessario	UC_2, UC_3, UC_3.11
RFO_29	L'utente deve poter configurare il parametro "livelli_supportati" qualora fosse necessario	UC_2, UC_3, UC_3.12
RFO_30	L'utente deve poter cliccare il pulsante per eseguire il test nel dispositivo Raspberry Pi	UC_5

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFO_31	L'utente deve poter visualizzare il risultato del test che ha eseguito	UC_6
RFO_32	L'utente deve poter visualizzare lo stato del test che ha eseguito	UC_6.1
RFO_33	L'utente deve poter visualizzare i campi di configurazione del Can bus	UC_7, UC_7.1
RFO_34	L'utente deve poter visualizzare il campo di configurazione "canale" del Can bus	UC_7.1.1
RFO_35	L'utente deve poter visualizzare il campo di configurazione "interfaccia" del Can bus	UC_7.1.2
RFO_36	L'utente deve poter visualizzare il campo di configurazione "*bitrate*" del Can bus	UC_7.1.3
RFO_37	L'utente deve poter visualizzare i campi di configurazione ISO-TP	UC_7, UC_7.2
RFO_38	L'utente deve poter visualizzare il campo di configurazione "id_richiesta" ISO-TP	UC_7.2.1
RFO_39	L'utente deve poter visualizzare il campo di configurazione "id_risposta" ISO-TP	UC_7.2.2
RFO_40	L'utente deve poter visualizzare il campo di configurazione "esteso" ISO-TP	UC_7.2.3
RFO_41	L'utente deve poter configurare i campi di configurazione del Can bus	UC_8, UC_8.1
RFO_42	L'utente deve poter configurare il campo "canale" del Can bus	UC_8.1.1
RFO_43	L'utente deve poter configurare il campo "interfaccia" del Can bus	UC_8.1.2
RFO_44	L'utente deve poter configurare il campo "*bitrate*" del Can bus	UC_8.1.3

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFO_45	L'utente deve poter configurare i campi di configurazione ISO-TP	UC_8, UC_8.3
RFO_46	L'utente deve poter configurare il campo "id_richiesta" ISO-TP	UC_8.3.1
RFO_47	L'utente deve poter configurare il campo "id_risposta" ISO-TP	UC_8.3.2
RFO_48	L'utente deve poter configurare il campo "esteso" ISO-TP	UC_8.3.3
Desiderabili		
RFD_1	L'utente deve poter visualizzare il bottone per importare la configurazione di un test via file .yml	UC_1.1.1.14
RFD_2	L'utente deve poter cliccare il bottone per importare la configurazione di un test via file .yml	UC_4
RFD_3	L'utente vede il form di configurazione del test compilato prendendo i dati dal file .yml importato	UC_4
RFD_4	L'utente deve poter visualizzare il pulsante per esportare il risultato in un file .yml	UC_9
RFD_5	L'utente deve cliccare visualizzare il pulsante per esportare il risultato in un file .yml	UC_9
RFD_6	L'utente deve poter visualizzare il pulsante per esportare il risultato in un file .json	UC_10
RFD_7	L'utente deve cliccare visualizzare il pulsante per esportare il risultato in un file .json	UC_10

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFD_8	L'utente deve poter visualizzare il bottone per importare la configurazione del Can bus di sessione da file .yml	UC_7.1.4
RFD_9	L'utente deve poter cliccare il bottone per importare la configurazione del Can bus di sessione da file .yml	UC_8.2
RFD_10	L'utente deve poter visualizzare il form di configurazione Can Bus compilato prendendo i dati dal file importato	UC_8.2
RFD_11	L'utente deve poter visualizzare il bottone per importare la configurazione ISO-TP di sessione da file .yml	UC_7.2.4
RFD_12	L'utente deve poter cliccare il bottone per importare la configurazione ISO-TP di sessione da file .yml	UC_8.4
RFD_13	L'utente deve poter visualizzare il form di configurazione ISO-TP compilato prendendo i dati dal file importato	UC_8.4
RFD_14	L'utente deve poter visualizzare il pulsante "Crea workflow"	UC_11
RFD_15	L'utente deve poter cliccare il pulsante "Crea workflow"	UC_11
RFD_16	L'utente deve poter visualizzare i test selezionabili per il workflow	UC_11.1
RFD_17	L'utente deve poter selezionare un test, tra quelli possibili, da aggiungere al workflow	UC_11.1
RFD_18	L'utente deve poter configurare i parametri del test selezionato	UC_11.2

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFD_19	L'utente deve poter configurare il parametro "id" del test selezionato, qualora fosse necessario	UC_11.2.1
RFD_20	L'utente deve poter configurare il parametro "data" del test selezionato, qualora fosse necessario	UC_11.2.2
RFD_21	L'utente deve poter configurare il parametro "*delay*" del test selezionato, qualora fosse necessario	UC_11.2.3
RFD_22	L'utente deve poter configurare il parametro "durata" del test selezionato, qualora fosse necessario	UC_11.2.4
RFD_23	L'utente deve poter configurare il parametro "esteso" del test selezionato, qualora fosse necessario	UC_11.2.5
RFD_24	L'utente deve poter configurare il parametro "intervallo_id" del test selezionato, qualora fosse necessario	UC_11.2.6
RFD_25	L'utente deve poter visualizzare un messaggio d'errore qualora inserisca un intervallo non valido, ovvero l'estremo inferiore è maggiore dell'estremo superiore, nel parametro "intervallo_id"	UC_11.2.6.1
RFD_26	L'utente deve poter configurare il parametro "intervallo_servizi" del test selezionato, qualora fosse necessario	UC_11.2.7

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFD_27	L'utente deve poter visualizzare un messaggio d'errore qualora inserisca un intervallo non valido, ovvero l'estremo inferiore è maggiore dell'estremo superiore, nel parametro "intervallo_servizi"	UC_11.2.7.1
RFD_28	L'utente deve poter configurare il parametro "*timeout*" del test selezionato, qualora fosse necessario	UC_11.2.8
RFD_29	L'utente deve poter configurare il parametro "intervallo_did" del test selezionato, qualora fosse necessario	UC_11.2.9
RFD_30	L'utente deve poter visualizzare un messaggio d'errore qualora inserisca un intervallo non valido, ovvero l'estremo inferiore è maggiore dell'estremo superiore, nel parametro "intervallo_did"	UC_11.2.9.1
RFD_31	L'utente deve poter configurare il parametro "cambia_sessione" del test selezionato, qualora fosse necessario	UC_11.2.10
RFD_32	L'utente deve poter configurare il parametro "sessioni_supportate" del test selezionato, qualora fosse necessario	UC_11.2.11
RFD_33	L'utente deve poter configurare il parametro "livelli_supportati" del test selezionato, qualora fosse necessario	UC_11.2.12
RFD_34	L'utente, tramite una checkbox, deve poter indicare se il test deve utilizzare il contesto del workflow	UC_11.2.13

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFD_35	L'utente deve poter visualizzare il pulsante "Aggiungi al workflow"	UC_11.3
RFD_36	L'utente deve poter cliccare il pulsante "Aggiungi al workflow"	UC_11.3
RFD_37	L'utente deve poter visualizzare il pulsante "Esegui workflow"	UC_12
RFD_38	L'utente deve poter cliccare il pulsante "Esegui workflow"	UC_12
RFD_39	L'utente deve poter visualizzare il risultato di ogni test del workflow	UC_13, UC_13.1
RFD_40	L'utente deve poter visualizzare lo stato del risultato di ogni test del workflow	UC_13.1.1
Opzionali		
RFOP_1	L'utente deve visualizzare la pagina di storico delle esecuzioni	UC_14
RFOP_2	L'utente deve visualizzare un singolo elemento dello storico	UC_14.1
RFOP_3	L'utente deve visualizzare l'id univoco associato all'esecuzione del test	UC_14.1.1
RFOP_4	L'utente deve poter visualizzare un test dello storico nel dettaglio	UC_14.2
RFOP_5	L'utente deve poter visualizzare il nome del test presente nello storico	UC_14.2.1
RFOP_6	L'utente deve poter visualizzare il test dello storico nel dettaglio	UC_15
RFOP_7	L'utente deve poter visualizzare i parametri della configurazione di un test	UC_15.1

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFOP_8	L'utente deve poter visualizzare il risultato del test	UC_15.2
RFOP_9	L'utente deve poter visualizzare lo stato del risultato del test	UC_15.2.1
RFOP_10	L'utente deve visualizzare il bottone "esporta risultato". Per esportare i risultati del test in un file .yml	UC_15.3
RFOP_11	L'utente deve poter cliccare il bottone "esporta risultato". Per esportare i risultati del test in un file .yml	UC_15.4
RFOP_12	L'utente deve poter visualizzare il pulsante "Rilancia test"	UC_15.5
RFOP_13	L'utente deve poter cliccare il pulsante "Rilancia test"	UC_15.6
RFOP_14	L'utente deve poter visualizzare un singolo workflow all'interno della pagina storico test	UC_14.3
RFOP_15	L'utente deve poter visualizzare il numero di test presenti all'interno del workflow	UC_14.3.1
RFOP_16	L'utente deve poter visualizzare il workflow, presente nello storico, nel dettaglio	UC_16
RFOP_17	L'utente deve poter visualizzare il workflow nel dettaglio	UC_16
RFOP_18	L'utente deve poter visualizzare il pulsante "Esporta workflow"	UC_16.1
RFOP_19	L'utente deve poter cliccare il pulsante "Esporta workflow", che esporta i dettagli del workflow in un file .yml	UC_16.2

Continua nella prossima pagina...

Tabella 3.1 – Continuo della tabella

Requisito	Descrizione	Use Case
RFOP_20	L'utente deve poter visualizzare il bottone "Rilancia workflow"	UC_16.3
RFOP_21	L'utente deve poter cliccare il bottone "Rilancia workflow"	UC_16.4
RFOP_22	L'utente deve poter visualizzare il singolo test presente nel workflow	UC_16.5
RFOP_23	L'utente deve poter visualizzare la configurazione di ogni singolo test presente nel workflow	UC_16.6
RFOP_24	L'utente deve poter visualizzare il pulsante "Esegui test" per ogni test presente nel workflow	UC_16.7
RFOP_25	L'utente deve poter cliccare il pulsante "Esegui test" per ogni test presente nel workflow	UC_16.8
RFOP_26	L'utente deve poter visualizzare i risultati di ogni test presente nel workflow	UC_16.9
RFOP_27	L'utente deve poter visualizzare lo stato dei risultati di ogni test presente nel workflow	UC_16.9.1

Tabella 3.2: Tabella del tracciamento dei requisiti qualitativi.

Requisito	Descrizione	Use Case
RQO_1	È necessario utilizzare Gitlab come piattaforma di gestione della repository git remota	-
Continua nella prossima pagina...		

Tabella 3.2 – Continuo della tabella

Requisito	Descrizione	Use Case
RQO_2	È necessario utilizzare git come strumento di versionamento del codice sorgente	-

Tabella 3.3: Tabella del tracciamento dei requisiti di vincolo.

Requisito	Descrizione	Use Case
RVO_1	Utilizzo Raspberry Pi OS come sistema operativo nel dispositivo Raspberry	-
RVO_2	Il dispositivo Raspberry Pi deve essere in grado di ricevere ed inviare frame Can	-
RVO_3	La comunicazione Can deve avvenire tramite interfaccia compatibile *socketcan* [12]	-
RVO_4	Le funzionalità diagnostiche devono essere conformi allo standard ISO 14229	-
RVO_5	Il sistema deve supportare frame secondo lo standard 15765-2 (ISO-TP)	-
RVO_6	Il backend applicativo deve essere sviluppato in Python	-
RVO_7	La gestione del bus CAN deve utilizzare la libreria python-can	-
RVO_8	Le funzionalità diagnostiche UDS devono utilizzare la libreria udsoncan	-
RVO_9	La gestione dei frame ISO-TP deve utilizzare la libreria `can-isotp`	-
RVO_10	L'accesso alle funzionalità deve avvenire tramite API *REST*$_G$ (Livello 2 del Richardson Maturity Model)	-
Continua nella prossima pagina...		

Tabella 3.3 – Continuo della tabella

Requisito	Descrizione	Use Case
RVO_11	Le comunicazioni remote devono utilizzare protocollo HTTP/HTTPS	-

Capitolo 4

Progettazione e codifica

In questo capitolo viene descritto come il prodotto è stato progettato, in particolare nelle architetture adottate sia frontend che backend.

4.1 Progettazione

Una volta individuati i casi d'uso e i requisiti che l'ambiente dovrà soddisfare, è stato compiuto uno studio con l'intento di trovare quale architettura backend e frontend fosse quella più adatta al progetto.

4.1.1 Architettura Backend

Per l'architettura backend è stata adottata un'architettura di tipo "*layered*" a tre livelli:

- Applicazione;
- Dominio;
- Persistenza.

La suddivisione del sistema in tre livelli è sufficiente a garantire un basso livello di accoppiamento, favorendo così la scalabilità e la manutenibilità.

All'inizio è stata presa in considerazione l'adozione un'architettura esagonale, ma valutazioni successive con il tutor aziendale hanno portato alla bocciatura di quest'ultima: in quanto il suo sviluppo e la sua manutenzione risultavano più

onerosi di quanto fosse necessario e richiesto rispetto al bacino d'utenza del prodotto.

Applicazione

Nel livello di applicazione si trovano gli endpoint dell'API. Gli endpoint sono stati progettati per rispettare il livello 2 del *Richardson Maturity Model (RMM)*$_G$ [11], come da requisito di vincolo.

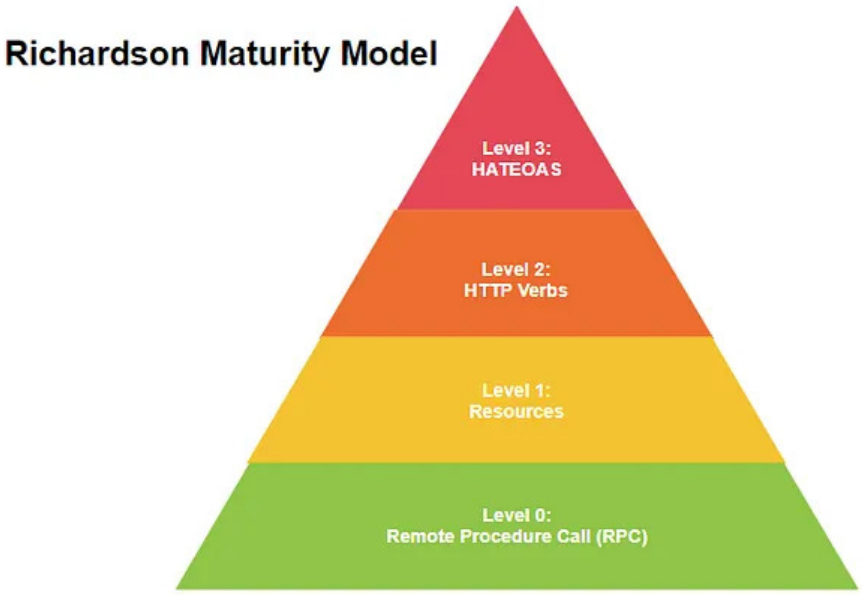

Figura 4.1: Livelli del RMM [11]

Come si evince dalla Figura 4.1 vi sono 4 livelli del RMM, dove più si sale di livello più si rispettano i principi REST:

1. **Livello 0**: consiste in chiamate a procedure remote, ovvero richiesta di esecuzione di codice da remoto tramite protocollo HTTP, senza tuttavia usarne le funzionalità;

2. **Livello 1**: consiste nell'introduzione delle risorse. Ogni risorsa ha il suo *endpoint* dedicato rendend l'API più facile da strutturare e navigare;

3. **Livello 2**: consiste nell'utilizzo corretto e coerente dei metodi HTTP, non limitandosi solamente all'utilizzo di richieste *GET* e *POST*, ma usare il metodo corretto per l'azione corretta;

4. **Livello 3**: consiste nell'utilizzo degli *Hypermedia As The Engine Of Application State (HATEOAS)*$_G$. In questo livello si rispettano completamente tutti i principi REST: la risposta dell'API contiene i link delle possibili azioni future che sono possibili a partire dalla risorsa richiesta. In questo modo il client non ha bisogno di conoscenze preliminari sulla struttura dell'API, rappresentando così il pieno soddisfacimento del vincolo HATEOAS e, di conseguenza, dell'architettura REST [5].

Dominio

Nel livello di dominio si trovano tutte le classi e funzioni python che effettuano le operazioni logiche e delegazione alle classi facenti parte del livello di persistenza.

Le classi di dominio degne di nota, ovvero che si interfacciano con CAN o ne fanno uso sono:

- *TargetCANTP*
- *BaseTest*
- *TestService*
- *Pipeline*

La classe `TargetCANTP` rappresenta il target ECU che dovrà essere testato: compreso di configurazione Can bus e ISO-TP.

È composta dagli attributi:

- *id_richiesta*: Rappresenta l'id con cui la richiesta can verrà effettuata;

- *id_risposta*: Rappresenta l'id con cui l'ECU ci si aspetti risponda;

- *canale*: Rappresenta la porta fisica da impiegare per la comunicazione. Perché effettivamente avvenga lo scambio di dati è importante mettere in stato *"UP"* la suddetta porta;

- *interfaccia*: Rappresenta L'interfaccia di accesso al bus. Siccome i test verranno effettuati in un dispositivo Raspberry Pi con Raspberry OS, il valore di *default* è *socketcan* [12];

- *bitrate*: Rapprsenta la velocità di scambio dati nel bus;

- *esteso*: `"True"` se l'id della richiesta è in formato esteso;

- *log*: `"True"` se si vuole stampare i messaggi scambiati durante la comunicazione, usato principalmente per *Debug*$_G$;

- *isotp_params*: Rappresenta i parametri di configurazione ISO-TP.

`TargetCANTP` espone i metodi:

- *get_isotp_transport_layer*: Questo metodo ritorna un dizionario che contiene la configurazione con cui è stato istanziato l'oggetto di tipo `TargetCANTP`;

- *get_target_config*: Questo metodo ritorna un dizionario ha come coppia chiave-valore l'attributo e il valore con cui è stato inizializzato, fatta eccezione per il CAN bus e lo *stack* ISO-TP;

- *send*: Questo metodo ha come parametri:

  – Dati da inviare;

  - Id messaggio, in caso si voglia inviare un messaggio con un id diverso da quello configurato durante la costruzione;

  - Se l'id del messaggio è nella forma estesa o meno.

  Invia i dati, passati come parametro, nel bus CAN;

- *recv*: Questo metodo mette il bus in uno stato di attesa di messaggi;

- *close*: Questo metodo chiude la connessione con il bus.

La classe `BaseTest` invece è una classe astratta e rappresenta il singolo test di sicurezza che dovrà essere eseguito dalla ECU. Tutti i test di sicurezza implementati infatti ereditano direttamente da questa classe.

È composta dagli attributi:

- *target*: Un oggetto di tipo `TargetCANTP`.

Espone i metodi:

- *run*: Questo è un metodo astratto, la sua implementazione deve essere il test di sicurezza vero e proprio. Per esempio: se si vuole implementare un attacco di tipo DoS_G, il metodo `run` dovrà contenere il codice per inviare un gran numero di frame in poco tempo alla ECU per intasarne la memoria e provocare dei rallentamenti del sistema;

- *stop*: Questo metodo interrompe l'esecuzione del test scollegandosi anche dal bus;

- *get_parameters*: Questo è un metodo statico e virtuale: ritorna la lista di parametri di configurazione del test concreto.

La classe `TestService` ha la responsabilità di invocare la creazione del test, fornendogli la configurazione, lanciarne l'esecuzione ed invocare il salvataggio nel *database* del risultato del test.

È composta dall' attributo:

- *target*: Un oggetto di tipo `TargetCANTP`.

Espone i metodi:

- *run_test*: Questo metodo prende come parametro la configurazione del test. Con quest'ultima, crea l'oggetto di tipo `BaseTest` corrispondente, ne invoca il metodo `run` che ritorna il risultato del test e delega il salvataggio nel *database* del risultato alla classe `TestRepository`.

La classe `Workflow` rappresenta appunto un workflow. In quanto tale, ha la responsabilità di eseguire un insieme di test e, qualora fosse necessario, fornire, come input, il risultato dei test tramite un contesto condiviso fra i test di un particolare workflow.

È composta dagli attributi:

- *test_service*: Un oggetto di tipo `TestService`. Usufruisce di un oggetto `TestService` così da eseguire, in ordine, la serie di test che l'utente ha specificato.

- *context*: Rappresenta il contesto del workflow. Qui vengono salvati gli output dei test tramite un dizionario che ha come chiave il nome dell'attributo di configurazione.
  Per esempio: Se un test produce in output i livelli UDS supportati dalla ECU. All'interno del contesto sarà presente la chiave "livelli_supportati" con i livelli trovati dal test.

Espone i metodi:

- *run*: Questo metodo riceve come parametro una lista di dizionari, ovvero la lista di configurazione dei test da eseguire. Se un test dichiara l'intenzione di voler utilizzare il contesto allora invoca il metodo `validate`.

In ogni caso esegue ogni test in ordine, usufruendo del metodo `run_test` della classe `TestService`, salvando i risultati in una lista che verrà immediatamente ritornata quando l'esecuzione di tutti i test sarà terminata;

- *validate*: Questo metodo viene invocato all'interno di `run`. Riceve come parametro un dizionario che rappresenta la configurazione del test in esame e verifica che possa usufruire dei dati presenti all'interno del contesto.

Persistenza

Nel livello di persistenza si trovano tutte le classi e funzioni python che vanno direttamente a leggere o scrivere nel *database*.

Nel contesto del progetto, una sola classe appartiene a questo livello: *TestRepository*. Questa classe ha il compito di connettersi al *database* e svolgere *Query*$_G$ di lettura e scrittura.

È composta dagli attributi:

- *engine*: Questo è un attributo di tipo `Engine`.
  `Engine` è una classe che fornisce la libreria `sqlalchemy` e rappresenta una connessione al *database*.

Espone i metodi:

- *save_target*: Questo metodo prende come parametri il target, un oggetto di tipo `TargetCANTP`, e il nome con cui salvare tale target nel *database*. Successivamente ritorna `True` se il salvataggio è andato a buon fine, `False` altrimenti;

- *save_test_config*: Questo metodo prende come parametri un dizionario che contiene come chiavi tutti i parametri di configurazione di un test e li salva sul *database*.
  Questo per garantire all'utente finale di salvare le configurazioni preferite.
  Successivamente ritorna `True` se il salvataggio è andato a buon fine, `False`

altrimenti;

- *save_test_in_history*: Questo metodo prende in input il risultato di un test, che comprende anche la sua configurazione, e lo salva nel *database* nella tabella dello storico. Successivamente ritorna `True` se il salvataggio è andato a buon fine, `False` altrimenti;

- *get_test_from_history*: Questo metodo prende in input un id e ritorna il test salvato nello storico, se presente;

- *get_target*: Questo metodo prende in input il nome con cui è stato salvato il target e ritorna l'oggetto `TargetCANTP` corrispondente, se presente;

- *get_config*: Questo metodo prende in input il nome con cui è stata salvata una determinata configurazione e la ritorna sotto forma di dizionario, se presente;

- *get_all_targets*: Questo metodo ritorna una lista di tuple contenente tutti i target `TargetCANTP` e i nomi con cui sono stati salvati;

- *get_all_test_configs*: Questo metodo ritorna una lista che contiene tutte le configurazioni salvate dall'utente;

- *get_complete_history*: Questo metodo ritorna una lista che contiene tutti i test presenti nello storico.

4.1.2 Architettura Frontend

Nel frontend è stata adottata l'achitettura *Feature Sliced Design (FSD)*$_G$ [4]: per ridurre l'accopiamento tra il codice frontend e una migliore separazione di

responsabilità, in quanto FSD concentra la propria suddivisione per tipologia logica di funzionalità piuttosto che layer puramente tecnici.

Sebbene FSD sia un'architettura nata per accomodare le esigenze di *React*$_G$, un approccio modulare e orientato alle funzionalità si presta molto bene anche con Svelte perché la sua natura compilativa trasforma i confini archietturali in prestazioni: isolando lo stato in entità autonome, si minimizza ogni invalidazione reattiva superflua, incrementando così la capacità del compilatore di generare codice.

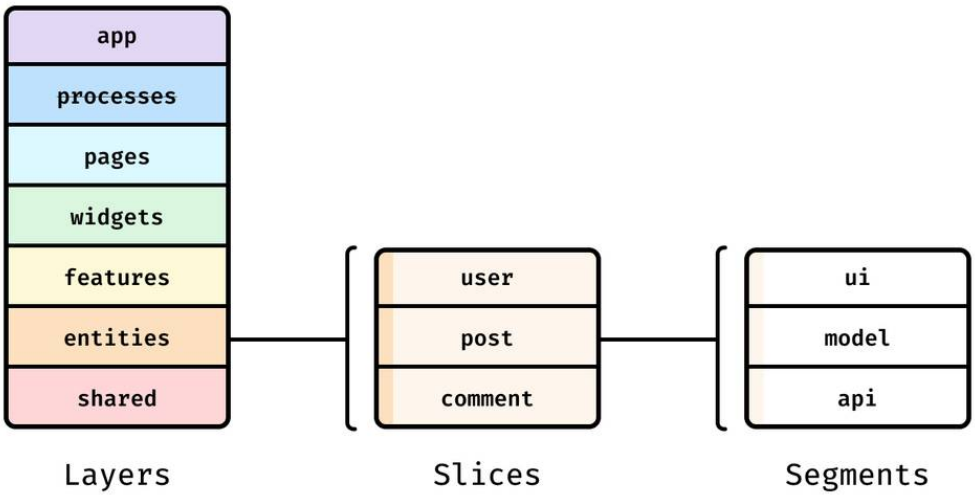

Figura 4.2: Organizzazione FSD [4]

FSD è composto da 7 *layers*, vedi Figura 4.2, e sono standardizzati in tutti i progetti che adottano FSD, questi sono [4]:

- *App*: tutto quello che fa funzionare l'app. Ad esempio: stili globali, routing;
- *Process* (Deprecato): usato per scenari complessi dentro la stessa pagina;
- *Pages*: le pagine che verranno visualizzate;

- *Widgets*: qui vengono inserite grandi porzioni autonome di funzionalità o interfaccia utente, che solitamente risolvono un caso d'uso completo;

- *Features*: qui vengono inserite implementazioni riutilizzabili di intere funzionalità di prodotto, ovvero azioni che portano valore di business all'utente. Per esempio: la funzionalità di "Aggiungi al carrello", questa funzionalità porta valore di business all'utente (capacità di acquistare) e può essere riutilizzata in diverse parti del sito. Non è solo il pulsante "Aggiungi al carrello", ma tutta l'implementazione dell'azione di acquisto che ci sta dietro;

- *Entities*: entità di dominio su cui il progetto lavora. Riprendendo l'esempio di "Aggiungi al carrello", un'entità può essere il prodotto che viene aggiunto e in seguito acquistato;

- *Shared*: funzionalità riutilizzabili che però trascendono le specifiche di dominio del progetto.
  Rimanendo nel contesto di "Aggiungi al carrello": il componente "*Button*" viene inserito in questo *layer*. In quanto può essere impiegato anche in altri contesti di dominio oltre che all'aggiunta del carrello.

I *layers* inoltre, sono organizzati in una struttura gerarchica: ogni *layer* può utilizzare codice da *layer* sottostanti, ma non da *layer* soprastanti, dunque se prendiamo come riferimento Figura 4.2: codice che sta nel *layer features* può utilizzare codice dei *layers entities* e *shared*, ma non di *widgets*, *pages* e *app* [4].

Ogni *layer* è composto da un insieme di *slices* e ogni *slice* è composto da *segments*, con le uniche eccezioni rappresentatate dai *layers shared* e *app*, che sono composti solo da *segments*.

Uno *slice* partiziona il codice per dominio all'interno di un *layer*. Non c'è un limite al numero di *slices* per *layer*. Questi rendono la base di codice più facile da navigare ragruppando moduli che sono logicamente interconnessi. Una nota

importante da aggiungere è che *slices* dello stesso layer non possono utilizzare o importare codice da *slices* dello stesso layer.

Le *slices*, così come i *layers shared* e *app*, sono composti da *segments*.

I *segments* raggruppano il codice per scopo piuttosto che funziononalità logica. Non ci sono *segments* standard, ma quelli più comuni che ho adottato nel mio progetto sono:

- *ui*: Tutto il codice responsabile per la visualizzazione grafica;
- *lib*: Codice comune che serve ai moduli dello stesso *slice*;
- *api*: Interazioni con il *backend* come: richiste, tipi e adattatori.

4.2 Design Pattern utilizzati

Durante la progettazione sono emersi problemi che hanno richiesto l'impiego di *Design patterns*$_G$: la necessità di costruire la risposta di un test con parametri e valori che differiscono fra test, la necessità di avere un'interfaccia unica e scalabile dove invocare il test corretto tra una moltitudine di test ed infine evitare il più possibile l'accoppiamento tra classi.

I design patterns che ho impiegato sono:

- *Dependency Injection*
- *Builder* [6]
- *Strategy* [6]

4.2.1 Dependency Injection

Il pattern dependency injection è stato impiegato via costruttore in quasi ogni classe di dominio, così da evitare il più possibile l'accoppiamento tra le classi. Nel backend, tuttavia, ho usato anche la libreria python `dependency-injector`

(versione 4.49.0) in modo da creare una sola istanza di `engine`, ovvero una sola connessione attiva al *database* alla volta.

Inoltre questo pattern migliora molto la testabilità del codice in quanto permette di sostituire dipendenze reali con dei mock, ovvero oggetti simulati e preconfigurati per lo scopo del test.

4.2.2 Builder

Il pattern Builder è stato impiegato per la costruzione strutturata dell'oggetto di risposta `TestAnswer`, restituito dal metodo `run` di `BaseTest`. Durante l'esecuzione `run` invoca il metodo `add_output` in modo da costruire la risposta in base agli esiti del test. La costruzione della risposta termina con la chiamata al metodo `build_answer`, responsabile dell'assemblaggio e della restituzione di un dizionario contenente la configurazione del test, i risultati ottenuti ed eventuali eccezioni rilevate durante l'esecuzione.

4.2.3 Strategy

Il pattern Strategy è stato impiegato nel contesto di `BaseTest`: ci sono molti test di sicurezza possibili da eseguire in una ECU. Attraverso una classe base astratta `BaseTest` il `TestService` può eseguire polimorficamente qualsiasi test specifico, attraverso il metodo virtuale `run`. In questo modo elimino istruzioni condizionali complesse e facilito l'aggiunta di nuovi test di sicurezza semplicemente creando nuove sottoclassi, mantenendo comunque invariata la logica di esecuzione centrale, in piena adesione al principio *Open/Closed* [13].

Capitolo 5

Verifica e validazione

In questo capitolo vengono descritte le tecniche di verifica del codice sorgente con la copertura dei test. La validazione è stata effettuata tramite lo sviluppo di un PoC e la sua presentazione al tutor aziendale che ha riscontrato il corretto soddisfacimento dei requisiti individuati nel Capitolo 3.

5.1 Verifica

Per garantire qualità del codice sono stati implementati test di unità e di integrazione. La copertura del codice sorgente è stata misurata tramite diverse librerie ausiliarie che analizzano i test e forniscono una percentuale di codice sorgente coperto.

5.1.1 Test di unità

I test di unità sono stati scritti per testare la logica di singole porzioni di codice, unità, in modo da garantire il più possibile che il codice sorgente nel suo insieme sia corretto. Inoltre, visto la loro granularità, permettono di individuare con precisione dove il codice produce errori oppure restituisce valori errati.

Backend

Sono state testate tutte le classi di dominio facendo un largo uso delle `fixture` messe a disposizione da pytest.

Le `fixture` di pytest forniscono un modo deterministico per definire il contesto di un test. In combinazione con `unittest.mock`, la libreria standard python per effettuare i mock, sono in grado di isolare intere porzioni di codice in modo da testare la logica interna delle singole unità.

L'analisi della copertura del codice è stata effettuata mediante l'utilizzo della libreria `pytest-cov` [15]. Questo tool ha permesso di analizzare il codice eseguito durante il testing e al termine delle misurazioni, la copertura finale si è attestata sulla soglia dell'80%.

Frontend

Nel frontend oltre alle unità di codice, è stato verificato anche il rendering dei componenti attraverso la libreria `@testing-library/svelte` (Svelte Testing Library). Quest'ultima permette di simulare il comportamento dei componenti all'interno di un ambiente DOM virtuale.

Nello specifico, la verifica del rendering dei componenti è stata effettuata mediante l'utilizzo della funzione `render`, del modulo `screen` per effettuare query sugli elementi visibili nella pagina e l'oggetto `userEvent` per simulare le azioni dell'utente, come click dei pulsanti o l'inserimento di testo nei form.

L'analisi della copertura del codice è stata effettuata mediante l'utilizzo della libreria `@vitest/coverage-v8`. Questo tool ha permesso di analizzare il codice eseguito durante il testing e al termine delle misurazioni, la copertura finale si è attestata sulla soglia dell'88%.

5.1.2 Test di integrazione

Nel progetto sono stati implementati dei test di integrazione per verificare la corretta interazione tra il layer applicativo e quello di dominio, assicurando così una corretta comunicazione dei dati via richieste HTTP.

Nello specifico, la verifica si è concentrata sui due endpoint principali esposti dal backend tramite FastAPI [18]:

- `/test`: questo endpoint esegue una singola tipologia di attacco informatico (come un attacco DoS);

- `/workflow`: questo endpoint esegue un workflow, in cui vengono concatenati più test consecutivi (come analisi di fuzzing o algoritmi di verifica della sicurezza UDS).

Per l'implementazione di questi test è stato impiegata la classe `TestClient` messa a disposizione da FastAPI, la quale permette di emulare l'invio di richieste HTTP, effettuare controlli sugli stati di risposta e sul payload JSON restituito dall'endpoint in esame.

Per isolare la logica applicativa dalle dipendenze esterne, come il database o le interfacce di rete come *SocketCAN*, si è fatto uso della parola chiave `with` in combinazione con il modulo `unittest.mock.patch`. Questo ha permesso di sostituire durante l'esecuzione i componenti critici (quali `TestService` e `PersistanceService`) con mock, consentendo di verificare in modo deterministico il corretto coordinamento delle componenti di sistema, il parsing dei parametri di configurazione e il conteggio delle chiamate delle funzioni appartenenti alle logiche interne.

L'uso della parola chiave `with` ha permesso inoltre di usufruire dei mock solo per l'esecuzione locale del test, senza alterare il comportamento di altri test di integrazione.

5.2 Validazione tramite PoC

Per la validazione è stato implementato un PoC con lo scopo di illustrare al tutor aziendale che il sistema soddisfa i requisiti individuati durante l'analisi.

Il PoC è stato sviluppato creando una semplice interfaccia grafica interattiva tramite Svelte, quest'ultima invoca gli endpoint dell'API fornendo loro la configurazione dei test che ha inserito l'utente. L'utilizzatore ha piena libertà nell'inserimento dei valori, gli unici controlli sono quelli di compatibilità di tipo. La piena libertà è una funzionalità voluta in accordo con il tutor aziendale. Si è deciso che limitare i valori inseribili sarebbe stato un danno piuttosto che una funzionalità di sicurezza, in quanto questi controlli avrebbero impedito al test di considerare valori che causerebbero malfunzionamenti o fuga di dati, rendendo paradossalmente inutile il test di sicurezza.

Gli endpoint danno inizio al processo di esecuzione dei test di sicurezza, instanziando un oggetto di tipo `TestService` o `Workflow` ed invocando rispettivamente il metodo `run_test` o `run`, e rimanendo in attesa della risposta. Terminate tutte le esecuzioni, l'API ritornerà un JSON con i risultati che verranno poi mostrati all'utente finale.

La validazione è stata effettuata mediante un'esecuzione del PoC: è stata collegata la scheda ECU al dispositivo Rasberry Pi e tramite l'interfaccia web esposta da quest'ultimo è stato possibile eseguire i test e i workflow da un dispositivo remoto. Il tutto è avvenuto sotto la supervisione del tutor aziendale, che ha potuto constatare il corretto funzionamento del sistema rispetto alle funzionalità attese.

Capitolo 6

Conclusioni

In questo capitolo vengono riportate le conclusioni del lavoro svolto, con un consuntivo finale, una riflessione sulle conoscenze acquisite e una valutazione personale.

6.1 Consuntivo delle attività e degli obiettivi

Lo stage ha avuto una durata complessiva di 300. La progettazione e lo sviluppo del progetto ha avuto una durata di circa 260 ore, e le restanti 40 ore sono state impiegate per lo sviluppo dei test di unità e di integrazione.

La maggior parte del tempo è stata impiegata per l'analisi dei requisiti e la progettazione del sistema, mentre l'attività di sviluppo ha richiesto circa 60 ore a fronte delle 20 pianificate inizialmente. Per questo motivo non è stato possibile sviluppare il PoC dell'estendibilità del sistema tramite un altro canale di comunicazione, attività per la quale erano state preventivate 40 ore. Nonostante questo, è stato rispettato il monte orario di 300 ore previsto dal piano di lavoro.

La Tabella 6.1 mostra le attività presenti nel piano di lavoro e il tempo impiegato rispetto al tempo previsto per ciascuna attività.

Tabella 6.1: Tabella che illustra il consuntivo delle attività del piano di lavoro.

Attività	Ore pianificate	Ore effettive
Scelta delle tecnologie, setup dell'ambiente di sviluppo e definizione dei requisiti minimi	40	40
Setup del canale di comunicazione (CAN) ed esecuzione manuale di test	40	40
Definizione di una struttura per descrivere i test e implementazione di un test runner	40	40
Sviluppo di API per l'esecuzione di test da remoto	40	40
Refactoring test esistenti	40	40
Refactoring e modularizzazione del sistema di test	40	40
Documentazione e sviluppo di un'applicazione dimostrativa	20	60
PoC dell'estendibilità del sistema tramite setup di un altro canale di comunicazione (es. UART)	40	0

6.1.1 Requisiti soddisfatti

A fine progetto, tutti i requisiti funzionali obbligatori, definiti nella Tabella 3.1, così come quelli di vincolo, definiti nella Tabella 3.3, e di qualità, definiti nella Tabella 3.2, sono stati soddisfatti. Sono stati soddisfatti inoltre tutti i requisiti funzionali desiderabili riguardanti il workflow.

La Tabella 6.2 riassume i numeri dei requisiti soddisfatti.

Tabella 6.2: Tabella riepilogativa dei requisiti soddisfatti.

Tipologia	Soddisfatti	Numero Requisiti Definiti
Funzionali	88	115
Qualitativi	2	2
Di vincolo	11	11
Totali	98	128

6.1.2 Consuntivo degli obiettivi

A seguito della validazione da parte del tutor aziendale, gli obiettivi inizialmente fissati nel piano di lavoro, come riportati nella Tabella 2.2 sono stati in gran parte raggiunti, vedi Tabella 6.3. Tuttavia, come detto precedentemente nella Sezione 6.1, l'eccessivo impiego di ore nell'attività di sviluppo ha portato al mancato soddisfacimento dell'obiettivo F1, questo però non ha avuto gravi ripercussioni sul valore del progetto in quanto aveva una bassa priorità di implementazione. L'obiettivo F2, invece, è da considerarsi parzialmente completato in quanto è stato sviluppato solo lato backend senza integrazioni frontend lato utente.

Tabella 6.3: Tabella riepilogativa degli obiettivi raggiunti.

Codice	Stato
O1	Completato
O2	Completato
O3	Completato
O4	Completato
D1	Completato
D2	Completato
D3	Completato
Continua nella prossima pagina...	

Tabella 6.3 – Continuo della tabella

Codice	Stato
D4	Completato
F1	Non Completato
F2	Completato Parzialmente

6.2 Conoscenze acquisite

Questo tirocinio mi ha permesso di acquisire conoscenze in un ambito dell'informatica che non avevo mai affrontato prima: quello della sicurezza informatica nei sistemi embedded. Ho avuto l'occasione di approfondire il protocollo CAN e come cercare vulnerabilità in sistemi embedded.

L'affrontare tecnologie completamente nuove, come CAN, mi ha permesso di sviluppare una forte autonomia e capacità di ricerca; sopratutto oggi, con la tecnologia che sembra concentrarsi attorno all'intelligenza artificiale, dove la capacità di reperire informazioni, saperle filtrare e confrontare con la documentazione è fondamentale.

Oltre agli aspetti legati alla cybersecurity, il tirocinio ha incrementato le mie competenze nello sviluppo Full-Stack. Ho avuto modo di consolidare l'uso di Python attraverso il framework FastAPI per la realizzazione di API e, al contempo, tramite l'utilizzo di Svelte e l'architettura FSD, ho compreso come si sviluppano le interfacce utente basate su componenti reattivi.

Infine, la combinazione di `mock` e `fixture` mi ha permesso di comprendere come isolare le singole unità di codice per testarle in maniera deterministica.

6.3 Valutazione personale

Dato che non avevo mai lavorato all'interno di un contesto aziendale, ritengo che il tirocinio sia un'esperienza fondamentale nel mio percorso di laurea, in quanto ho avuto l'opportunità di lavorare su un progetto reale, con obiettivi

chiari e scadenze da rispettare, il che mi ha permesso di sviluppare capacità di *problem solving* e di gestione del tempo.

L'esperienza è stata ulteriormente valorizzata da Bluewind S.r.l., in quanto è stata molto disponibile nel fornire supporto e chiarimenti. Inoltre, il tutor aziendale ha sempre fornito feedback che mi sono stati utili per il progetto, rendendo l'esperienza di tirocinio molto stimolante.

In conclusione, il bilancio personale di questo tirocinio è pienamente positivo. L'attività svolta ha alimentato un forte interesse verso la sicurezza informatica e verso i sistemi embedded, dandomi una maggiore consapevolezza delle mie capacità e degli stimoli preziosi per il mio futuro percorso professionale.

Acronimi e abbreviazioni

API Application Program Interface. 3

DID Identificatori Di Dati. 12

ECU Electronic Control Units. 7

FSD Feature Sliced Design. 66

HATEOAS Hypermedia As The Engine Of Application State. 61

IDE Bit aggiuntivo di identificazione. 8

PCI Protocol Control Information. 10

PoC Proof of Concept. 5

RMM Richardson Maturity Model. 60

SPA Single Page Application. 3

UDS Unified Diagnostic Services. 11

Glossario

Bus Canale di comunicazione che permette a componenti elettroniche di comunicare tra loro. 7

Debug Atto di diagnostica del codice per scoprire la causa di un errore a *runtime*. 62

Design patterns Soluzioni architetturali ad un problema informatico ricorrente. 69

DoS Tipologia di attacco informatico con l'obiettivo di rendere un servizio inaccessibile sovraccaricando le risorse di sistema, saturando la memoria e la CPU del server. 63

Embedded Sistema hardware e software progettato per eseguire una sola funzione specifica. Questi sistemi sono ottimizzati per dimensioni, bassi consumi e costi. 3

Frame Unità di dati logica dello strato Data Link. 3

Query Interrogazione al *database* con l'obiettivo di inserire o leggere dati. 65

React Libreria JavaScript open source, utilizzata per la creazione di interfacce utente dinamiche, basata su un'architettura a componenti riutilizzabili. 67

Refactoring attività di riscrittura del codice sorgente per migliorarne la struttura e la leggibilità, senza alterarne il comportamento esterno. 4

Rest Stile architetturale per la creazione di servizi web. Le caratteristiche di questi servizi sono il fatto che:

- Utilizza risorse identificabili tramite indirizzo univoco;
- Utilizza protocollo HTTP;
- Stateless ovvero il server non memorizza alcuna informazione sullo stato del client tra una richiesta e l'altra.

Spesso adottato per la progettazione di API. 57

Stack Standard concettuale che suddivide logicamente l'architettura di rete in 7 strati: Fisico, Data Link, Rete, Trasporto, Sessione, Presentazione e Applicazione. 7

Workflow Insieme strutturato di attività che vengono eseguite in una sequenza logica ben definita. 5

Riferimenti bibliografici e sitografici

Testi

[5] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Università di California, 2000 (cit. a p. 61).

[6] Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994 (cit. a p. 69).

[7] ISO. *ISO 11898-1:2024: Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling*. 2024 (cit. a p. 7).

[8] ISO. *ISO 14229-1:2026: Road vehicles — Unified diagnostic services (UDS) Part 1: Application layer*. 2026 (cit. a p. 12).

[9] ISO. *ISO 14229-3:2022:Road vehicles — Unified diagnostic services (UDS) — Part 3: Unified diagnostic services on CAN implementation (UDSon-CAN)*. 2022 (cit. a p. 11).

[10] ISO. *ISO 15765-2:2024: Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) Part 2: Transport protocol and network layer services*. 2024 (cit. a p. 9).

[13] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002 (cit. a p. 70).

Siti

[1] Bluewind S.r.l. *Bluewind S.r.l.* 2026. URL: https://www.bluewind.it/ (visitato il giorno 30/06/2026) (cit. a p. 2).

[2] CSS Electronics. *CAN Bus Explained: A simple intro [2026].* 2026. URL: https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial (visitato il giorno 30/06/2026) (cit. alle pp. 7–9).

[3] CSS Electronics. *UDS Explained - A Simple Intro (Unified Diagnostic Services).* 2026. URL: https://www.csselectronics.com/pages/uds-protocol-tutorial-unified-diagnostic-services (visitato il giorno 30/06/2026) (cit. alle pp. 3, 11, 12).

[4] Feature-Sliced-Design community. *Feature-Sliced-Design Documentation.* URL: https://feature-sliced.design/ (visitato il giorno 30/06/2026) (cit. alle pp. 66–68).

[11] Anita Liberatore. *Richardson Maturity Model and HATEOAS: Understanding the Evolution of RESTful APIs.* 2026. URL: https://medium.com/@liberatoreanita/richardson-maturity-model-and-hateoas-understanding-the-evolution-of-restful-apis-53702173a138 (visitato il giorno 30/06/2026) (cit. a p. 60).

[12] Linux Kernel Organization. *SocketCAN - Controller Area Network.* URL: https://docs.kernel.org/networking/can.html (visitato il giorno 30/06/2026) (cit. alle pp. 57, 62).

[14] Pytest community. *pytest.* 2026. URL: https://docs.pytest.org/en/stable/ (visitato il giorno 30/06/2026) (cit. a p. 14).

[15] Pytest-cov community. *pytest-cov.* 2024. URL: https://pytest-cov.readthedocs.io/en/latest/ (visitato il giorno 30/06/2026) (cit. a p. 72).

[16] Python Software Foundation. *Python.* 2026. URL: https://www.python.org/ (visitato il giorno 30/06/2026) (cit. a p. 13).

[17] Raspberry Pi Foundation. *Raspberry Pi*. 2026. URL: https://www.raspberrypi.com/ (visitato il giorno 30/06/2026) (cit. a p. 15).

[18] Sebastián Ramírez. *FastAPI*. 2026. URL: https://fastapi.tiangolo.com/ (visitato il giorno 30/06/2026) (cit. a p. 72).

[19] SQLite community. *SQLite*. 2026. URL: https://www.sqlite.org/ (visitato il giorno 30/06/2026) (cit. a p. 14).

[20] Svelte community. *Svelte*. 2026. URL: https://svelte.dev/ (visitato il giorno 30/06/2026) (cit. alle pp. 3, 13).

[21] Vite community. *Vite*. 2026. URL: https://vite.dev/ (visitato il giorno 30/06/2026) (cit. a p. 14).

[22] Vitest community. *Vitest*. 2026. URL: https://vitest.dev/ (visitato il giorno 30/06/2026) (cit. a p. 15).

[23] Wikipedia. *Wikipedia, l'enciclopedia libera*. 2026. URL: https://www.wikipedia.org/ (visitato il giorno 30/06/2026) (cit. a p. 10).