

Documentazione Butler extensions, Inventory & Behaviour

Titolo del progetto: Butler extensions, Inventory & Behaviour
Alunno: Filippo Zinetti
Classe: Info 4AC
Anno scolastico: 2020/2021
Docente responsabile: Fabio Piccioni

1	Introduzione	4
1.1	Informazioni sul progetto	4
1.2	Abstract	4
1.3	Scopo	4
2	Analisi	5
2.1	Analisi del dominio	5
2.2	Analisi e specifica dei requisiti	5
2.3	Use case	7
2.4	Pianificazione	8
2.5	Analisi dei mezzi.....	9
2.5.1	Software	9
2.5.2	Librerie del progetto precedente	10
2.5.3	Nuove librerie	11
2.5.4	Hardware	11
2.6	Nuovi concetti	11
3	Progettazione	12
3.1	Design dell'architettura del sistema	12
3.1.1	Modulo inventario	12
3.1.2	Modulo comportamento	12
3.1.3	Invio dei dati	12
3.1.4	Raccolta delle informazioni di sistema.....	12
3.1.5	Altri dati	14
3.2	Design dei dati e database.....	14
3.2.1	Modello dei dati	15
3.3	Design delle interfacce	16
3.4	Design procedurale	16
3.4.1	UML del server	16
3.4.2	UML del client	18
3.4.3	Modifiche previste	19
3.4.4	Diagramma di sequenza	19
3.4.5	Schema dei componenti	21
4	Implementazione	22
4.1	Linee guida e convenzioni.....	22
4.1.1	Distinzione tra i due progetti	22
4.2	Modifiche al codice di base	23
4.2.1	Errori corretti	23
4.2.2	Nomenclatura.....	23
4.2.3	Autenticazione con il server	23
4.2.4	Gestione dei token da parte del server	24
4.3	Classi comuni	24
4.4	Parte client (Butler).....	25
4.4.1	Inventory	26
4.4.2	Behaviour	27
4.4.3	Butler.....	29
4.4.4	CommandListener.....	30
4.4.5	Messenger	30
4.4.6	Config.json (client)	30
4.5	Parte server.....	31
4.5.1	Manager	32
4.5.2	DbHelper	34
4.5.3	Butler.....	35
4.5.4	ButlerController	35
4.5.5	ControlCenterAPI	35
4.5.6	ButlerAPI	35
4.5.7	Lista e dettagli dei clients "butlerList.html"	36
4.5.8	Aspetto grafico	38
5	Test.....	40
5.1	Protocollo di test.....	40

5.2	Risultati test.....	46
5.3	Mancanze/limitazioni conosciute.....	54
6	Consuntivo.....	55
7	Conclusioni	56
7.1	Sviluppi futuri.....	56
7.2	Considerazioni personali.....	57
8	Glossario	58
9	Bibliografia	60
9.1	Sitografia	60
9.2	Indice delle figure	60
10	Allegati	61
10.1	Fisici.....	61
10.2	In Gitsam	61

1 Introduzione

1.1 Informazioni sul progetto

Il progetto è stato commissionato all'allievo Filippo Zinetti dal docente Fabio Piccioni ed è supervisionato dal perito Antonio Fontana, mentre Claudio Bortoluzzi assisterà alla presentazione come secondo perito. Il responsabile generale dei progetti è Guido Montalbetti.

Si tratta del progetto individuale valido per l'omonima materia come esame finale LPI del quarto anno della sezione SAM informatica, ed avrà una durata di 80 ore reali: dal 3 al 27 maggio 2021. La presentazione del lavoro si terrà il 7 giugno 2021 alle ore 10:30.

1.2 Abstract

To improve evacuation procedures in the event of emergencies like fire ones, the Butler notification system was previously developed. It consists of a server and a client part: from the server, a security operator can define the style of a notification on a web-based control center and send it as a pop-up message to some clients. These are small agent running on multiple PCs, and they are called "Butlers" as they can serve and show notifications to users.

Aiming to also enhance the computer security by having better control over their status, this solution is now extended with two additional modules to monitor the operative system itself. One extension will provide an inventory of the PC's hardware, while the second one will allow to track the system network traffic. The client side self-analysis will be structured in a learning and an actual security analysis phase. The results will then be shown on the server control center for potential further investigations if needed. The administrator will be able to manage the clients thanks to a preexistent and already fully functional software, without any complex additional requirement. The result will be an upgraded version of the same Butler system already developed, therefore a simple update of the two sides' files will be enough to immediately enable all the new functions.

1.3 Scopo

Il sistema Butler è stato realizzato con l'obiettivo di portare informazioni sotto forma di finestre pop-up da un computer centrale agli host connessi sulla rete, ed è composto da due parti: quella server ha un'unica istanza che centralizza la gestione tramite un'interfaccia di controllo web (teoricamente usata da uno o più addetti alla sicurezza specifici), mentre quella client è un semplice agent client detto Butler e, distribuito sui computer degli impiegati, rimane in background in attesa di comandi. Nella versione originale del programma, questi comandi includono principalmente la ricezione e l'interpretazione della grafica delle notifiche da mostrare. Lo scopo principale era quindi quello permettere alle informazioni di raggiungere in modo affidabile gli utenti tramite una connessione sicura, per una comunicazione efficace tra gli impiegati e gli amministratori.

Dato che il programma è stato sviluppato con l'intenzione di essere modulare, è ora richiesto di ampliare la soluzione per un miglior monitoraggio dei client. Sfruttando la struttura di comunicazione già esistente, saranno aggiunte due estensioni: una per raccogliere informazioni sull'hardware dei computer degli utenti (chiamata **inventory**) e una per l'analisi del comportamento (detta **behaviour**), cioè delle connessioni che questi hanno verso l'esterno. Sarà la parte client ad analizzare i suoi dati, che invierà poi intelligentemente al server per essere salvati in un database e mostrati all'amministratore attraverso il centro di controllo.

Il modulo dell'inventario avrà la forma di una semplice lista di dati, mentre quello del comportamento permetterà più interazione tra i programmi e da parte dell'addetto alla sicurezza incaricato: il client, dopo aver definito i collegamenti normali durante una fase di apprendimento (ovvero una raccolta di dati passiva), metterà al corrente il server di qualunque deviazione per permettere all'amministratore di indagare e decidere se impostare ogni connessione come sicura o come sospetta, per poi eventualmente intervenire fisicamente sulla macchina. Questi moduli andranno ad aggiungere funzionalità sia al client che al server in parallelo alla logica già esistente, mettendo quindi alla prova il codice di base nell'accettare nuove funzionalità senza necessitare di ingenti modifiche. Dovrà essere trovata rapidamente una soluzione per ricavare le informazioni necessarie sui PC, e dovrà anche essere implementato in modo efficiente lo scambio dei dati come già accade con il resto della comunicazione. È anche richiesto di considerare la suddivisione del carico di lavoro tra i due programmi evitando che sia una sola parte ad eseguire tutte le operazioni.

2 Analisi

2.1 Analisi del dominio

Per meglio supervisionare i computer in azienda, sarà necessario estendere la soluzione Butler con alcuni componenti aggiuntivi. Il progetto si appoggia sui programmi del sistema “Butler” sviluppati in precedenza, aggiungendo funzionalità ma senza stravolgere la logica: gli utenti client non hanno bisogno di interagire con l'agent Butler, mentre un addetto alla sicurezza disporrà di un'interfaccia web per la gestione delle informazioni e degli host della quale dovrà imparare il funzionamento. A questa interfaccia saranno aggiunti dei controlli per i nuovi moduli. La logica della comunicazione sicura e del funzionamento generale dei programmi è quindi già stabilita e va solo ampliata possibilmente senza stravolgerla.

2.2 Analisi e specifica dei requisiti

I requisiti includono l'aggiunta di due nuovi moduli, che andranno a modificare sia il programma client che quello server insieme all'interfaccia web.

ID: REQ-001	
Nome	Modulo inventory server
Priorità	1
Versione	1.0
Sotto requisiti	
001	Il server dovrà memorizzare l'inventario del computer per ogni client

ID: REQ-002	
Nome	Modulo behaviour server
Priorità	1
Versione	1.1
Sotto requisiti	
001	Il server dovrà memorizzare le informazioni del modello delle connessioni per ogni client
002	Il server dovrà inviare le informazioni aggiornate del modello delle connessioni ad ogni client
003	Dovrà essere possibile gestire il modello standard
004	Dovrà essere possibile applicare il modello standard ai singoli clients

ID: REQ-003	
Nome	Modulo inventory client
Priorità	1
Versione	1.0
Sotto requisiti	
001	Il client dovrà raccogliere informazioni sull'hardware del PC in uso

ID: REQ-004	
Nome	Modulo behaviour client
Priorità	1
Versione	1.1
Sotto requisiti	
001	Il client dovrà raccogliere informazioni sulle connessioni attive
002	Il client dovrà notificare le connessioni che deviano dal modello
003	Il client ha una fase di apprendimento
004	Il client ha una fase di analisi

ID: REQ-005	
Nome	Informazioni inventory nel centro di controllo
Priorità	1
Versione	1.0
Sotto requisiti	
001	Per ogni client devono essere visualizzati i dati dell'inventario

ID: REQ-006	
Nome	Informazioni behaviour nel centro di controllo
Priorità	1
Versione	1.0
Sotto requisiti	
001	Per ogni client devono essere visualizzati i dati del comportamento
002	Dovrà essere possibile mettere in whitelist delle connessioni del modello

ID: REQ-007	
Nome	Attivazione e disattivazione moduli client
Priorità	1
Versione	1.0
Sotto requisiti	
001	Dovrà essere possibile disattivare i singoli moduli per ogni client
002	Dovrà essere possibile attivare i singoli moduli per ogni client

2.3 Use case

Lo schema dei casi d'uso è stato aggiornato per entrambi gli applicativi. Le azioni dello scorso progetto sono rappresentate con sfondo bianco, mentre quelle nuove in giallo. Alcune delle indicazioni "include" e "extend" sono state abbreviate rispettivamente con "inc" e "ext" per risultare meno invasive. Il funzionamento interno degli applicativi e della comunicazione che questi hanno tra di loro è stata limitata al minimo e rappresentata, dove presente, con bordo tratteggiato e sfondo semitrasparente.

Gli impiegati normali avviano in automatico il programma client all'accensione del computer. Alla ricezione di notifiche, possibile solo in seguito a un suo invio e alla precedente autenticazione con il server, possono reagire in vari modi in base al contenuto del messaggio. Sono anche disponibili alcune azioni tramite un menu nella barra delle applicazioni.

L'addetto alla sicurezza (che fa parte degli impiegati) può interagire con il server dal centro di controllo web eseguendo diverse azioni. Oltre alle notifiche, potrà ora gestire anche i clients, i loro dettagli e i loro moduli. È ad esempio possibile cambiare lo stato dei moduli, visualizzarne i dettagli e, in alcuni casi, anche modificarli. In tutti questi casi si tratta di azioni possibili ma non strettamente richieste.

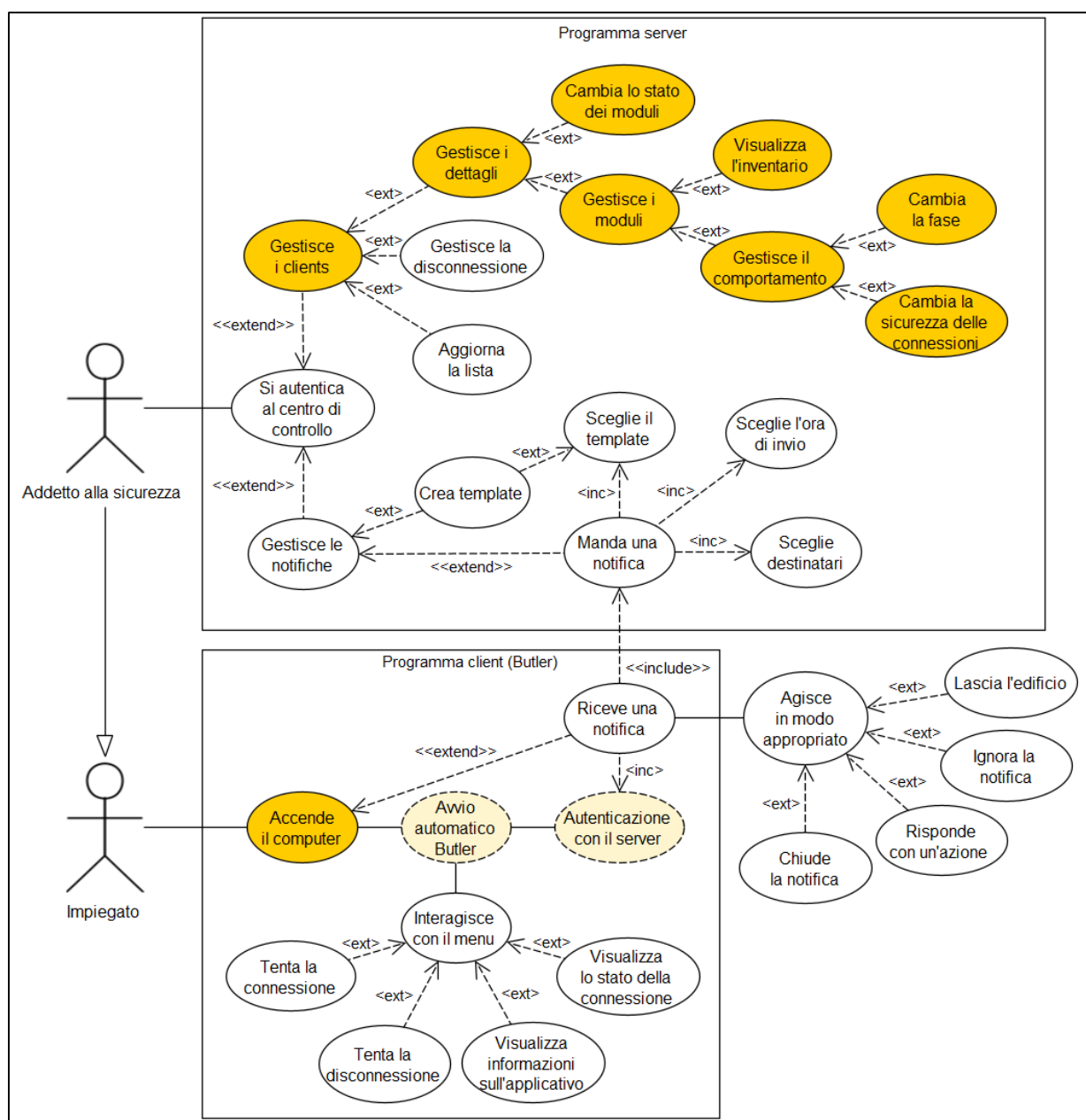


Figura 1 - Lo schema degli use case dei due programmi

2.4 Pianificazione

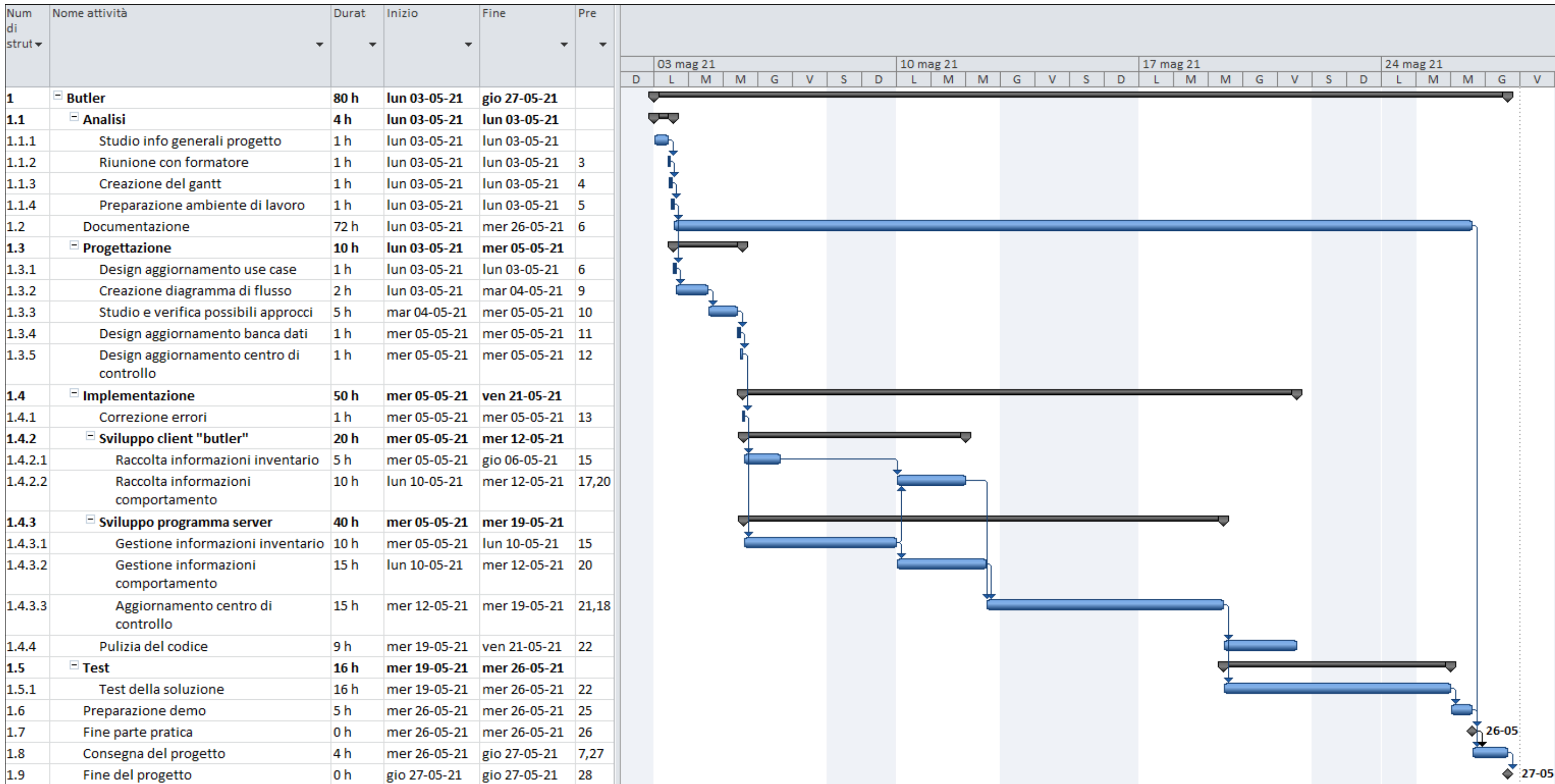


Figura 2 - Il diagramma di gantt preventivo

Il progetto avrà una durata di circa quattro settimane, a partire da inizio maggio. L'approccio scelto per la pianificazione è quello waterfall, cioè le attività sono eseguite una dopo l'altra in modo sequenziale in base al modello standard analisi – progettazione – implementazione – test. Sono stati presi in considerazione le vacanze e gli orari di lavoro, mentre le attività usano blocchi minimo da 1 ora. Le risorse comprendono una sola persona per ogni attività, perciò non è esplicitato nel grafico.

Dopo aver acquisito e chiarito le informazioni sul progetto nell'analisi, la fase di progettazione includerà la scelta del design procedurale e delle modifiche da effettuare ai programmi. Saranno anche creati alcuni schemi per guidare lo sviluppo. L'implementazione include modifiche al client per l'acquisizione dei dati, al server per l'analisi e all'interfaccia grafica per mostrare i risultati, con sviluppo dei moduli client e server in parallelo. I test avverranno in contemporanea con la correzione e la finalizzazione codice. È stato anche pianificato un momento finale per la consegna.

2.5 Analisi dei mezzi

2.5.1 Software

La seguente tabella riassume tutti i programmi utilizzati. I software principali sono **Visual Studio Code** come editor e ovviamente il linguaggio di programmazione **Python**, mentre l'interfaccia grafica è testata su **Microsoft Edge**.

Titolo/nome	Descrizione
Microsoft Edge	Il browser usato per il test del centro di controllo dell'app è Microsoft Edge 87 (la nuova versione basata su Chromium).
Microsoft Office	Molti dei documenti sono creati grazie ai software del pacchetto office di Microsoft: per quelli scritti è usato Word, per la presentazione finale Power Point (tutti e tre alla versione 2019) mentre i gantt sono creati con Project 2010.
Pynsource	È un piccolo strumento per la generazione automatica di grafici delle classi a partire da codice python, che possono poi anche essere leggermente modificati.
Python	L'applicazione è sviluppata in Python 3.7.7, un famoso linguaggio di programmazione open source, dinamico, di alto livello, interpretato e disponibile su diverse piattaforme, che si distingue per la sintassi severa rispetto all'indentazione a favore di una scrittura spesso più breve ¹ .
Sistema operativo della macchina fisica	Il progetto è svolto su un computer fisso Windows 10 Pro versione 2004, build 19041.928
Sistema operativo di test	Per verifica il funzionamento multiplatforma sono usate delle macchine virtuali con Mint 20 Cinnamon 4.6.7 e Windows 10 2004
Virtualbox	Per il creare il network di test è usato VirtualBox 6.1.16 come software di virtualizzazione. È un hypervisor open source con diverse funzionalità.
Visual Studio Code	L'editor usato è Visual Studio Code 1.4/1.5. Questo editor open source e multiplatforma risulta comodo poiché permette il debugging, l'highlighting della sintassi, il completamento intelligente del codice ed è integrato con github.
yEd Graph Editor	Gli schemi (UML, use case, ...) sono creati con yEd 3.19, un software desktop sviluppato da yworks disponibile per ogni piattaforma che permette di creare diagrammi scegliendo tra diversi elementi grafici base, per poi esportare il risultato in vari formati.

¹ <https://www.netguru.com/blog/python-pros-and-cons>, Python Pros and Cons | Netguru Blog on Python, visitato il 19-01-2021

2.5.2 Librerie del progetto precedente

Di seguito sono elencate le librerie più importanti usate dai programmi originali. Quelle python aggiuntive necessarie sono **PySimpleGUI**, **PySimpleGUIWeb**, **PySimpleGUIQt**, **plyer**, **pillow**, **flask**, **requests**, **authlib**, **pycryptodome** e **pymongo**.

Titolo/nome	Descrizione
Authlib	Gestisce i JWT per l'autenticazione e la sicurezza della comunicazione
Bootstrap	Definisce varie classi con stili preimpostati per semplificare l'uso di CSS. È composta da un file di stile e uno di script, che una volta importati rendono disponibile il loro contenuto
Bootstrap-colorpicker	È una libreria che aggiunge dei campi di scelta del colore con alcune opzioni in più e una migliore
Flask	È usato per i server REST, poiché permette di mettere il programma in ascolto su un indirizzo IP e una porta e fornire degli endpoints, cioè degli specifici punti identificati da nomi raggiungibili da chi desidera comunicare con il server
Fontawesome	Comprende varie centinaia di icone da poter essere usate nelle pagine web
HTML, CSS, JavaScript	Per lo sviluppo delle interfacce web sono usati i tre strumenti standard per definire struttura (HTML), stile (CSS) ed elementi interattivi (JavaScript)
jQuery	È una libreria javascript che semplifica la gestione degli elementi e dell'interazione nelle pagine web
PyCryptoDome	Permette di creare delle chiavi di sicurezza, che in questo caso andranno a crittare i token in modo asimmetrico
PySimpleGUI	Questa libreria unifica i più famosi gestori di interfacce grafica in python semplificandone notevolmente l'uso e rendendoli completamente intercambiabili. La versione base usa tkinter, ma esistono anche PySimpleGUIWx che si basa su Wx, PySimpleGUIQt basato su Qt e PySimpleGUIWeb su Remi. Essendo in costante sviluppo, alcune delle funzionalità non sono presenti o completamente funzionanti in tutte le varianti: è per questo che nel progetto è necessaria, oltre a quella tkinter, anche la versione Qt. Quella basata su remi porta la stessa interfaccia della pagina desktop anche sul browser e, nonostante sia ancora in beta, sarà usata per mostrare le notifiche anche sull'interfaccia web
Requests	Permette di eseguire richieste HTTP con vari parametri. È usato per comunicare con i server REST
Split.js	Si tratta di una piccola libreria per trasformare delle colonne HTML in elementi dalle dimensioni variabili, permettendo all'utente di cambiare a piacimento altezza e larghezza di alcune sezioni ridimensionandole in modo intuitivo

2.5.3 Nuove librerie

Le seguenti sono invece le librerie usate insieme ai nuovi moduli. L'unica nuova dipendenza è **psutil** per il client.

Titolo/nome	Descrizione
PSUtil	Python System and process utility è una libreria multiplatforma per l'acquisizione di informazioni su processi e sull'utilizzo del sistema. Permette di monitorare e gestire le risorse di sistema emulando diversi altri comandi come ifconfig, netstat, ps, top, free, who,... La maggior parte delle informazioni sul sistema e sul computer in generale potranno essere ricavate grazie a quest'unica libreria.
pdoc3	È una libreria usata come eseguibile a sé state, che genera una documentazione in HTML in base ai commenti di un modulo python. È la versione aggiornata di pdoc. Il comando va avviato dall'interno della cartella butler/src, ed è: <code>pdoc3.exe --html -o ../doc</code>

2.5.4 Hardware

L'hardware richiesto per il progetto consiste in un singolo computer; i vari host sono simulati attraverso macchine virtuali.

Titolo/nome	Descrizione
PC fisso scolastico	È disponibile un computer fisso offerto dalla scuola, che dispone di 32GB di RAM e una CPU da 8 core e 8 processori logici. Lo spazio disco offerto è di circa 400GB.

2.6 Nuovi concetti

Nello scorso progetto, al centro del programma c'erano le notifiche, cioè messaggi personalizzabili salvati come stringhe JSON. Queste sono ora messe da parte, e la soluzione è resa più generica: verranno introdotti i moduli **inventory** e **behaviour** che saranno affiancati a **notification**. Non sarà possibile assegnare ad ognuno di questi una classe apposita, dato che le notifiche devono per forza esistere nel loop principale del programma, ma si tratta comunque di plug-in piuttosto indipendenti tra loro.

Il modulo di comportamento contiene un **modello delle connessioni**, ovvero una lista delle connessioni di rete stabilite o in ascolto del computer locale. Questi dati serviranno poi agli amministratori per individuare possibile traffico al di fuori dei compiti standard del PC. Il modello viene aggiornato in modi differenti a seconda di due fasi: una di **apprendimento** e una di **analisi**. Nella prima, ogni nuova connessione trovata o fornita viene aggiunta al modello tra quelle sicure dato che il computer sta definendo il suo funzionamento normale. Stabilito il modello si passa alla fase di analisi, nella quale ogni Butler confronterà il suo modello con le connessioni attive, stabilendo intelligentemente quali siano sospette.

3 Progettazione

3.1 Design dell'architettura del sistema

Sono ora spiegati i ragionamenti per la progettazione dell'architettura dei programmi, come si intende organizzare lo sviluppo e i punti più importanti considerati. Saranno sviluppate due estensioni principali, ma ci sono altri componenti che necessitano di un approfondimento ulteriore.

3.1.1 Modulo inventario

Si tratta di un'estensione dal funzionamento piuttosto lineare, il quale scopo consiste nel **ricavare informazioni generali** sull'hardware sul computer in uso. Una volta ricevute dal client, il server le salverà nel database.

3.1.2 Modulo comportamento

A differenza dell'inventario, quest'estensione richiede uno **scambio di dati maggiore** tra client e server: è quest'ultimo che cambierà la fase e che si occuperà di fornire i dati corretti al momento giusto. Già all'autenticazione, il server dovrà impostare al client la fase ed eventualmente allegare il modello delle connessioni precedente.

3.1.3 Invio dei dati

La connessione, essendo full-duplex, permette al client di inviare periodicamente le sue informazioni al server, in modo che il database sia costantemente aggiornato. È un comportamento utile, ma sarà aggiunta la possibilità di disattivarlo tramite un parametro della configurazione. Il centro di controllo web invece, comunicando **unidirezionalmente** solo verso il server, non è progettato per essere aggiornato in tempo reale con le nuove informazioni. Come succede nel resto dell'interfaccia, dunque, i dati verranno **caricati all'occorrenza** automaticamente tramite AJAX, ma non si aggiorneranno tramite polling. In breve, l'utente richiederà involontariamente i dati durante il normale uso dell'applicativo, ma una volta mostrati non cambieranno senza interazione.

Al cambio dello stato dei moduli, il client smetterà o riprenderà a gestire alcuni tipi di dati, decidendo se inviarli o meno. È previsto di utilizzare una thread **sempre attiva**, la quale logica potrà essere disattivata controllando lo stato della connessione al server, ma senza interrompere il ciclo di controllo.

3.1.4 Raccolta delle informazioni di sistema

Tra le diverse librerie python che permettono di ricavare singole informazioni sull'hardware e il software in uso (ad esempio l'utente con `getpass`, la GPU con `wmi2`, ...), **psutil** dà accesso a molti dati contemporaneamente³, perciò sarà usato come fonte principale. Permette di leggere informazioni sia su gran parte dell'hardware, sul suo uso corrente e sull'uso medio, sia sulle connessioni di rete attive stabilite e in ascolto. È sufficiente importare la libreria per avere accesso a tutti i suoi metodi. Come descritto nella documentazione ufficiale⁴, però, alcune funzioni come la velocità delle ventole (`psutil.fan_speed()`) non sono totalmente multiplatforma. Sarà dunque necessario prestare attenzione a mantenere i software simili, evitando l'uso di questi dati.

² <https://stackoverflow.com/questions/38103690/get-system-informationcpu-speed-total-ram-graphic-card-model-etc-under-window>, python - Get system information(CPU speed-Total RAM-Graphic Card model etc.) under Windows - Stack Overflow, visitato il 04-05-2021

³ <https://stackoverflow.com/questions/3993731/how-do-i-access-netstat-data-in-python>, How do I access netstat data in Python? - Stack Overflow, visitato il 04-05-2021

⁴ <https://psutil.readthedocs.io/en/latest/>, psutil documentation — psutil 5.8.1 documentation, visitato il 04-05-2021

3.1.4.1 Informazioni sulle connessioni di rete

Grazie alla funzione `psutil.net_connections()` è possibile ricavare informazioni sulle connessioni di rete. Poiché queste sono simili a quelle di **netstat** (probabilmente internamente viene proprio usato questo comando), gli stati delle connessioni sono gli stessi. Non tutti sono utili ad un'analisi: ad esempio, quelle in stato `TIME_WAIT` rappresentano connessioni terminate ma in attesa della chiusura completa del socket da parte dell'utente (ad esempio la chiusura del browser). Queste in particolare sono molto "instabili", poiché ne possono essere generate diverse in poco tempo che poi scompaiono rapidamente allo stesso modo andando ad espandere il database con dati poco rilevanti. Sarà quindi implementata, nel file di configurazione, una lista degli stati delle **connessioni da ignorare**.

3.1.4.2 Struttura delle connessioni

Una connessione comprende diversi dati:

- PID
- Nome del processo
- Sorgente (ip e porta)
- Destinazione (non sempre presente; ip e porta)
- Stato

Purtroppo, nessuno di questi permette di riconoscere con sicurezza una connessione da un'altra: il PID varia ad ogni avvio del programma, la porta sorgente è spesso scelta a caso tra quelle dinamiche (49152-65535) e non è quindi affidabile, la destinazione non sempre è presente, ...

Per distinguerle, dunque, sarà **necessario considerare anche lo stato** per poter decidere come trattarle: ad esempio, quelle in stato `LISTEN` non avranno una destinazione e bisognerà considerare anche la porta sorgente oltre che l'indirizzo. Per quelle `ESTABLISHED`, invece, potrebbe essere utile considerare anche l'indirizzo sorgente (ma non la porta) oltre che ovviamente indirizzo e porta di destinazione. Non può però nemmeno essere usata la stringa di stato poiché impedirebbe di distinguere le connessioni in stato `CLOSE_WAIT` e simili. Di seguito è stato schematizzato l'UML del controllo che andrà effettuato.

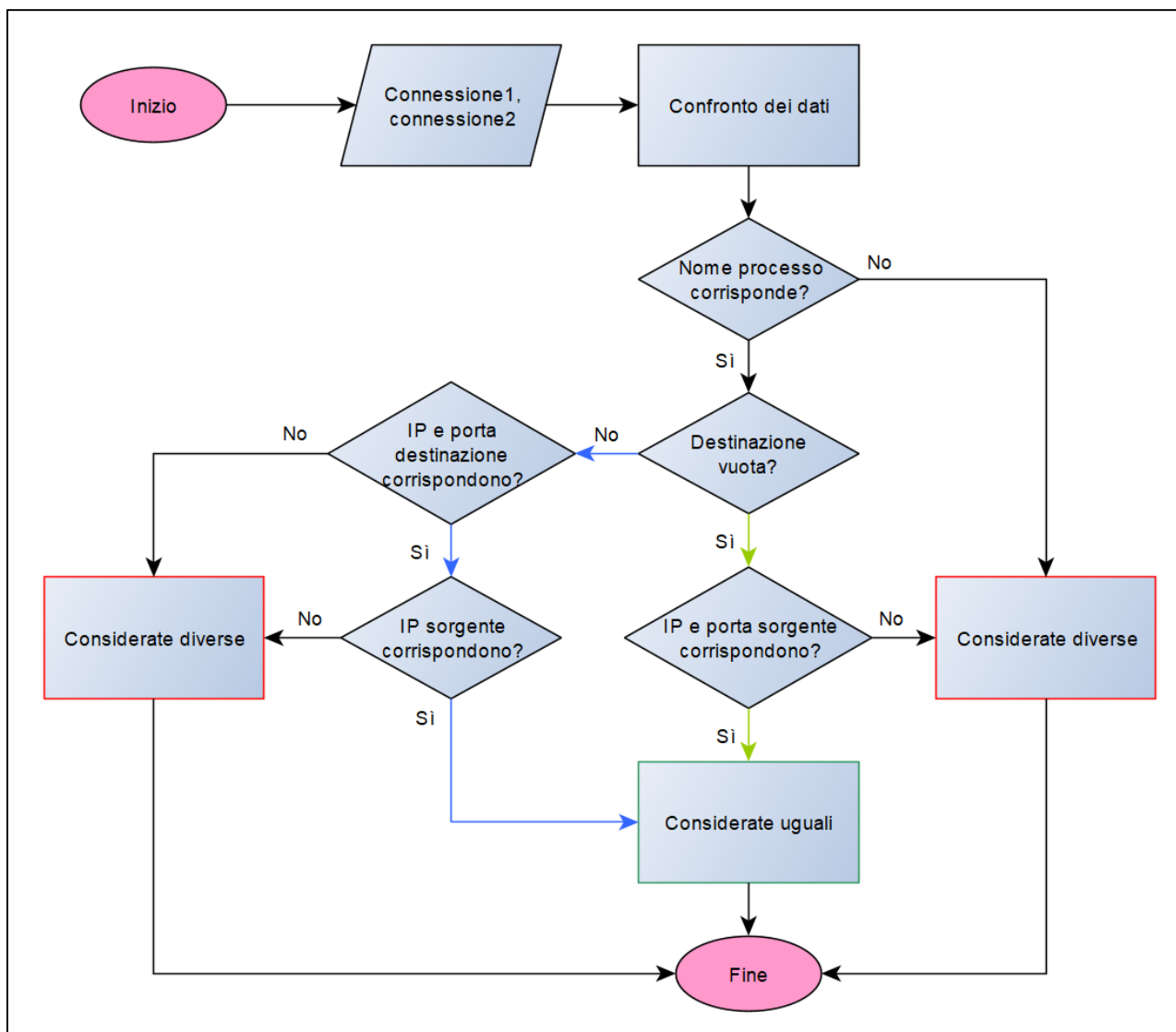


Figura 3 - Il confronto delle connessioni in dettaglio

3.1.5 Altri dati

Nonostante i diversi parametri raccolti da psutil, alcune informazioni vanno trovate in altri modi. Ad esempio, per trovare il modello della CPU e la build di sistema sarà usato **processor**, il quale funzionamento è simile a psutil (espone una serie di funzioni che ritornano vari parametri).

3.2 Design dei dati e database

I dati di entrambe le estensioni vanno salvati su un database, dunque è necessario legarli ad un identificativo: quelli considerati sono l'attuale **ip:porta**, l'**hostname** e il **MAC Address**. La soluzione attuale non era progettata per una distinzione assoluta e infallibile dei Butlers. È stata scartata, più che le collisioni/duplicazioni, per il problema della possibile dinamicità degli indirizzi. L'hostname sarebbe stato una soluzione nel caso si fosse disposto unicamente di hosts Windows che, collegati ad un'active directory, non permettono di impostare un nome uguale. In Linux, però, non viene posto alcun blocco, portando alla possibilità di avere macchine dallo stesso nome. L'**indirizzo MAC** sembra quindi la scelta più appropriata. Malgrado possa essere alterato in vari modi, tutto sommato si tratta di un'operazione poco accessibile ed è quindi scelto per distinguere le informazioni dei computer sul database.

3.2.1 Modello dei dati

Come nel precedente progetto, sarà usato il database MongoDB. È per questo che lo schema di seguito non è definibile ER e non ci sono relazioni. Non sono rappresentate le informazioni del database originale, dato che non hanno alcun collegamento con la nuova tabella.

Il database da rappresentare comprende, di base, unicamente i computers, poi divisi in altri documenti.

Per le due nuove estensioni è stata progettata un'unica nuova collection. Si tratta di **computer**, ufficialmente identificato dall' **_id** generato automaticamente, ma che nel programma sarà riconosciuto grazie a **mac** (il MAC Address). Rappresenta le macchine dei clients, ai quali sono poi legate altre informazioni: un inventario, lo stato dei moduli, una lista di connessioni e la fase del comportamento.

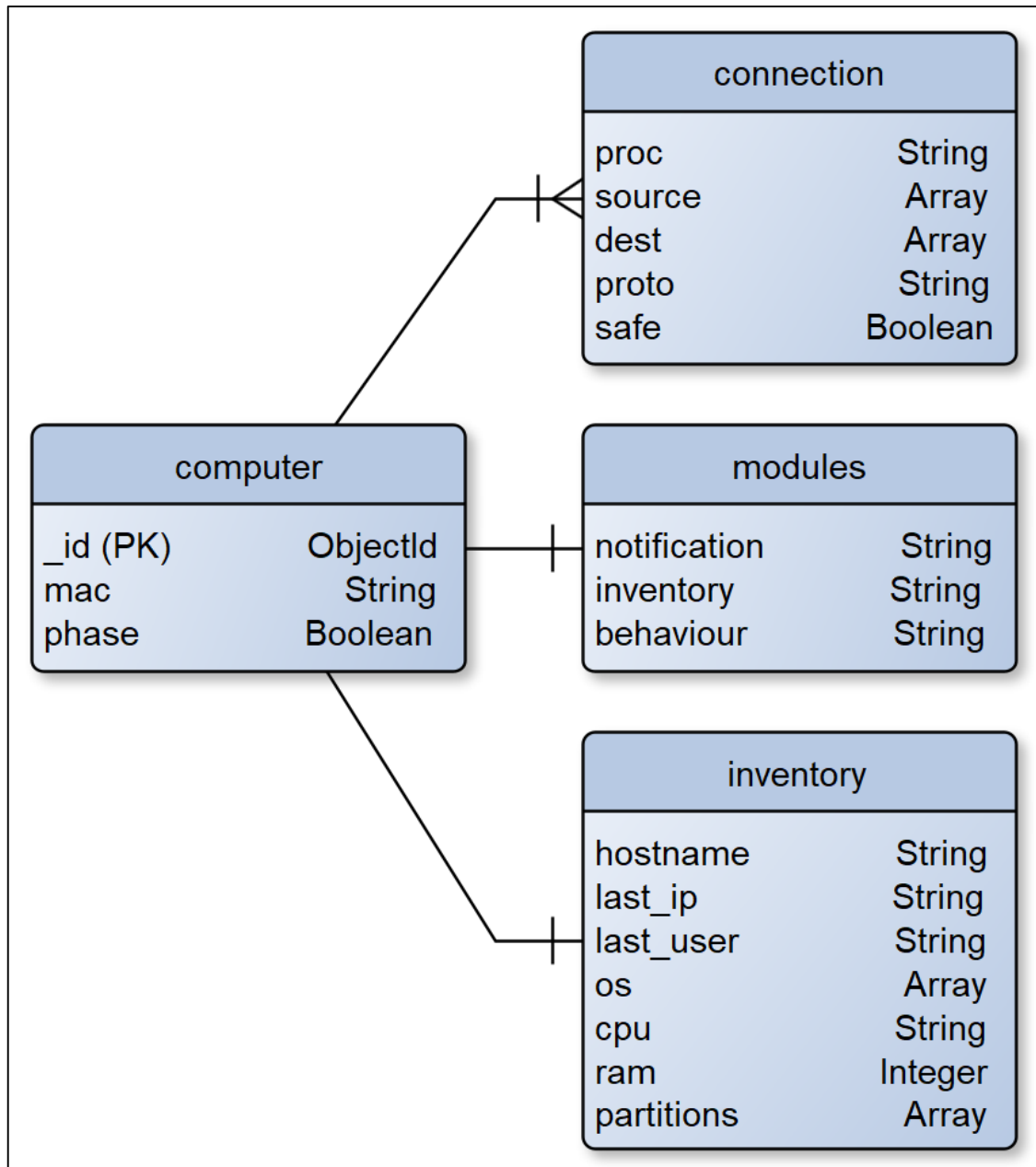


Figura 4 - Il modello dei dati

La libreria psutil permetterebbe di raccogliere tantissime informazioni sul computer, ma sono richieste solo le più basilari per semplificare poi la rappresentazione grafica.

3.3 Design delle interfacce

L'interfaccia attuale consiste in un'unica pagina, composta da colonne verticali definite **sezioni**, ognuna delle quali contiene una piccola interfaccia a sé stante che assolve un compito preciso. Il contenuto di ogni sezione è caricato in modo dinamico ed è facilmente possibile aggiungerne altre senza il bisogno di alcun cambiamento alle altre. Le modifiche grafiche previste, però, vedono aggiunte solo alla già presente sezione della lista dei Butlers connessi. Dato che i moduli aggiuntivi aumentano le informazioni ed il controllo sui clients, l'opzione più pratica individuata è l'aggiunta di un elemento **modal**, che appare sopra alla lista e, all'apertura, carica i controlli per l'host selezionato. Altre possibili soluzioni considerate sono la creazione di una nuova sezione (che sarebbe risultata superflua) e l'aggiunta delle informazioni in un contenitore a singola istanza direttamente sotto alla lista piuttosto che in sovrapposizione (quando non usati, però, i dati avrebbero unicamente sottratto spazio di lavoro).

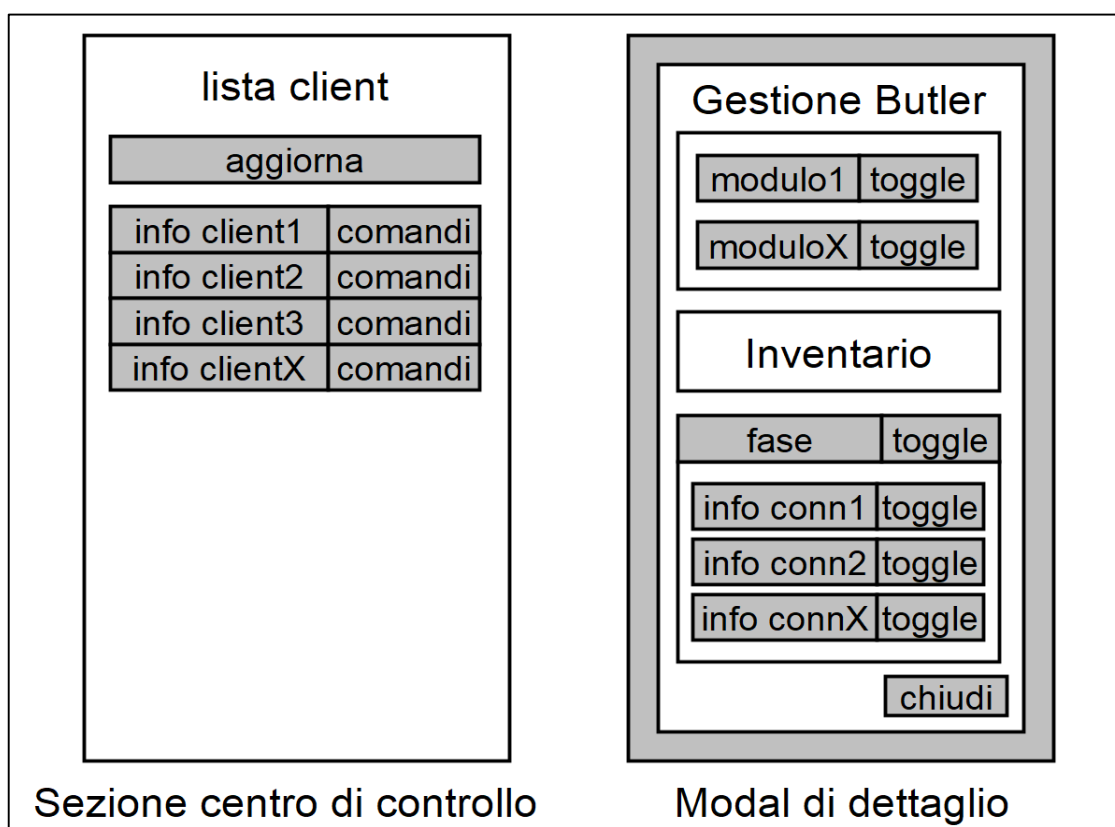


Figura 5 - Il mockup dell'interfaccia

A sinistra è rappresentata l'interfaccia già presente, isolata dal resto del centro di controllo, mentre a destra si vede il **modal da aggiungere** che gli apparirà sopra. Alla sezione di base andrà aggiunto un comando per aprire la visualizzazione dei dettagli. Questi comprenderanno la lista dei moduli del client disponibili (con lo stato modificabile da dei toggle switch), l'inventario dei componenti e la gestione del comportamento. Quest'ultima parte è composta, a sua volta, da un toggle per la fase (apprendimento o analisi) e la lista di tutte le connessioni attive: anche qui è possibile cambiare lo stato, cioè metterle nella whitelist o tra quelle sospette.

3.4 Design procedurale

3.4.1 UML del server

Nel diagramma di flusso del server è stata semplificata la parte di invio delle notifiche, ed è stato aggiunto il funzionamento interno dei nuovi componenti. La scelta delle azioni disponibili è duplicata per migliorare l'impatto grafico e comprende, appunto, la procedura per l'invio di notifiche dello scorso progetto e la nuova parte di gestione del comportamento dei client. Di questi è possibile cambiare lo stato dei vari moduli, cambiare la fase di comportamento e modificare l'attendibilità delle varie connessioni.

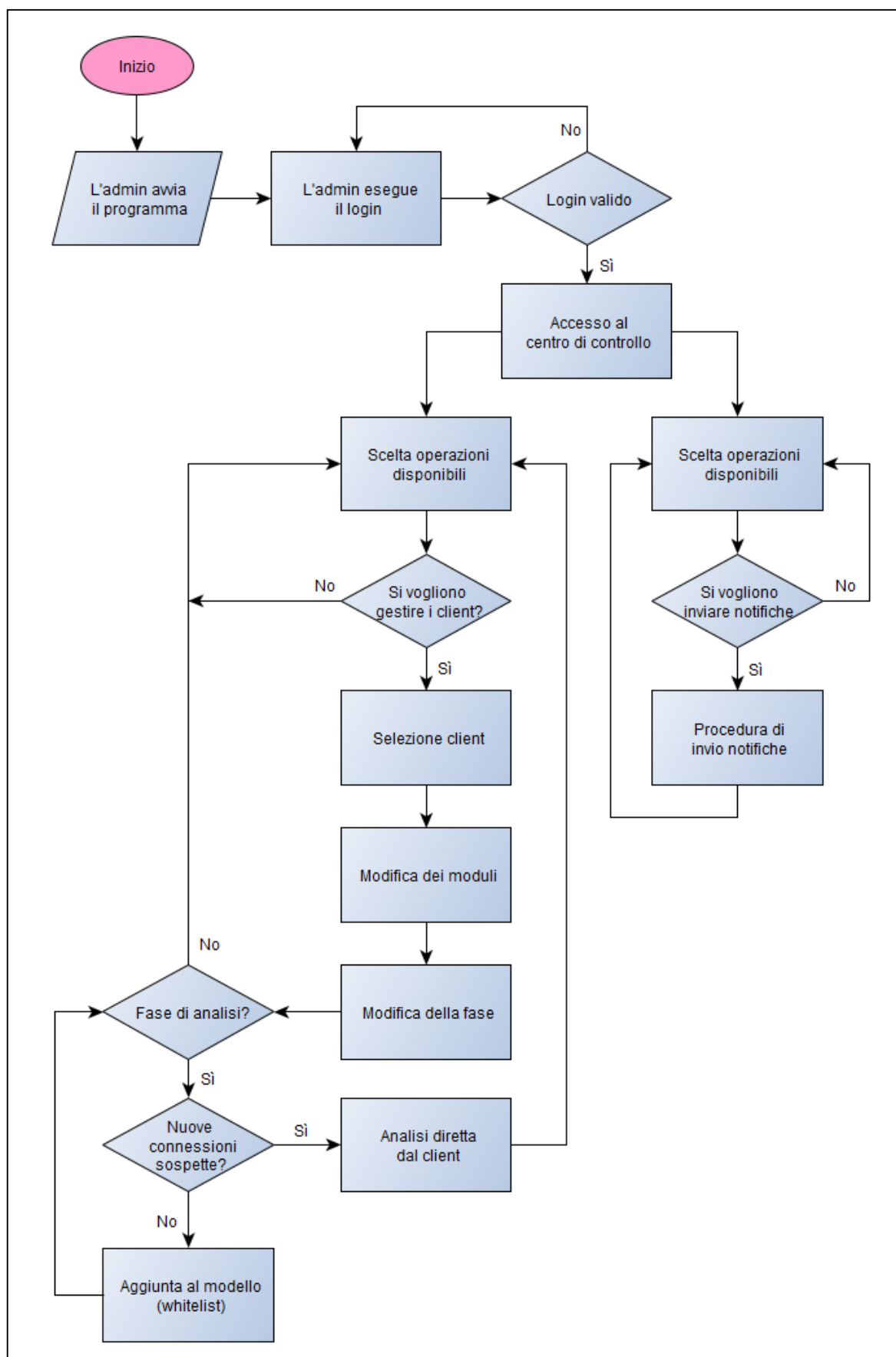


Figura 6 - L'UML del programma server

3.4.2 UML del client

Il flusso del client è completamente automatizzato. Tutti i moduli eseguono le loro operazioni **solo quando attivi**, ma il complesso funzionamento di ognuno è stato semplificato a poche azioni.

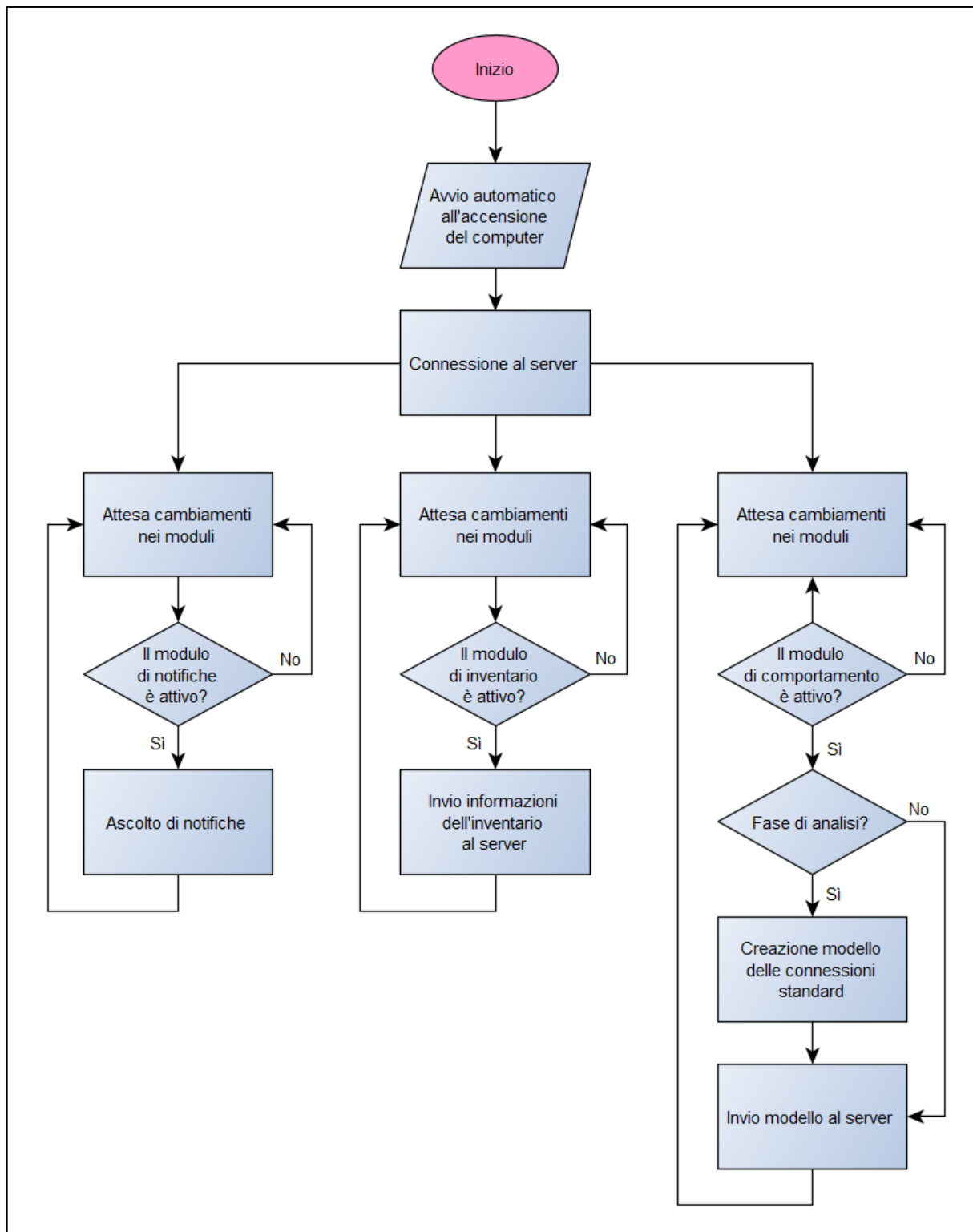


Figura 7 - L'UML del programma client

3.4.3 Modifiche previste

I software sono composti da vari livelli di “profondità” e specializzazione, che vanno dalla gestione della GUI al centro dello scambio di dati tra i componenti interni. Per permettere a nuovi dati di essere trasferiti sarà necessario **modificare varie classi**. Ad esempio, andranno aggiornate quelle che espongono i server RESTful (ButlerAPI, ControlCenterAPI,...) con ulteriori endpoint per gestire i nuovi input. Dato il requisito di poter disattivare i singoli moduli, è stata considerata l'opzione di aggiungere alla parte client una nuova classe di comunicazione ed una porta apposita per ognuno. Alla fine, considerando che nessuno dei moduli avrebbe richiesto molti scambi di dati, la complessità di disattivare dei server flask (che avrebbe richiesto un cambio della gestione di tutte le threads) e le numerose modifiche che sarebbero state necessarie al server, è stato preferito lasciare il traffico unito in un unico “ascoltatore”. Al server andranno anche aggiunte alcune funzioni per salvare e recuperare i dati dal database, e sarà necessario modificare anche una singola pagina per la grafica del centro di controllo.

Non sono previsti cambiamenti ai file della cartella **common**, poiché questi contengono le basi per altre classi e dovrebbero già essere pronti per qualsiasi modifica a quelle che le estendono. Infine, le classi che ora faranno riferimento alle notifiche (centro dello scorso progetto), necessiteranno un cambio di nome per rappresentare meglio le nuove funzioni.

3.4.4 Diagramma di sequenza

Il seguente schema rappresenta gli scambi di dati tra i due programmi. È diviso in GUI (dove necessario), backend e parte REST. Sono mostrati solo i flussi di informazioni che subiranno modifiche. Alcuni dati che partono dall'interfaccia del server sembra siano inviati direttamente ai clients poiché non è necessario eseguire altre operazioni in mezzo. In realtà, la richiesta attraversa comunque ogni componente: inviarla direttamente ad un client senza passare dal server non sarebbe stata una soluzione **né conveniente, né pratica** a causa dell'autenticazione e della struttura dei programmi.

Butler extensions, Inventory & Behaviour

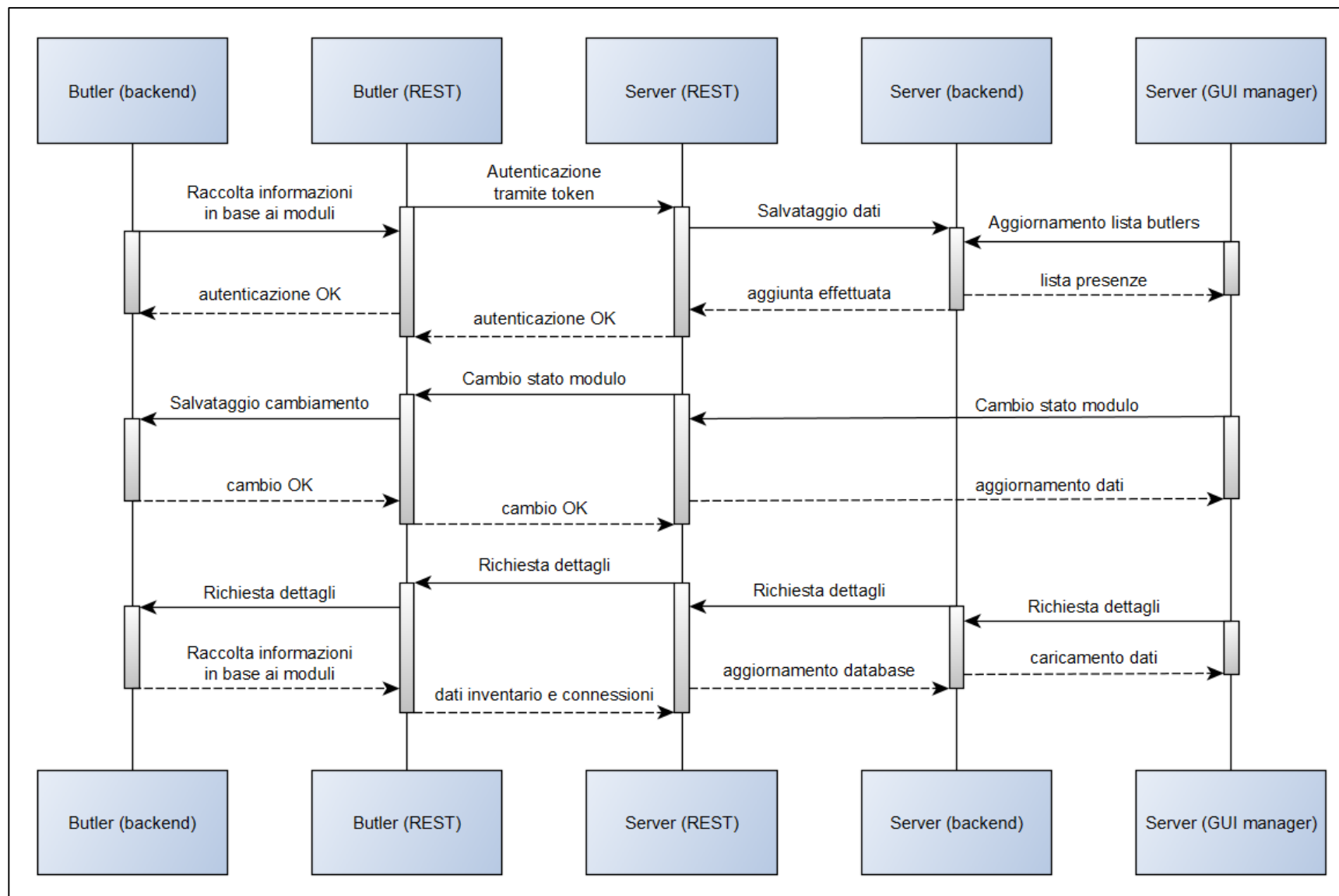


Figura 8 - Il diagramma di sequenza

3.4.5 Schema dei componenti

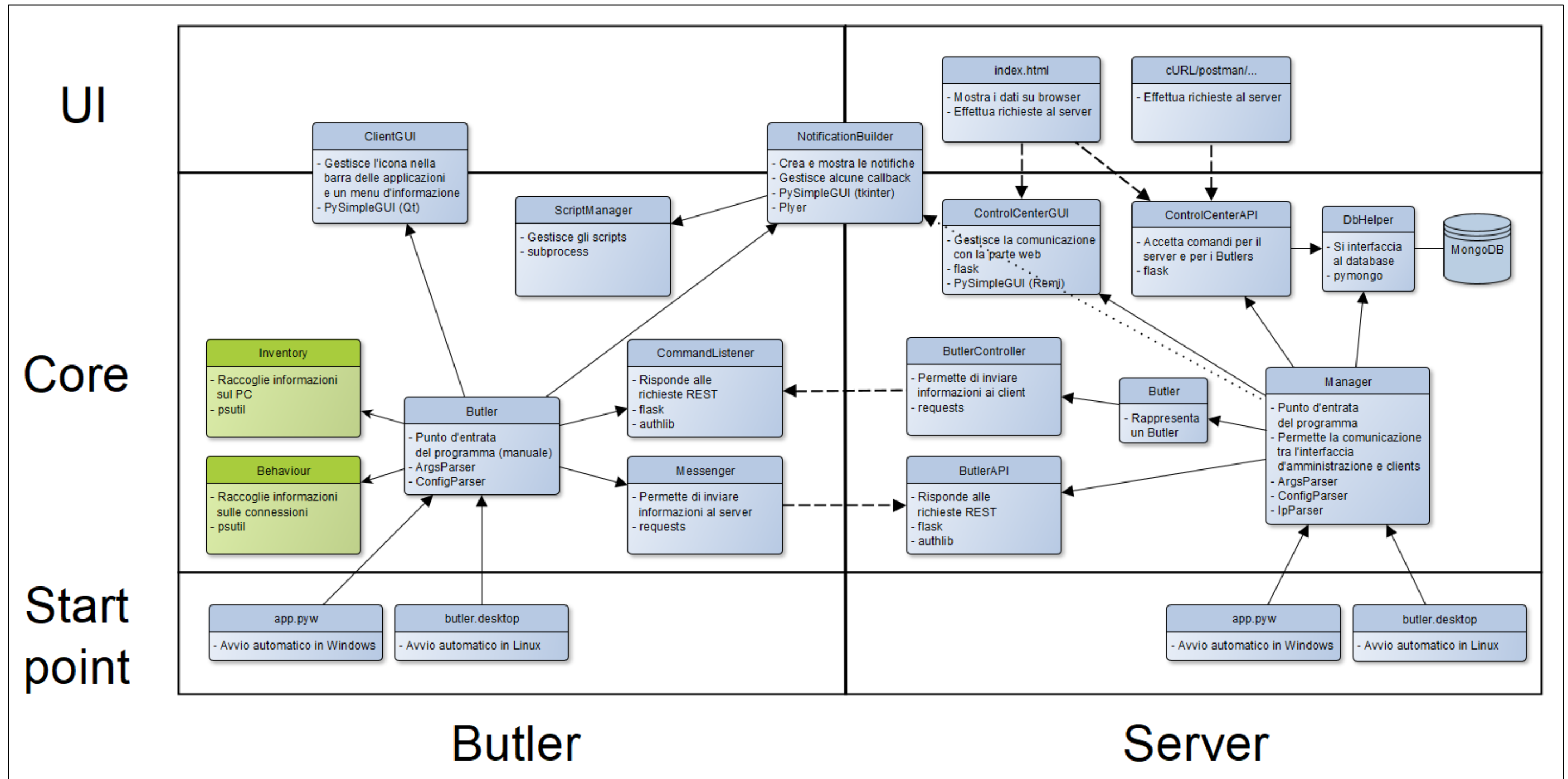


Figura 9 - Lo schema dei componenti

Il diagramma dei componenti è rimasto simile a quello dello scorso progetto. In orizzontale sono rappresentati i due programmi, mentre in verticale questi sono stati divisi in punto d'avvio, logica principale e interfaccia utente. Non sono rappresentate tutte le classi, ma solo quelle principali. Sono stati aggiunti i due moduli di raccolta dei dati dalla parte client (rappresentati in verde), mentre per il server si prevedono unicamente nuove funzioni, non presenti nello schema.

4 Implementazione

4.1 Linee guida e convenzioni

Sono di seguito spiegati in modo dettagliato il funzionamento delle aggiunte e le motivazioni delle scelte implementate, sostenute da parti specifiche di codice (leggermente adattate dove necessario) che illustrano unicamente i passaggi più importanti del software. È seguita la struttura della precedente documentazione ma, dato il minore quantitativo di nuovo codice, questo è descritto più nello specifico.

L'implementazione non ha previsto stravolgimenti del codice, quindi sono ancora valide tutte le scelte generali adottate per lo scorso progetto che includono lo stile di codifica, la lingua del programma e dell'output, la logica generale e la forma finale del prodotto.

4.1.1 Distinzione tra i due progetti

Dato che il codice di base è rimasto perlopiù invariato, è stato scelto di inserire i seguenti specifici commenti tra i blocchi di codice per rendere chiara la distinzione tra il lavoro dei progetti:

```
"""
#####
Funzioni aggiuntive butler-extensions
#####
"""

def ...():
    ...

"""
#####
Fine funzioni aggiuntive butler-extensions
#####
"""
```

Solitamente, questi tag sono posizionati alla fine dei file. Per mantenere intatto il funzionamento del codice, in alcuni casi è stato necessario inserirli più volte all'interno della stessa classe.

Le altre modifiche di qualsiasi genere effettuate alle funzioni già esistenti sono invece qui descritte, ma non hanno elementi identificativi all'interno del codice.

4.2 Modifiche al codice di base

Prima di applicare modifiche al programma, è stato assicurato il completo funzionamento del codice di base. Sono ora descritti anche alcuni errori sorti durante lo sviluppo delle estensioni, ma che riguardano la logica pensata precedentemente.

4.2.1 Errori corretti

Nel codice di base non sono stati rilevati importanti errori, ma solo due imprecisioni minori: per quanto riguarda il centro di controllo, la gestione di due eventi javascript non era stata adattata al resto del codice. Questo portava a malfunzionamenti nel forzare la disconnessione dei client in alcuni casi. È stato risolto cambiando l'handler di questi eventi. In precedenza era il seguente:

```
$("<id>").on('click', function () { ... });
```

Come eseguito nel resto del codice, invece, per permettere al click di essere riconosciuto anche dopo aver modificato la finestra è bastato cambiarlo in questo modo:

```
$("body").on('click', "<id>", function () { ... });
```

Il secondo “errore” vedeva l'uso dell'algoritmo di hashing MD5, apparentemente non più sicuro⁵, dunque ne vanno preferiti altri come SHA256, che è poi anche quello scelto. È bastato cambiare la funzione della libreria hashlib da .md5() a .sha256().

Sono stati trovati e risolti anche alcuni piccoli errori grammaticali e di stile nei commenti.

4.2.2 Nomenclatura

Alcune classi includevano nel nome dei riferimenti alle notifiche e, per coerenza, hanno subito un cambio di nome:

Programma	Nome precedente	Nuovo nome	Motivo
Client	NotificationListener	CommandListener	Come per Notifier, ora ascolta vari comandi e non più principalmente le notifiche
Server	Notifier	ButlerController	In precedenza era usato principalmente per inviare le notifiche, ma ora permette di inviare diversi altri comandi ai Butlers

4.2.3 Autenticazione con il server

È stata constatata una strana lentezza nell'attivazione dei server flask che devono legarsi a indirizzi di rete diversi da 127.0.0.1. Questo portava ad un'autenticazione client-server solo parzialmente completa, con il client convinto di essere connesso ma il server impossibilitato ad inviare lui qualunque istruzione (un errore mai riscontrato durante l'implementazione originale dei programmi). La procedura vedeva inizialmente il client che si connette al server e, in maniera asincrona subito dopo, questo usa un altro canale per connettersi al client. È stato necessario collegare questi due passaggi: il server confermerà di essere pronto solo quando avrà terminato la sua parte di autenticazione verso il client, rendendo le due parti della comunicazione più dipendenti tra loro.

```
status = self.ACCEPTED if self.callbacks['add_butler'](mac, addr, user)
else self.BAD_METHOD
return self.standard_response(returnData, status)
```

⁵ <https://www.section.io/engineering-education/what-is-md5/>, *What Is MD5 and Why Is It Considered Insecure?* | Section, visitato il 07-05-2021

4.2.4 Gestione dei token da parte del server

È stato rilevato un comportamento diverso rispetto alla logica prevista che vedeva un continuo aggiornamento dei tokens JWT al posto di riutilizzare quelli già stabiliti. Grazie alla gestione degli errori, questo malfunzionamento riusciva a passare inosservato non bloccando i programmi, ma solo rallentandoli leggermente. Più nello specifico, il server era obbligato a ristabilire nuovamente la connessione con ogni client (tranne l'ultimo connesso) prima di ogni richiesta, poiché il token da usare veniva sovrascritto alla connessione di altri Butlers. La perdita dei token era già stata considerata e affrontata nello scorso progetto, creando un ButlerController (l'incaricato di inviare le richieste da parte del server) per ogni client connesso, ognuno con il suo token personale. Nonostante il salvataggio dei tokens sia rimasto invariato, il programma sembrava unire tutti i RequestSubmitter (la classe dalla quale ButlerController eredita, e anche quella che memorizza effettivamente i tokens) in uno solo, seppur fossero assegnati a oggetti, appunto, ben distinti tra loro. In questo modo, i token venivano messi in condivisione e di conseguenza solo uno alla volta poteva essere memorizzato; un problema alquanto improbabile dati gli accorgimenti appositi presi. È stato necessario applicare due correzioni, entrambe essenziali per evitare la perdita di tokens:

- all'aggiunta di un nuovo Butler, dell'oggetto viene effettuata una deepcopy con il comando `copy.deepcopy(...)` in modo da escludere l'uso di qualunque riferimento ed assicurare la copia completa
- al metodo `super()` di RequestSubmitter, richiamato da ButlerController, viene passato l'attributo `header` (che contiene anche il token), in modo che sia più legato alla classe figlio

4.2.4.1 Pulizia della lista di Butlers

A causa del controllo effettuato durante l'acquisizione di ogni dato dei client, in nessun punto del programma server era necessario ripulire la lista dagli host disconnessi. Nonostante non abbia mai causato problemi, sono ora stati aggiunti due punti nella funzione `check_butlers()` che rimuovono coloro che non rispondono alla richiesta di stato, eseguendo:

```
del self.butlers[addr]
```

4.3 Classi comuni

Non è stato necessario modificare quasi nessuna di queste classi in modo sostanziale, dimostrando che la loro astrazione era sufficientemente elevata da poter rimanere invariante nonostante le modifiche al resto del codice. Gli unici piccoli cambiamenti riguardano FailedRequest, al quale è stata aggiunta un'istruzione di log per notificare esattamente l'errore di connessione riscontrato, e RequestSubmitter che ora riceve anche `headers` come parametro opzionale nel costruttore.

4.4 Parte client (Butler)

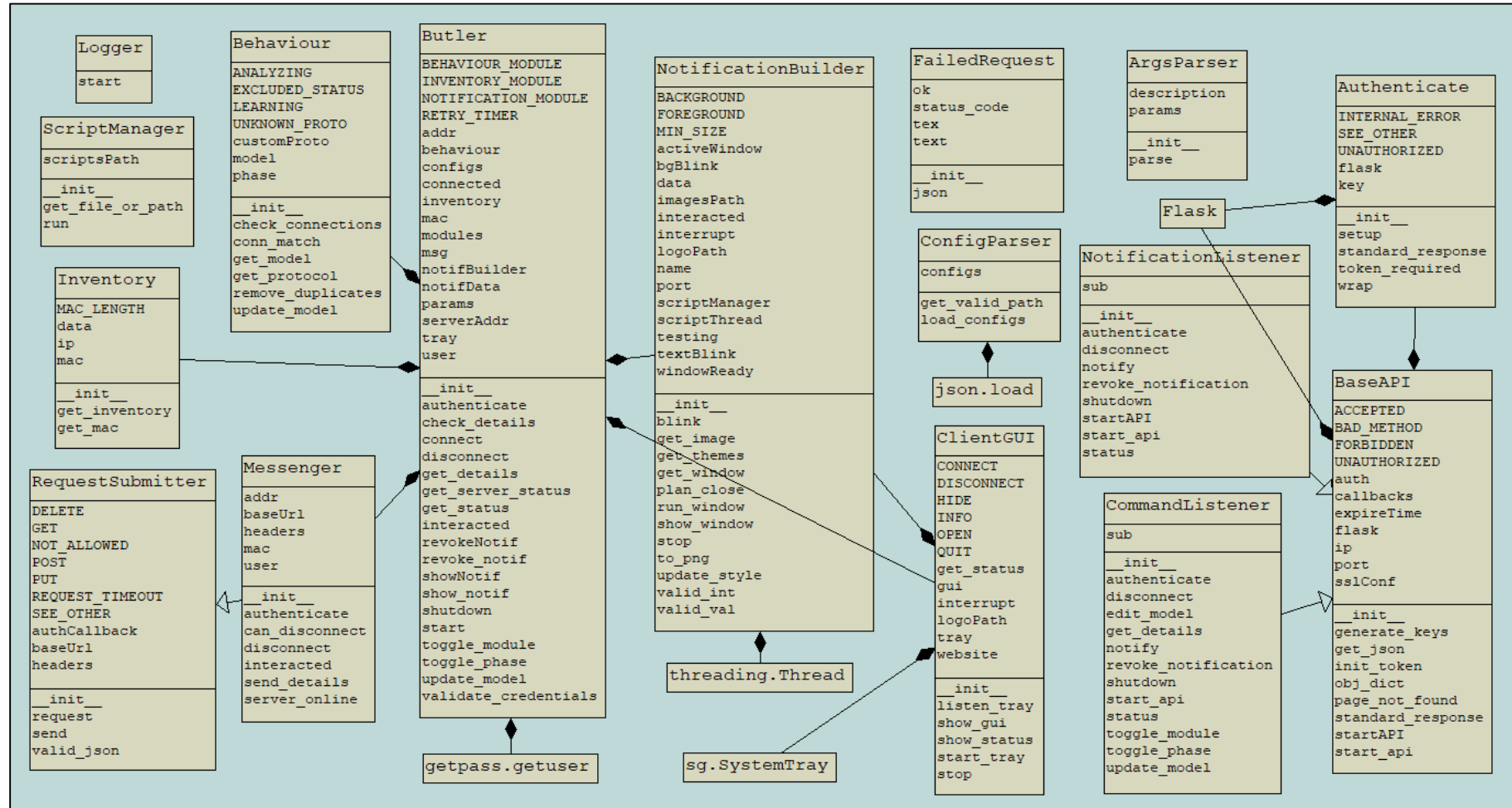


Figura 10 - Il diagramma delle classi del programma client

Al client sono state aggiunte due classi corrispondenti ai due moduli; il resto delle modifiche sono cambiamenti minori. Lo schema è stato generato con pynsource.

4.4.1 Inventory

Questa classe esegue un inventario dei componenti del computer. Contiene una funzione per ricavare l'indirizzo MAC dall'ip (assegnata a questa classe poiché coerente con lo scopo), e una per eseguire effettivamente l'inventario. Quest'ultima, nonostante sia il metodo principale della classe, è composta da semplici chiamate a metodi di psutil i quali valori sono poi assegnati ad un dizionario.

4.4.1.1 Funzione get_mac()

```
for name in psutil.net_if_addrs():
    for interface in addrs[name]:
        if interface.address == self.ip:
            for c in addrs[name]:
                if len(c.address) == self.MAC_LENGTH:
                    mac = c.address
                    break
            if self.mac == mac:
                break
        break
self.mac = mac
return mac
```

Per trovare l'indirizzo MAC, è necessario sapere un IP con il quale risalire all'interfaccia. Per ogni scheda di rete, quindi, vengono analizzati i complessi oggetti di psutil alla ricerca di una corrispondenza tra gli IP. L'array risultante contiene diversi altri attributi, tra i quali appaiono altri "address" diversi dal MAC. Quello richiesto viene riconosciuto in base alla lunghezza, sempre uguale a 17 (ad esempio "0A-00-27-00-00-0F").

Se questo non era ancora salvato in un attributo, significa che è la prima volta che viene richiesto: in questo caso, esegue anche un log.

4.4.2 Behaviour

Questa classe studia e analizza il comportamento del computer distinguendo il traffico di rete normale da quello sospetto. Nella fase di apprendimento, tutte le nuove connessioni sono considerate sicure, mentre durante l'analisi sono tutte sospette.

Alcuni dei suoi metodi sono soggetti ad un polling continuo, che ne verifica i cambiamenti. Per evitare generare log superflui, è stato scelto di notificare **solo le informazioni "nuove"**. Viene quindi eseguito un log solo quando il numero di connessioni è diverso da quello precedente: negli altri casi, non sarà effettuata alcuna operazione degna di nota.

4.4.2.1 Funzione check_connections()

Prende le connessioni e riempie il modello in modo diverso a seconda della fase attiva. Analizza ogni processo iterando i PID e, se questo ha effettivamente delle connessioni attive, le salva in un array temporaneo che sarà trattato dalla funzione `update_model()`.

```
for id in psutil.pids():
    try:
        p = psutil.Process(id)
        # il processo è ignorato se non ha connessioni o se non è attivo
        if p.connections() == [] or not p.is_running():
            continue
        # ogni processo può avere più connessioni
        for conn in range(0, len(p.connections())):
            if p.connections()[conn][5] not in self.EXCLUDED_STATUS:
                # la sicurezza (safe) è impostata ora, ma sarà il server
                # a decidere se mantenere effettivamente questo stato
                c = {'proc': p.name(), 'status': p.connections()[conn][5],
                    'source': ['', ''], 'dest': ['', ''], 'proto': 'unknown', 'safe': self.phase}

                port = p.connections()[conn].laddr.port
                c['source'] = [p.connections()[conn].laddr.ip, port]
                c['proto'] = self.get_protocol(port)

                # non tutte le connessioni hanno un indirizzo di destinazione
                try:
                    if hasattr(p.connections()[conn].raddr, 'ip'):
                        # port è una variabile perchè viene usato anche in seguito
                        port = p.connections()[conn].raddr.port
                        c['dest'] = [p.connections()[conn].raddr.ip, port]
                except:
                    pass

                # se il protocollo è sconosciuto, prova a prendere quello della destinazione
                if c['proto'] == self.UNKNOWN_PROTO:
                    c['proto'] = self.get_protocol(port)
                data.append(c)
            except Exception:
                pass
```

Delle connessioni trovate è salvato nome, indirizzo e porta sorgente, indirizzo e porta di destinazione, il protocollo e lo stato. L'attributo della fase di comportamento `phase` permette anche di stabilire se la connessione è sicura: se è `True`, quindi la fase è di apprendimento, ogni connessione sarà accettata. Se è `False` (fase di analisi), ogni nuova connessione deve essere considerata sospetta. La sicurezza è impostata in questo punto, ma **sarà il server a decidere** se mantenere effettivamente questo stato o se applicare quello del database. Le altre informazioni sono prese dagli attributi di `psutil.Process(id)[i]`, che salva IP e porta nelle namedtuple `laddr` e `raddr`: l'uso di questo particolare tipo di dato rende complesso controllare se l'informazione esiste. È quindi usato un semplice blocco try-catch per evitare errori. Di default, la destinazione ha il valore `['', '']` (due elementi di una lista che corrispondono ad ip e porta, ma vuoti): questo **evita problemi con i controlli futuri** di MongoDB, che non gestisce i valori vuoti.

Dopo aver salvato le connessioni, viene avviato il metodo `update_model()` in un modo particolare: gli array sono scambiati per rispettare l'ordine delle priorità che questi hanno.

```
originalModel = self.model.copy()
self.model = data
self.update_model(originalModel)
```

In questo modo, `update_model()` considererà i dati appena ricavati dal sistema come **secondari**, evitando di applicare incorrettamente l'attributo "safe". Il modello originale sarà la base di riferimento, poiché è anche l'attributo più vicino alle informazioni del database.

4.4.2.2 Protocolli personalizzati

Grazie all'attributo `customProto` è possibile stabilire una **lista di coppie porta-protocollo personalizzata** per software aziendali o altri protocolli noti. In questo caso sono specificate le porte del sistema Butler con delle piccole descrizioni identificative.

```
customProto = {
    8080: 'CPT Proxy', 27017: 'Butler MongoDB',
    20210: 'ButlerAPI', 20211: 'Butler Server Control',
    20212: 'Butler Web GUI', 20213: 'Butler live preview',
    20219: 'Butler Client CommandListener'}
```

Dato che ogni connessione usa una sola porta tra quelle riservate (80, 443, ...), che corrisponde ad un unico protocollo, di questo è **salvato un solo attributo** per connessione (l'altro sarebbe sempre nel range dinamico). La funzione **riconosce per esclusione** se il protocollo specifico è per la sorgente o per la destinazione: se il primo non è noto, la connessione non è in ascolto, e il servizio sarà associato alla destinazione.

4.4.2.3 Funzione `update_model()`

Unisce il modello attuale con le nuove connessioni, senza scartarne nessuna e facendo attenzione a non duplicarle. Dato che non è sicuro modificare un array del quale si stanno iterando i valori, il modello attuale viene copiato in una lista temporanea `newModel`. Sia questo che i nuovi dati vengono controllati per rimuovere "duplicati" cioè connessioni molto simili ma considerate uguali (in base al controllo di `conn_match()`).

```
self.model = self.remove_duplicates(self.model.copy())
newModel = self.model.copy()
newData = self.remove_duplicates(newData)
```

La priorità delle informazioni viene data ai **nuovi dati passati** (che solitamente corrispondono a quelli del database), mentre nell'attributo `self.model` sono memorizzati dati che vanno potenzialmente rimpiazzati ma in nessun caso dimenticati (è questo che contiene le nuove connessioni).

Nel loop principale della funzione, per ogni nuova connessione viene cercata una **corrispondenza nel modello corrente**: se viene trovata, dei dati viene aggiornata unicamente la sicurezza contenuta nei nuovi valori poiché questi **provengono dal database**. Il resto delle informazioni può invece essere ignorato, perché potrebbe contenere porte dinamiche non più usate. È possibile che durante questo loop, in parallelo un array cambi i suoi valori, modificati da un'altra thread: la condizione `len(newModel) > i` (falsa se il modello ha perso dati) si accorge di questa inconsistenza e riavvia la funzione intera:

```
for newConn in newData:
    found = False
    i = 0
    for c in self.model:
        if self.conn_match(newConn, c):
            if len(newModel) > i:
                newModel[i]['safe'] = newConn['safe']
            else:
                self.update_model(newData)
            found = True
            break
        i+=1
    if not found:
        newModel.append(newConn)
```

Infine, se nessuna delle connessioni corrisponde, questa viene aggiunta: questa condizione si avvera, ad esempio, al riavvio del Butler quando il server comunica lui **l'intero modello salvato** completo di connessioni al momento non attive:

```
if newData == []:
    newModel = self.model
```

In questo modo si mantengono **sia i dati del modello precedente, sia quelli nuovi**, ma si assicura che la sicurezza e le informazioni secondarie applicate siano aggiornate.

Con questa funzione è possibile modificare l'intero modello o anche una singola connessione, dato che accetta liste di dizionari (cioè liste di connessioni).

4.4.2.4 Funzione conn_match()

Per distinguere le connessioni tra di loro è stata implementata una **funzione di confronto** apposita. Ogni connessione comprende diverse informazioni, nessuna delle quali può essere usata come identificativo. È quindi necessario controllare diversi possibili casi nel seguente modo:

```
return (
    # il nome del processo deve corrispondere
    conn1['proc'] == conn2['proc'] and (
        (
            # se entrambe le destinazioni sono vuote, la sorgente deve
            # corrispondere completamente
            conn1['dest'] == ['', ''] and
            conn2['dest'] == ['', '']
            and conn1['source'] == conn2['source']
        ) or (
            # se la destinazione non è vuota,
            # 3 su 4 degli altri campi (ip1, porta1, ip2, porta2) devono corrispondere
            conn1['dest'] != ['', '']
            and (
                conn1['dest'] == conn2['dest']
                and conn1['source'][0] == conn2['source'][0]
            ) or (
                conn1['dest'][0] == conn2['dest'][0]
                and conn1['source'] == conn2['source'])
        )
    )
)
```

4.4.3 Butler

Questa classe rappresenta un modello astratto di Butler (maggiordomo), ed è il punto di avvio dell'agent client. Le nuove funzioni comprendono `get_details()` per ricavare i dettagli del computer (sono ritornati unicamente quelli dei moduli attivi) più altre tre di impostazione di parametri: `toggle_module()`, `toggle_phase()` e `update_model()`. Alcune ulteriori modifiche sono state effettuate per la gestione del MAC address: ora, dopo aver ricavato l'IP, questo è usato per trovare la scheda di rete che identificherà l'host (trovata grazie a `get_mac()` della classe Inventory).

Il Butler può inviare le sue informazioni automaticamente grazie a `check_details()`. Sono ora anche gestiti i moduli attivi grazie al dizionario `modules` e a delle costanti per gli indici: in alcuni punti del programma si fa riferimento a questo array per decidere se procedere con un'operazione o evitarla perché parte di un modulo disattivato. Lo stato di default dei moduli è specificato in un parametro del file di configurazione del client, ma è anche **assegnato dal server** all'autenticazione.

4.4.3.1 Funzione check_details()

Controlla periodicamente le informazioni dell'host e, se sono cambiate dalle precedenti, le invia al server.

```
while True:
    if self.connected:
        oldDetails = {
            'inventory': self.inventory.data,
            'model': self.behaviour.model}
        currentDetails = self.get_details()
        newDetails = {}
        # inventario e model potrebbero essere disattivati
        # quindi verifica anche che la chiave esista
        if 'inventory' in currentDetails and oldDetails['inventory'] != currentDetails['inventory']:
            newDetails['inventory'] = currentDetails['inventory']
        if 'model' in currentDetails and oldDetails['model'] != currentDetails['model']:
            ...
        # invia solo se sono state trovate differenze
        if newDetails != {}:
            newDetails['mac'] = self.mac
            successful = self.msg.send_details(newDetails)
            if not successful and self.msg.server_online(self.addr):
                self.disconnect()
        sleep(interval)
```

Viene avviato come thread e, quando il Bulter è connesso, controlla ad intervalli regolari le sue informazioni e le confronta con quelle precedenti: le **differenze** sono inviate al server, che non potrà mostrare subito i dati aggiornati nel centro di controllo ma almeno **garantirà il salvataggio** di questi nel database in caso di disconnessione del client. L'attivazione di questa funzionalità e l'intervallo possono essere specificati nel file di configurazione. I dati "vecchi" sono presi direttamente da attributi (`behaviour.model` e `inventory.data`), mentre quelli nuovi sono ricavati grazie alle funzioni apposite con aggiornamento integrato (`behaviour.check_connections()` e `inventory.get_data()`). I dati divergenti sono aggiunti ad un array e, se questo non è vuoto, viene attivata la procedura di invio al server.

4.4.4 CommandListener

Questa classe funge da API REST per le richieste del server. Ascolta ogni possibile informazione proveniente dalla parte server, tra le quali ci sono i comandi relativi ai nuovi moduli. Sono dunque stati aggiunti dei **semplici endpoints** per gestire le richieste dei dettagli, per cambiare lo stato di un modulo, la fase del comportamento e lo stato delle connessioni. Tutte richiamano apposite funzioni di callback della classe Butler, alle quali passano i dati contenuti nella richiesta ricevuta. Infine, ritornano il codice di stato **200** (ok)⁶ poiché coerente la gestione effettuata, dato che altri come il 202 (accepted) implicano che la richiesta stia venendo processata in parallelo, ma non è questo il caso.

4.4.5 Messenger

L'unica modifica apportata a questa classe è l'aggiunta della funzione `send_details()`, per mandare periodicamente al server le informazioni aggiornate del client.

4.4.6 Config.json (client)

Nel file di configurazione del client sono stati aggiunti due parametri. Il primo, `automaticSendInterval`, permette di gestire contemporaneamente sia l'attivazione dell'invio dei dati al server, sia il tempo tra un aggiornamento e l'altro. Funziona come `bufferTimer` del programma server: se il parametro è un intero maggiore o uguale a 1, il suo valore viene usato come timer.

Il secondo è `modules` e gestisce lo stato di default dei moduli tramite un dizionario di valori booleani. Questo parametro è semplicemente letto dal codice e applicato all'omonimo array. La gestione del resto delle configurazioni è rimasta identica: non sono stati aggiunti controlli di validità sui parametri.

⁶ <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>, HTTP response status codes - HTTP | MDN, visitato il 12-05-2021

4.5 Parte server

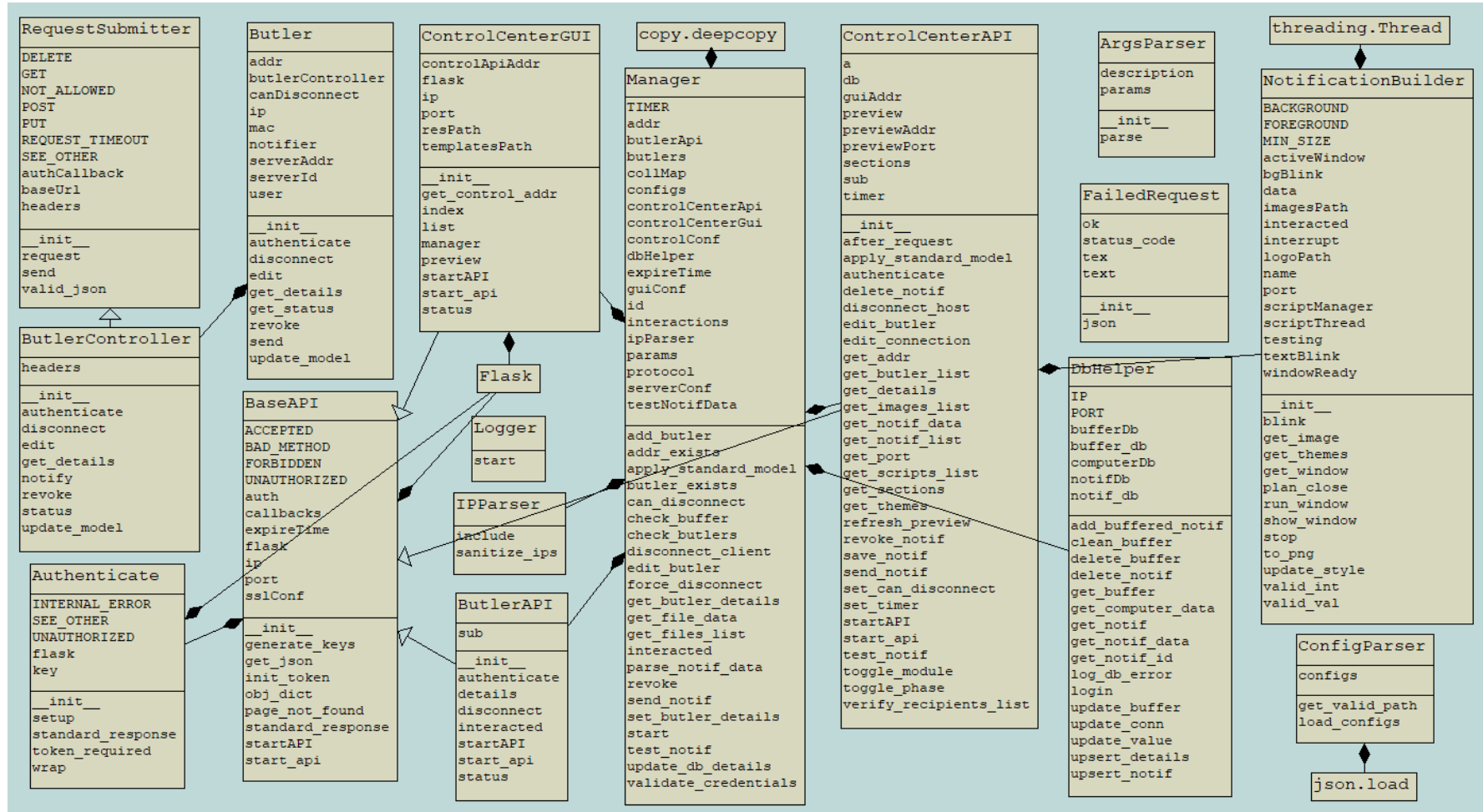


Figura 11 - Il diagramma delle classi del programma client

Il programma server ha visto unicamente delle modifiche alle classi già esistenti; non sono stati aggiunti componenti. Questi cambiamenti sono ora descritti.

4.5.1 Manager

Questa classe gestisce la parte server del programma Butler. Oltre alla modifica all'autenticazione, sono state aggiunte alcune funzioni per permettere i nuovi scambi di informazioni tra centro di controllo web, database e client.

4.5.1.1 Sincronizzazione dei dati

Con due nuove estensioni e con l'obiettivo di separare il carico di lavoro è stato necessario porre particolare attenzione **sull'integrità e sulla consistenza dei dati**: questi si possono trovare contemporaneamente sia sul client che sul server, ma sono ulteriormente sparsi tra input del client, output dal client, interfaccia del server, classe Manager del programma server e database (anche questi a tratti diversi tra input e output). Se per l'inventario non è un problema, dato che è una lista piuttosto statica dopo la prima definizione all'autenticazione del Butler, le connessioni compongono invece un problema. Trattandosi di **serie di liste annidate**, la consistenza è messa ancora più in discussione a causa del tempo necessario per trattare tutti i dati. Non è però possibile delegare tutta la gestione ad un solo componente (ad esempio far fare tutto al server), poiché il traffico dei dati aumenterebbe a dismisura.

È stato scelto di aggiornare in parallelo il **client e il database**, dato che entrambi hanno un modo di distinguere le connessioni, mentre l'oggetto Butler continuerà a fare unicamente da tramite. Le informazioni, prima di essere mostrate nel centro di controllo, sono:

- richieste al Butler
- salvate sul database
- riscaldate corrette dal database
- inviate corrette al Butler

In questo modo, si assicura coerenza tra i dati. Le funzioni che applicano questo controllo sono `get_butler_details()` e `set_butler_details()`.

Si ha dunque un ruolo secondario del database: da solo, questo non fornisce dati (tranne in pochi fondamentali casi come durante l'avvio di un Butler), ma si limita a salvarli per analisi future. Il programma server richiede sempre le informazioni ai clients per garantire che questi siano sempre aggiornati.

4.5.1.2 Funzione `get_butler_details()`

Richiede i dettagli al Butler e gestisce i **valori vuoti** che il client ritorna se non ha certi moduli attivi. La sua funzione è quindi limitata a riempire di dati ricevuti o vuoti un array, ma questo permetterà ad altri componenti di sapere come comportarsi: quando usata come callback di `ContolCenterAPI`, le informazioni nulle permettono all'interfaccia di sapere quali sezioni caricare (o se non caricare del tutto il modal), mentre quando è `set_butler_details()` a chiamarla, questa può decidere dove mandare quali informazioni senza incorrere in errori. Gestisce inventory e model, ma **non modules**. Si tratta di un flag utilizzabile da altre classi per capire che il database non ha ritornato dati.

4.5.1.3 Funzione `set_butler_details()`

Sincronizza le informazioni tra database e client garantendo la loro consistenza. Usa `get_butler_details()` per prendere in modo sicuro le informazioni di un Butler, e tratta ogni attributo in modo opportuno: l'inventario è modificato solo dal client, quindi sono le sue informazioni ad avere la precedenza sul resto e a dover essere aggiornate sul database. Il modello, invece, è principalmente gestito dal database e modificato dall'amministratore, dunque il client non può scegliere quali informazioni modificare ma può comunque inserirne di nuove. I dettagli del Butler sono richiesti **prima** di controllare se questo esiste ancora, visto che durante la chiamata l'host potrebbe essere rimosso dalla lista.

Questa funzione ha un compito delicato: oltre al fatto che si richiama a vicenda con quella che richiede i dati (deve quindi evitare qualsiasi potenziale loop), va anche limitato l'avverarsi delle sue condizioni interne, dal momento che ognuna esegue una pesante operazione di scrittura sul client o sul database. Il controllo viene eseguito confrontando l'array dei dettagli presi dal Butler e quelli presi dal database, per evitare che dati identici vadano comunque ad essere inviati:

```
sameAttr = []
for attr in butlerDetails:
    if attr in dbDetails and butlerDetails[attr] == dbDetails[attr]:
        sameAttr.append(attr)
for attr in sameAttr:
    del butlerDetails[attr]
    del dbDetails[attr]
```

Dovendo modificare le chiavi di dizionari, anche in questo caso è stato necessario dividere l'operazione di loop degli attributi e quella di eliminazione in due passaggi. La condizione verifica che l'attributo esista in entrambi gli array e che i valori al loro interno corrispondano.

4.5.1.4 Funzione update_db_details()

Aggiorna le informazioni ricevute dal Butler nel database. Non esegue alcun controllo, dunque all'apparenza si tratta di un modo per evitare le verifiche di integrità dei dati: in realtà, per **separare il carico di lavoro**, in questo caso è il client ad eseguire i controlli (tramite il metodo check_details() della classe Butler) al fine di far raggiungere a questa funzione unicamente dati sconosciuti al database. In poche parole, il server certifica la congruenza dei dati modificati dall'amministratore, mentre il client si occupa della correttezza delle informazioni nuove da lui trovate.

4.5.1.5 Funzione apply_standard_model()

Invia i dati del modello standard ad un Butler. Il modello contenente le connessioni con uno stato fisso (di solito sono quelle sempre permesse) è memorizzato nel database sotto il **MAC ""** (una stringa vuota). Si tratta di un flag usato anche per il salvataggio del template di base delle notifiche.

```
standardModel = self.dbHelper.get_computer_data(self.STANDARD_MODEL_MAC)
if 'model' in standardModel:
    standardModel['mac'] = self.butlers[addr].mac
    self.dbHelper.upsert_details(standardModel)
    self.butlers[addr].update_model(standardModel['model'])
```

Grazie a questa meccanica, la sua implementazione può essere limitata ad una query con id "". I programmi sono già pronti per accettare liste di connessioni e gestirne la priorità.

4.5.2 DbHelper

Questa classe permette di stabilire una connessione al database MongoDB e di eseguire varie operazioni di base sui documenti delle notifiche, dei buffer e delle informazioni dei computers.

Insieme a Behaviour del programma client, è la classe con la logica aggiuntiva **più complessa**. Per leggere i dati è sufficiente la funzione `get_computer_data()`, mentre la scrittura si basa su `upsert_details()` e altre due funzioni di supporto `update_conn()` e `update_value()`. Per connettersi alla nuova collezione o generarla se non presente, è stato creato l'attributo `computerColl` collegato a "computer". Come progettato, è sempre sfruttato l'indirizzo MAC come chiave dei dati.

4.5.2.1 Funzione `upsert_details()`

Analizza i dati passati e li inserisce o li aggiorna nel database. Necessita di verificare i dati **per ogni attributo**, dato che devono essere trattati in modo differente tra loro: con attributo si intende la chiave del primo livello del database, che può essere allegata a qualunque altro dato. Sono effettuate cinque distinzioni, due delle quali sono generiche e tre specifiche:

- MAC Address: viene ignorato perché non va aggiornato
- Lista di connessioni: ognuna viene trattata da `update_conn()`
- Connessione singola: viene inviata direttamente a `update_conn()`
- Valori generici: vengono salvati grazie a `update_value()`
- Dizionario di valori generici: per ognuno, viene trovata la chiave specifica dei dati e viene aggiunta a quella dell'attributo. Tralasciando il secondo identificativo, tutti gli altri valori del dizionario sarebbero sovrascritti. Ad esempio, aggiornare `modules` con `{inventory: True}` renderebbe `inventory` l'unico dato, mentre modificando `modules.inventory` si cambia il singolo dato.

```

▼ modules: Object
  notification: true
  inventory: true
  behaviour: true
    
```

Figura 14 - L'array originale

```

▼ modules: Object
  notification: false
  inventory: true
  behaviour: true
    
```

Figura 13 - Un esempio di modifica corretta

```

▼ modules: Object
  notification: false
    
```

Figura 12 - La modifica sbagliata senza la chiave aggiuntiva

4.5.2.2 Funzione `update_conn()`

Inserisce o aggiorna una connessione del modello. La query usa il tag `$elemMatch` per verificare la corrispondenza delle informazioni "identificative" della connessione. La condizione è la stessa di `conn_match()` di Behaviour, ma "tradotta" per essere riconosciuta da MongoDB. Data l'**estrema differenza** tra le due sintassi, non sarebbe stato conveniente unificare il controllo in una funzione accessibile ad entrambe le classi.

```

result = self.computerDb.find_one_and_update(
    {'mac': mac, 'model': {'$elemMatch': {
        'proc': conn['proc'],
        '$or': [
            {'$and': [
                {'dest.{}'.format(self.IP): ''},
                {'dest.{}'.format(self.PORT): ''},
                {'dest.{}'.format(self.IP): conn['dest'][self.IP]},
                {'dest.{}'.format(self.PORT): conn['dest'][self.PORT]},
                {'source.{}'.format(self.IP): conn['source'][self.IP]},
                {'source.{}'.format(self.PORT): conn['source'][self.PORT]},
            ]},
            {'$and': [
                {'dest.{}'.format(self.IP): {'$ne': ''}},
                {'dest.{}'.format(self.PORT): {'$ne': ''}},
                {'dest.{}'.format(self.IP): conn['dest'][self.IP]},
                {'dest.{}'.format(self.PORT): conn['dest'][self.PORT]},
                {'source.{}'.format(self.IP): conn['source'][self.IP]},
                {'source.{}'.format(self.PORT): conn['source'][self.PORT]},
            ]}
        ]}
    }},
    {'$set': {'model.$': conn}})
    
```

4.5.3 Butler

Questa classe rappresenta un client Butler, salvando diversi attributi utili alla gestione e alla comunicazione. Il server può comunicare con questa classe come se si trattasse di un client, delegando la gestione della comunicazione. Sono quindi state aggiunte le funzioni `get_details()`, `edit()` e `update_model()` che contengono la logica minima per far avanzare i dati alla classe che manderà effettivamente la richiesta.

4.5.4 ButlerController

Questa classe permette di inviare messaggi ai Butlers. Porta a termine le richieste generate dal centro di controllo, con le tre funzioni `get_details()`, `edit()` e `update_model()`. Per ognuna prepara i dati della richiesta, li invia e ritorna la risposta del Butler.

4.5.5 ControlCenterAPI

Questa classe permette il controllo del programma server tramite richieste REST. È stato ampliato con alcune funzioni (“comandi”) ulteriori, tutte usate per cambiare informazioni dei Butlers. Lo **stato dei moduli** (“/module”, la **fase del comportamento** (“/phase”) e la **sicurezza delle connessioni** (“/connection”) possono essere visti come una coppia chiave-stato, perciò tutti e tre puntano alla funzione `edit_butler()`. È presente anche l’endpoint “/standardModel” per l’applicazione del **modello standard delle connessioni**, e “/details” per poter accedere alle informazioni dei Butlers. Quest’ultima funzione usa il controllo integrato in `get_butler_details()` per verificare che i dati siano validi, e se non lo sono ritorna un messaggio di errore.

4.5.5.1 Funzione `edit_butler()`

Permette di modificare un parametro di un Butler, usando sempre l’attributo “data” per ricavare in modo globale le informazioni passate.

```
addr = self.get_json(request, 'addr', '')
data = self.get_json(request, 'data', [])
# request.path corrisponde sia all'endpoint appena raggiunto che a quello del client
self.callbacks['edit_butler'](addr, data, request.path)
return self.standard_response()
```

In questo modo, è possibile evitare di avere tre funzioni a cascata per diverse altre classi. In più, non essendo specifico per i comandi usati in questo caso (data può assumere qualunque valore), può facilmente essere usato in espansioni future senza alcuna ulteriore modifica: è un modo per cambiare liberamente qualunque valore del client, se questo ne gestisce il caso. Passa i dati alla callback `edit_butler()`, alla quale allega anche `request.path` che corrisponde sia al nome dell’endpoint raggiunto che a quello da usare verso il client.

4.5.6 ButlerAPI

Questa classe permette di interpretare le richieste originate dai Butler. È stato aggiunto l’endpoint “/details”, richiamato alla ricezione dei dettagli di un Butler, che chiama una callback passando i dati ricevuti.

4.5.7 Lista e dettagli dei clients “butlerList.html”

La parte dell'interfaccia grafica che ha subito modifiche è la sezione della lista dei Butlers. È stato aggiunto un bottone (con icona “+”) di fianco agli altri controlli:

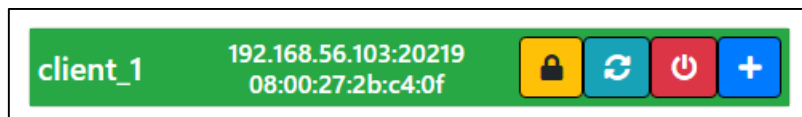


Figura 15 - Lo stile dei Butlers connessi

Al click, questo apre il modal di dettaglio del Butler specifico.

4.5.7.1 Funzione loadDetails()

Imposta i dettagli nei contenitori corretti e ne gestisce lo stile. I dati vengono aggiornati all'apertura del modal con AJAX, grazie a questa funzione e all'endpoint del server “/details”.

```
function loadDetails(data) {
    generateModulesList(data['mac'], data['modules']);
    if ('inventory' in data) {
        // l'inventario è rappresentato come stringa
        $('#inventoryData').text(JSON.stringify(data['inventory'], null, 4));
    }
    if ('model' in data) {
        $('#phaseToggle').prop('checked', !data['phase']);
        generateConnectionsList(data['mac'], data['model']);
        $('#trafficModel').collapse((data['phase']? 'hide': 'show'));
        $('#analysisPhase, #learningPhase').removeClass('show');
        if (data['phase']) {
            $('#analysisPhase').addClass('collapse');
            $('#learningPhase').addClass('show');
            $('#learningPhase').removeClass('collapse');
        } else {
            $('#learningPhase').addClass('collapse');
            //$('#analysisPhase').addClass('show');
            $('#analysisPhase').removeClass('collapse');
        }
    }
}
```

Un altro compito di questa funzione è impostare lo stile corretto del toggle e del testo dinamico della fase del comportamento. Una volta ricevute le informazioni, queste permettono di generare automaticamente la lista dei moduli grazie a `generateModulesList()` e la lista delle connessioni con `generateConnectionList()`.

4.5.7.2 Funzione generateConnectionsList()

Genera la lista delle connessioni in base ai dati ricevuti. Il suo funzionamento è molto simile alla controparte per mostrare i moduli e la lista di clients, ma necessita alcune operazioni ulteriori: ad esempio, lo stile differisce in base allo stato della destinazione.

```
function generateConnectionsList(addr, model) {
    $('#connectionsList').children().remove();
    for ([key, data] of Object.entries(model)) {
        var dest = "";

        // i destinatari sono mostrati solo se presenti
        if (data['dest'][0] != '') {
            dest = `
                <div class="row">
                    <h6>Verso: ${data['dest'][0]} ${data['dest'][1]}</h6>
                </div>
            `;
        }
        # <generazione del codice HTML...>
    };
}
```

Per poter essere riconosciute in seguito, ogni connessione viene salvata nel **localStorage** con un ID che consiste nell'indirizzo concatenato alla stringa "conn" e all'indice della connessione (es. "0A-00-27-00-00-1Econn1", ...):

```
localStorage.setItem(addr+'conn'+key, JSON.stringify(data))
```

Questo identificativo è anche assegnato all'elemento per essere riconosciuto. Contenendo caratteri speciali, jQuery potrebbe interpretarne alcuni come selettori, fallendo nella ricerca dell'elemento: è quindi usata la funzione integrata \$.escapeSelector() che evita questi problemi:

```
$('#'+$.escapeSelector(addr)+'conn'+key).prop('checked', data['safe']);
```

Il dato salvato nello storage locale è un array JSON, e deve essere serializzato come stringa grazie a JSON.stringify(). Per poter essere ripreso (durante il cambio di stato), si usa l'id dell'elemento che genera l'evento relativo (i toggles):

```
$('#body').on('input', '.connectionToggle', function (e) {
    var connection = JSON.parse(localStorage.getItem(this.id));
    ...
});
```

Gli elementi generati includono classi legate a degli eventi predeterminati, mentre i dati più specifici vengono ricavati dagli ID: in questo esempio, ogni toggle ha la classe **connectionToggle** alla quale corrisponde un evento generico:

```
$('#connectionsList').append(`
  <div class="mb-3 card ${data['safe']]? 'bg-success': 'bg-warning'}">
    <div class="card-body">
      <label data-toggle="collapse" class="control-toggle">
        <input id="${addr}conn${key}" class="connectionToggle" type="checkbox"
          data-host="${addr}">
        <div class="toggle-button"></div>
      </label>
      <h5>${data['proc']] - "${data['proto']}" - ${data['status']}</h5>

      <div>
        <div class="row">
          <h6>${data['status']] == 'LISTEN'? 'Su': 'Da':
            ${data['source']][0]} ${data['source']][1]}</h6>
          </div>
          ${dest}
        </div>
      </div>
    </div>`);
```

Viene poi usato l'id che corrisponde a quello del localStorage per andare a recuperare la connessione specifica ed essere inviata al server.

4.5.8 Aspetto grafico

Seguendo lo stile del resto dell'interfaccia, il modal è sviluppato in verticale con sezioni a fisarmonica e toggle che, all'occorrenza, nascondono dalla pagina alcuni elementi.

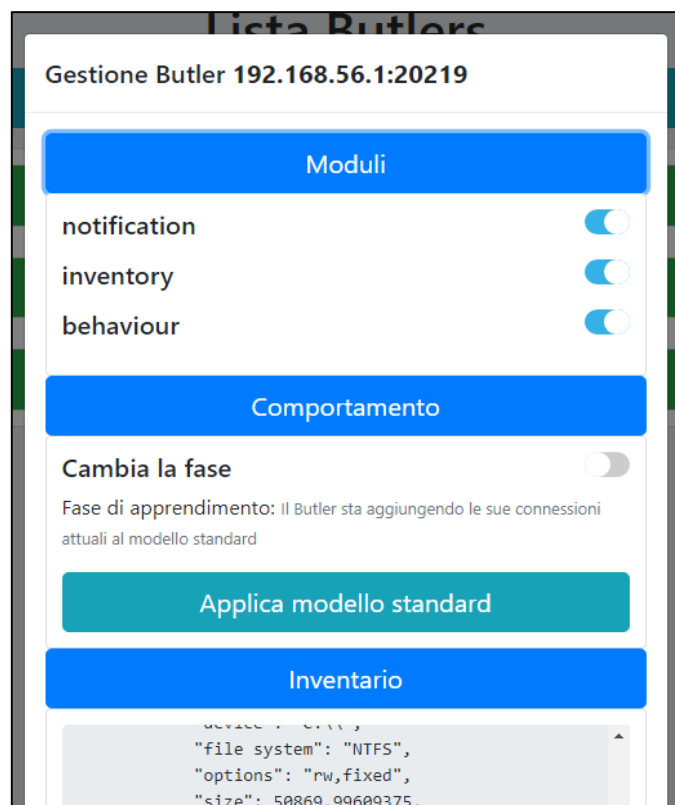


Figura 16 - Il modal di dettaglio

Anche se esiste una sola volta nello stile della pagina, modificando i dati ad ogni apertura si dà l'impressione di molteplici pagine apposite.

Disattivando i moduli, gli eventi legati agli switch nasconderanno o mostreranno le sezioni apposite sottostanti. La logica è simile a quella per il cambio tra notifiche di sistema e pop-up: un metodo apposito sfrutta delle classi specifiche, che in questo caso devono essere composte dal nome del modulo e il suffisso "OnlySection" (es. "inventoryOnlySection", cioè una sezione di pagina da mostrare solo quanto il modulo inventory è attivo).

```
function toggleSection(name) {
  if ($('#'+name).is(":checked")) {
    $('#'+name+'OnlySection').slideDown();
    $('#'+name+'OnlySection').attr('style', "");
  } else {
    $('#'+name+'OnlySection').slideUp();
  }
}
```

Sarà poi il server ad occuparsi di non inviare informazioni superflue leggendo lo stato dei moduli.

È stato scelto di aprire il modal esclusivamente quando i dati sono stati completamente caricati per offrire un'interfaccia sempre aggiornata, al posto di mostrarlo subito ma rischiare di fornire valori di default che non corrispondono a quelli reali e dare la possibilità di effettuare modifiche che non avranno effetto. Per come funziona internamente il programma, in caso di disconnessione del Butler, questo modal non si aprirà del tutto e sarà mostrato un messaggio di errore. Sarebbe stato possibile lasciare la possibilità di modifica salvando i cambiamenti solo nel database anche con il client disconnesso. È stato però scelto di impedire la modifica degli host quando questi non sono connessi, per coerenza con il resto del programma (che non prevede la gestione di informazioni di Butlers offline).

5 Test

5.1 Protocollo di test

I seguenti test danno per scontato la corretta riuscita di quelli del precedente progetto, quindi alcuni aspetti come la procedura di avvio non sono inclusi. Sono posti in ordine di esecuzione per rispettare e poter meglio rappresentare le dipendenze, dunque non corrispondono alla sequenza dei requisiti.

Alcuni test sfruttano l'interfaccia web del centro di controllo per semplificare il test delle procedure, ma, dato che tutti i comandi sono direttamente collegati a richieste HTTP, sarebbe possibile riprodurre gli stessi risultati anche con altri strumenti.

Test Case:	TC-001	Nome:	Raccolta informazioni hardware
Riferimento:	REQ-003.001		
Descrizione:	Il client raccoglie informazioni sull'hardware del PC		
Prerequisiti:	Il programma client funzionante (in Windows)		
Procedura:	<ol style="list-style-type: none"> 1. Avviare un Butler 2. Attendere che si colleghi al server 3. Visualizzare i log 		
Risultati attesi:	Appaiono informazioni riguardo l'inventario, ma non la lista completa		

Test Case:	TC-002	Nome:	Raccolta informazioni modello
Riferimento:	REQ-004.001		
Descrizione:	Il client raccoglie informazioni sul modello delle connessioni		
Prerequisiti:	Il programma client funzionante (in Windows)		
Procedura:	<ol style="list-style-type: none"> 1. Avviare un Butler 2. Attendere che si colleghi al server 3. Visualizzare i log 		
Risultati attesi:	Appaiono informazioni riguardo il modello, ma non la lista completa		

Test Case:	TC-003	Nome:	Salvataggio dati inventario
Riferimento:	REQ-001		
Descrizione:	Il server salva nel database i dati dell'inventario di ogni computer		
Prerequisiti:	TC-001, un computer con il database MongoDB installato e configurato, il programma server funzionante (in Windows)		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-001 2. Accedere al database e verificare i dati corrispondenti al PC del Butler appena connesso 		
Risultati attesi:	I dati dell'inventario sono presenti		

Test Case:	TC-004	Nome:	Salvataggio modello delle connessioni
Riferimento:	REQ-002.001		
Descrizione:	Il server salva nel database il modello delle connessioni di ogni computer		
Prerequisiti:	TC-002, un computer con il database MongoDB installato e configurato, il programma server funzionante (in Windows)		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-002 (raccolta informazioni modello) 2. Accedere al database e verificare i dati corrispondenti al PC del Butler appena connesso 		
Risultati attesi:	I dati delle connessioni sono presenti		

Test Case:	TC-005	Nome:	Visualizzazione Butlers
Riferimento:	REQ-005.001, REQ-006.001		
Descrizione:	Tutti i Butler connessi al server appaiono nell'interfaccia		
Prerequisiti:	Il programma client e il programma server funzionanti (in Windows)		
Procedura:	<ol style="list-style-type: none"> 1. Avviare il programma server 2. Avviare un Butler in condizione di connettersi al server 3. Accedere al centro di controllo 4. Eseguire il login 		
Risultati attesi:	Il client connesso appare nella lista		

Test Case:	TC-006	Nome:	Visualizzazione dettagli Butlers
Riferimento:	REQ-005.001, REQ-006.001		
Descrizione:	È possibile accedere alla scheda dei dettagli per ogni Butler		
Prerequisiti:	TC-005		
Procedura:	<ol style="list-style-type: none"> 1. Avviare almeno due Butlers 2. Cliccare il simbolo "+" di fianco ad un Butler 3. Cliccare "chiudi" 4. Cliccare il simbolo "+" di fianco ad un altro Butler 		
Risultati attesi:	Entro qualche istante vengono caricate le sezioni con i dettagli dell'host selezionato, diverse tra i vari Butlers		

Test Case:	TC-007	Nome:	Invio automatico informazioni del Butler
Riferimento:	REQ-003.001		
Descrizione:	Il client invia al server i suoi dati solo quando nota differenze		
Prerequisiti:	TC-003, TC-004		
Procedura:	<ol style="list-style-type: none"> 1. Avviare un Butler 2. Visualizzare il numero di connessioni del database 3. Aprire una nuova connessione sul computer del client (ad esempio eseguendo il comando "ssh 1.1.1.1") 4. Attendere alcuni secondi (di default nel file di configurazione sono 10) per la sincronizzazione delle informazioni 5. Aggiornare i dati del database 		
Risultati attesi:	C'è una connessione in più che corrisponde a quella aperta		

Test Case:	TC-008	Nome:	Cambio stato delle connessioni nel database
Riferimento:	REQ-006.001		
Descrizione:	Cambiare lo stato di una connessione aggiorna l'informazione nel database		
Prerequisiti:	TC-004, TC-006		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-006 (visualizzazione dettagli Butlers) 2. Espandere la sezione "Model" di un Butler 3. Accedere al database e scegliere una connessione del Butler connesso 4. Dal centro di controllo, cambiare lo stato di sicurezza di una connessione premendo il toggle a fianco 5. Aggiornare la visualizzazione sul database 		
Risultati attesi:	L'attributo "safe" cambia di stato in base a quello impostato		

Test Case:	TC-009	Nome:	Invio del modello ai clients
Riferimento:	REQ-002.002		
Descrizione:	Il server invia i dati del modello delle connessioni all'autenticazione di un Butler		
Prerequisiti:	TC-008		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-006 (visualizzazione dettagli Butlers) 2. Verificare le informazioni delle connessioni del client dal database 3. Cambiare lo stato di "safe" di alcune connessioni 4. Spegnerne il server 5. Cancellare i dati del client dal database 6. Accendere il server 7. Far ricollegare il client tramite il suo menu 8. Aggiornare la schermata di informazioni del database 9. Visualizzare lo stato delle connessioni 		
Risultati attesi:	Lo stato di tutte le connessioni è lo stesso di quello impostato in precedenza, dimostrando che il client conosce e rispetta lo stato delle impostazioni del server		

Test Case:	TC-010	Nome:	Cambio stato delle connessioni per i client
Riferimento:	REQ-002.002, REQ-006.001		
Descrizione:	Cambiare lo stato di una connessione aggiorna l'informazione sul client		
Prerequisiti:	TC-009		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-009 (invio del modello ai clients) 2. Sconnettere e riavviare il Butler al quale si ha cambiato la sicurezza della connessione 3. Aggiornare la sezione del centro di controllo 4. Visualizzare le informazioni del client 5. Verificare lo stato della connessione modificata 		
Risultati attesi:	Lo stato corrisponde a quello del TC-009, confermando che il client rispetta i cambiamenti imposti dal server		

Test Case:	TC-011	Nome:	Aggiunta automatica connessioni sicure
Riferimento:	REQ-004.001, REQ-004.002, REQ-004.003		
Descrizione:	Durante la fase di apprendimento, ogni nuova connessione viene aggiunta tra quelle sicure		
Prerequisiti:	TC-006		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-006 (visualizzazione dettagli Butlers) 2. Impostare il toggle sulla fase di apprendimento 3. Chiudere il modal di dettaglio 4. Recarsi sul Butler collegato 5. Aprire una nuova connessione sul computer (ad esempio eseguendo il comando "ssh 1.1.1.1") 6. Tornare sul centro di controllo e cliccare il simbolo "+" di fianco al Butler 7. Impostare il toggle sulla fase di analisi 8. Cercare la connessione appena aperta sul Butler 		
Risultati attesi:	La connessione appare con sfondo verde e con il toggle apposito attivato (quindi è impostata come sicura)		

Test Case:	TC-012	Nome:	Aggiunta automatica connessioni sospette
Riferimento:	REQ-004.001, REQ-004.002, REQ-004.004		
Descrizione:	Durante la fase di analisi, ogni nuova connessione viene aggiunta tra quelle sospette		
Prerequisiti:	TC-011		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-006 (visualizzazione dettagli Butlers) 2. Impostare il toggle sulla fase di analisi 3. Chiudere il modal di dettaglio 4. Recarsi sul Butler collegato 5. Aprire una nuova connessione sul computer (ad esempio eseguendo il comando "ssh 1.1.1.1") 6. Tornare sul centro di controllo e cliccare il simbolo "+" di fianco al Butler 7. Cercare la connessione appena aperta sul Butler 		
Risultati attesi:	La connessione appare con sfondo giallo e con il toggle apposito disattivato (quindi è impostata come sospetta)		

Test Case:	TC-013	Nome:	Gestione modello standard
Riferimento:	REQ-002.003		
Descrizione:	È possibile modificare il modello standard delle connessioni		
Prerequisiti:	TC-005		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-005 (visualizzazione Butlers) 		
Risultati attesi:	È disponibile una sezione con i comandi di gestione del modello generico		

Test Case:	TC-014	Nome:	Applicazione modello standard
Riferimento:	REQ-002.004, REQ-006.002		
Descrizione:	È possibile passare un modello standard delle connessioni ad ogni computer		
Prerequisiti:	TC-006		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-006 (visualizzazione dettagli Butlers) 2. Dal database, scegliere alcune connessioni di un Butler 3. Copiare le connessioni scelte in un documento con mac corrispondente a “” 4. Modificare l’attributo “safe” delle connessioni scelte nel documento originale 5. Dal centro di controllo, cambiare lo stato di alcune connessioni a caso, tra le quali ci devono essere alcune di quelle copiare 6. Cliccare “Applica modello standard” 		
Risultati attesi:	Le connessioni del modello standard tornano al loro stato originale, mentre le altre no		

Test Case:	TC-015	Nome:	Richiesta informazioni ad un Butler andato offline
Riferimento:	REQ-005, REQ-006.001		
Descrizione:	Non è possibile richiedere informazioni ad un Butler che è andato offline secondo la procedura standard		
Prerequisiti:	TC-006		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-006 (visualizzazione dettagli Butlers) 2. Cliccare il lucchetto del Butler 3. Disconnettere il Butler dal suo menu 4. Nuovamente dal centro di controllo cliccare il simbolo “+” di fianco al Butler 		
Risultati attesi:	Un messaggio d’errore spiega che il Butler è offline, e il modal di dettaglio non si apre		

Test Case:	TC-016	Nome:	Richiesta informazioni ad un Butler disconnesso forzatamente
Riferimento:	REQ-005, REQ-006.001		
Descrizione:	Non è possibile richiedere informazioni ad un Butler che è andato offline in modo forzato; senza notificarlo al server		
Prerequisiti:	TC-006		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-006 (visualizzazione dettagli Butlers) 2. Spegnerne forzatamente il Butler interrompendo il suo processo 3. Nuovamente dal centro di controllo cliccare il simbolo “+” di fianco al Butler 		
Risultati attesi:	Un messaggio d’errore spiega che il Butler è offline, e il modal di dettaglio non si apre		

Test Case:	TC-017	Nome:	Funzionamento moduli attivi
Riferimento:	REQ-007.001		
Descrizione:	È possibile interagire con le funzioni dei moduli attivi dei Butlers		
Prerequisiti:	TC-006		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-006 (visualizzazione dettagli Butlers) 2. Espandere la sezione “Moduli” di un Butler 3. Verificare che il toggle “notification” sia attivo 4. Nella sezione di gestione delle notifiche, selezionare “*” come destinatari, scegliere una notifica e inviarla 		
Risultati attesi:	Tutti i Butlers ricevono la notifica		

Test Case:	TC-018	Nome:	Disattivazione moduli
Riferimento:	REQ-007.001		
Descrizione:	È possibile disattivare i moduli dei Butlers		
Prerequisiti:	TC-017		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-017 (funzionamento moduli attivi) 2. Disattivare il toggle "notification" di un Butler 3. Scegliere la stessa notifica del TC-016 ed inviarla 		
Risultati attesi:	Il Butlers con il modulo disattivato non riceve la notifica, mentre gli altri sì		

Test Case:	TC-019	Nome:	Funzionamento con database offline
Riferimento:	REQ-007.001		
Descrizione:	Il programma gestisce il caso del database offline		
Prerequisiti:	TC-006		
Procedura:	<ol style="list-style-type: none"> 1. Eseguire il TC-006 (visualizzazione dettagli Butlers) 2. Chiudere il modal 3. Spegner il database o mandarlo offline 4. Cercare di aprire il modal di dettaglio di un qualsiasi client 		
Risultati attesi:	Un messaggio d'errore spiega che il database è offline, e il modal di dettaglio non si apre		

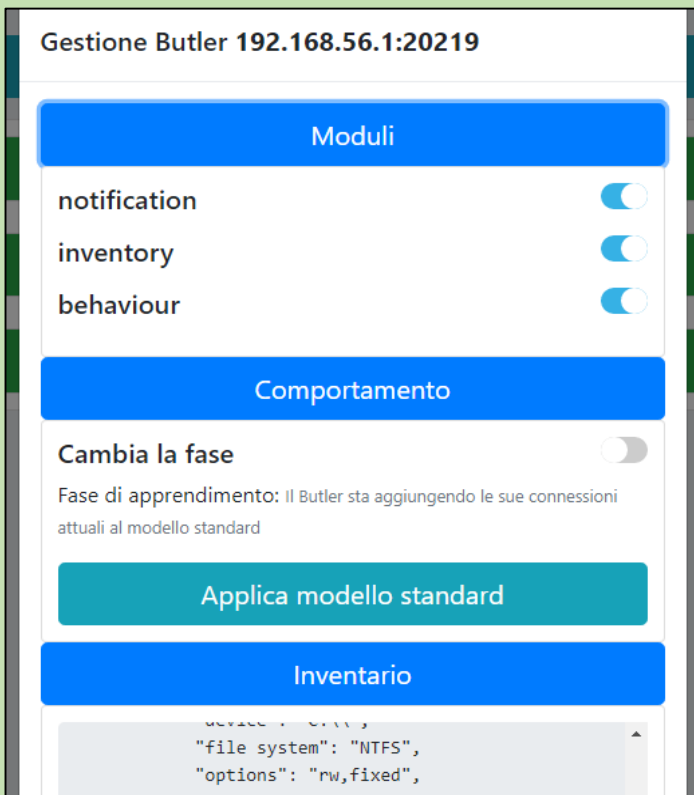
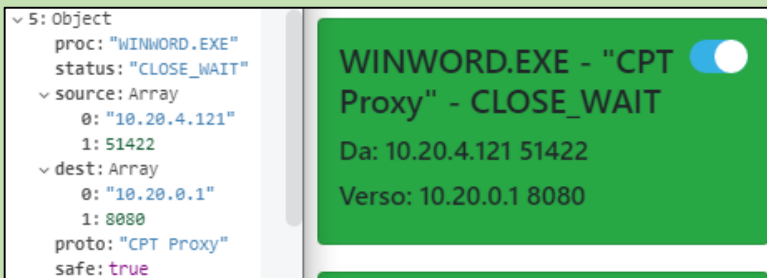
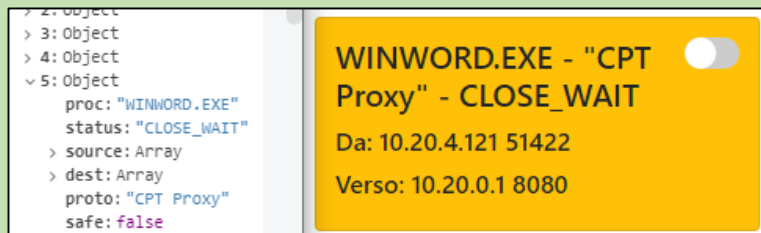
Test Case:	TC-020	Nome:	Funzionamento multiplatforma del client
Riferimento:	REQ-003, REQ-004, REQ-005, REQ-006		
Descrizione:	Il programma client funziona senza problemi anche in Linux		
Prerequisiti:	Il programma client funzionante (in Windows), TC-010, TC-012, TC-018		
Procedura:	<ol style="list-style-type: none"> 1. Usare una macchina Linux per il client e una Windows per il server 2. Eseguire il TC-010 (cambio stato delle connessioni per i client) 3. Eseguire il TC-012 (aggiunta automatica connessioni sospette) 4. Eseguire il TC-018 (disattivazione moduli) 		
Risultati attesi:	Il programma funziona in entrambi i casi con risultati uguali a quelli dei rispettivi test case		

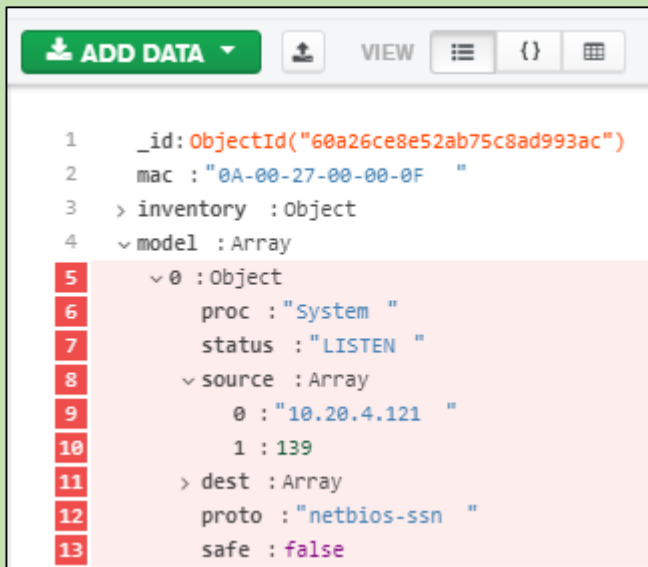
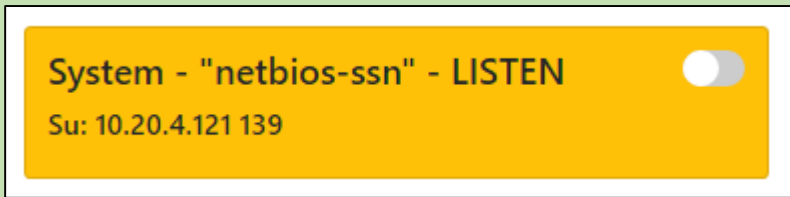
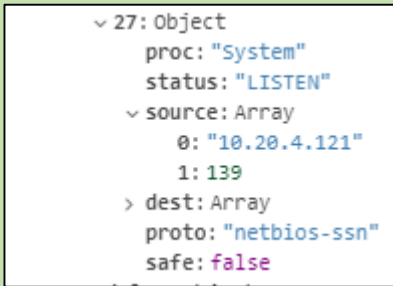
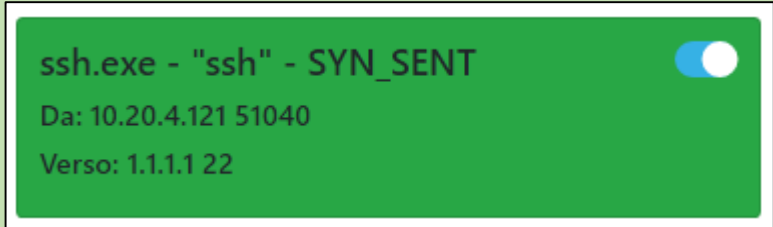
Test Case:	TC-021	Nome:	Funzionamento multiplatforma del server
Riferimento:	REQ-003, REQ-004, REQ-005, REQ-006		
Descrizione:	Il programma server funziona senza problemi anche in Linux		
Prerequisiti:	Il programma server funzionante (in Windows), TC-010, TC-012, TC-018		
Procedura:	<ol style="list-style-type: none"> 1. Usare una macchina Linux per il server e una Windows per il client 2. Eseguire il TC-010 (cambio stato delle connessioni per i client) 3. Eseguire il TC-012 (aggiunta automatica connessioni sospette) 5. Eseguire il TC-018 (disattivazione moduli) 		
Risultati attesi:	Il programma funziona in entrambi i casi con risultati uguali a quelli dei rispettivi test case		


5.2 Risultati test

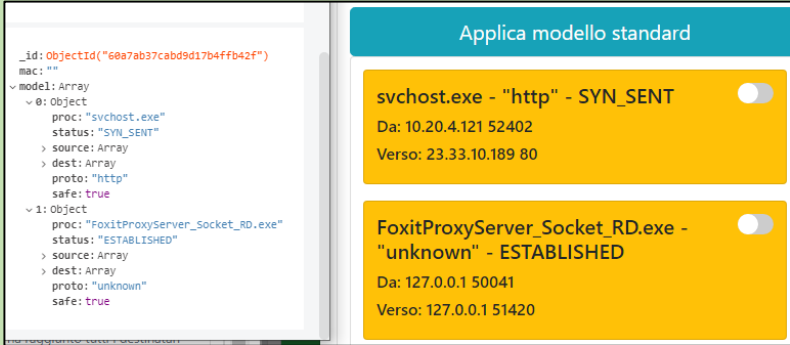
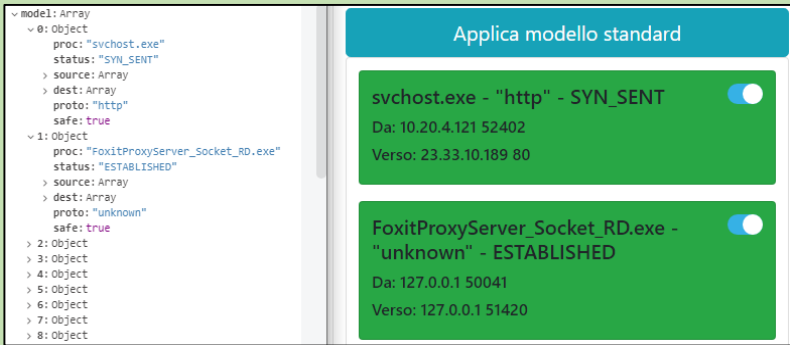
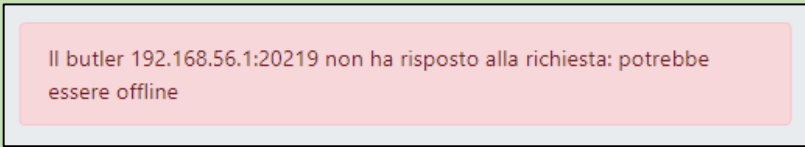
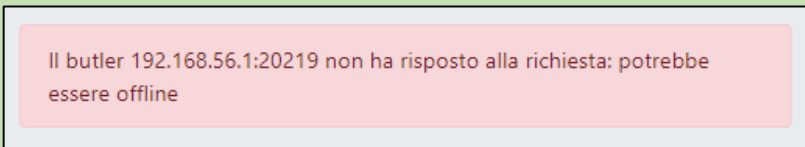
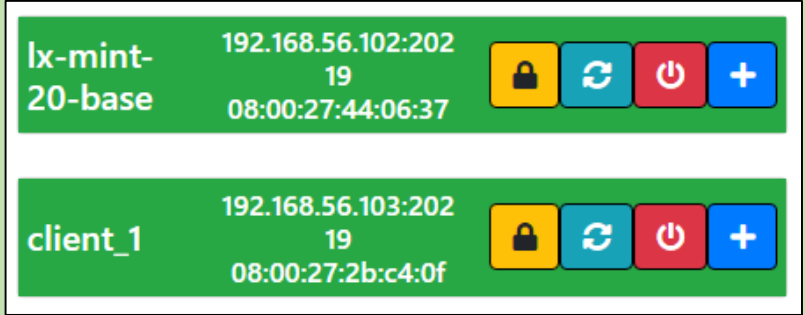
Test case	Risultato	Note	Data
TC-001	Passato	<p>Il log informa che i dati dell'inventario sono stati trovati:</p> <pre>2021-05-21 16:02:13,753 - INFO from inventory.py (get_inventory): Informazioni inventario DESKTOP-C9V1LJO (0A-00-27-00-00-0F) aggiornate</pre>	21-05-2021
TC-002	Passato	<p>Il log informa che i dati del modello sono stati trovati:</p> <pre>2021-05-21 15:55:06,764 - INFO from behaviour.py (check_connections): Ci sono 13 connessioni attive totali 2021-05-21 15:55:06,768 - INFO from behaviour.py (update_model): Il modello contiene ora 50 connessioni. Ne sono state verificate 50</pre>	21-05-2021
TC-003	Passato	<p>I dati dell'inventario vengono correttamente salvati:</p> <pre> v inventory: Object > os: Array hostname: "DESKTOP-C9V1LJO" mac: "0A-00-27-00-00-0F" last_ip: "192.168.56.1" > last_user: Array last_boot: "2021-05-20 13:15:22" > cpu: Object > interfaces: Array ram: 16382.984375 swap: 19710.984375 > disk: Array </pre>	20-05-2021

TC-004	Passato	I dati del modello vengono correttamente salvati:	20-05-2021												
		<pre> model: Array 0: Object proc: "python.exe" status: "ESTABLISHED" source: Array dest: Array proto: "unknown" safe: false 1: Object proc: "svchost.exe" status: "SYN_SENT" source: Array dest: Array proto: "unknown" safe: true 2: Object 3: Object 4: Object 5: Object 6: Object </pre>													
TC-005	Passato	I clients connessi sono visibili:	21-05-2021												
		<div> <div>Lista Butlers</div> <div> </div> <table> <tr> <td>filippo.zinet ti</td> <td>192.168.56.1:20219 0A-00-27-00-00-0F</td> <td> </td> </tr> <tr> <td>lx-mint-20- base</td> <td>192.168.56.102:20219 08:00:27:44:06:37</td> <td> </td> </tr> <tr> <td>client_1</td> <td>192.168.56.103:20219 08:00:27:2b:c4:0f</td> <td> </td> </tr> <tr> <td>FilClient</td> <td>192.168.56.105:20219 08-00-27-13-A8-FB</td> <td> </td> </tr> </table> </div>	filippo.zinet ti	192.168.56.1:20219 0A-00-27-00-00-0F		lx-mint-20- base	192.168.56.102:20219 08:00:27:44:06:37		client_1	192.168.56.103:20219 08:00:27:2b:c4:0f		FilClient	192.168.56.105:20219 08-00-27-13-A8-FB		
filippo.zinet ti	192.168.56.1:20219 0A-00-27-00-00-0F														
lx-mint-20- base	192.168.56.102:20219 08:00:27:44:06:37														
client_1	192.168.56.103:20219 08:00:27:2b:c4:0f														
FilClient	192.168.56.105:20219 08-00-27-13-A8-FB														

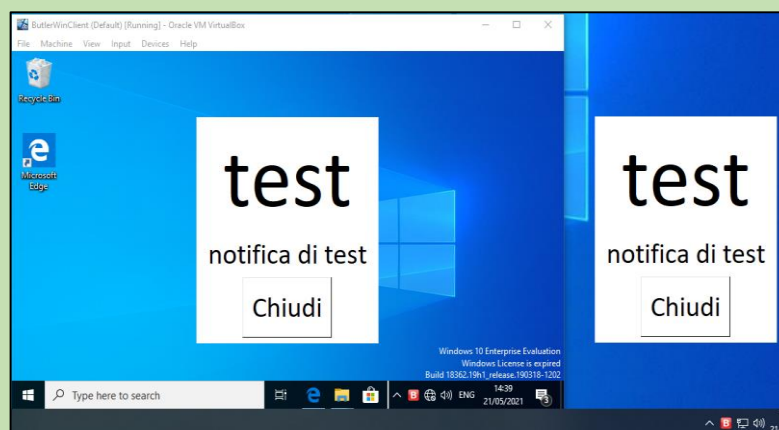
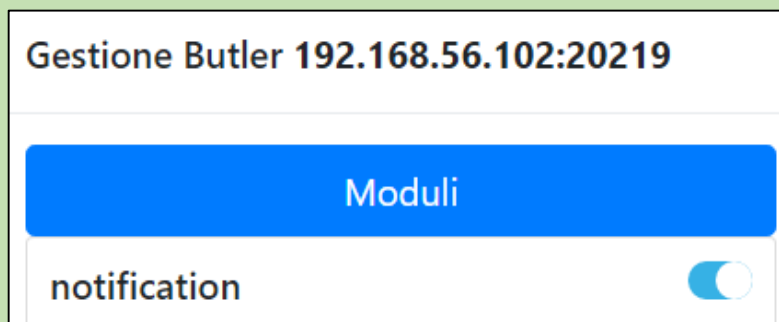
TC-006	Passato	<p>Per ogni Butler è possibile visualizzare i dettagli:</p> 	21-05-2021
TC-007	Passato	<p>Il client identifica correttamente le differenze:</p> <pre>2021-05-26 09:28:45,505 - INFO from butler.py (check_details): Invio di alcuni dettagli al server: ['inventory', 'mac']</pre>	26-05-2021
TC-008	Passato	<p>I dati dell'interfaccia grafica corrispondono a quelli del database:</p>  <p>Modificare i dati li aggiorna anche nel database (l'attributo "safe" cambia):</p> 	21-05-2021

TC-009	Passato	<p>I dati vengono salvati anche dai Butlers fintanto che questi sono attivi, perché anche eliminandoli dal database vengono ristabiliti:</p>   <p>I dati sono stati inviati dal Butler al server e salvati nuovamente nel database:</p> 	21-05-2021
TC-010	Passato	<p>Il Butler riceve le modifiche, poiché anche alla disconnessione dal server riesce a far ristabilire le informazioni nel database</p>	21-05-2021
TC-011	Passato	<p>Durante la fase di apprendimento, le nuove connessioni sono aggiunte come sicure:</p> 	26-05-2021

TC-012	Passato	<p>Durante la fase di analisi, le nuove connessioni sono aggiunte come sospette:</p> <pre> 39: Object proc: "svchost.exe" status: "SYN_SENT" > source: Array > dest: Array proto: "https" safe: false </pre> <p>La connessione, non presente inizialmente nel database, viene aggiornata nell'interfaccia grafica:</p>  <p>Viene anche salvata con lo stato corretto nel database:</p> <pre> 39: Object proc: "svchost.exe" status: "SYN_SENT" > source: Array > dest: Array proto: "https" safe: false > 40: Object > 41: Object > 42: Object > 43: Object proc: "ssh.exe" status: "SYN_SENT" > source: Array > dest: Array 0: "8.8.8.8" 1: 22 proto: "ssh" safe: false </pre>	21-05-2021
TC-013	Fallito	<p>La gestione completa da interfaccia grafica del modello standard non è stata implementata per mancanza di tempo e di una corretta progettazione. Rimane comunque possibile creare e modificare il modello dal database, mentre dall'interfaccia questo può essere applicato ai singoli clients</p>	19-05-2021

TC-014	Passato	<p>Applicare il modello standard modifica le connessioni dei client:</p>  <p>Le connessioni modificate corrispondono a quelle del modello definito:</p> 	21-05-2021
TC-015	Passato	<p>Il server sa che il Butler è offline, e ritorna un messaggio di errore:</p> 	21-05-2021
TC-016	Passato	<p>Il server si accorge che il Butler è andato offline inaspettatamente, e gestisce il caso ritornando un messaggio di errore:</p> 	21-05-2021
TC-017	Passato	<p>I Butlers gestiscono i comandi dei moduli attivi:</p> 	21-05-2021

Con l'estensione delle notifiche attiva, entrambi ricevono la notifica:

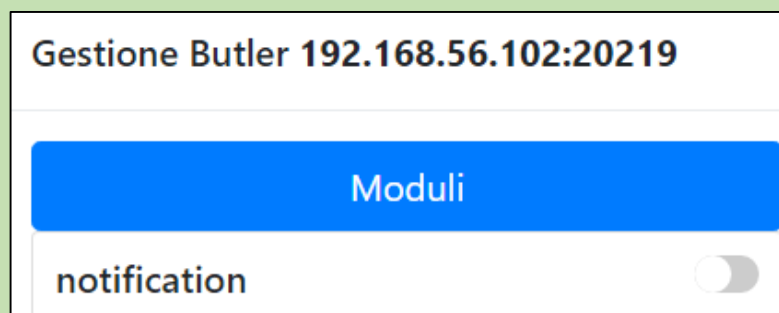


TC-018

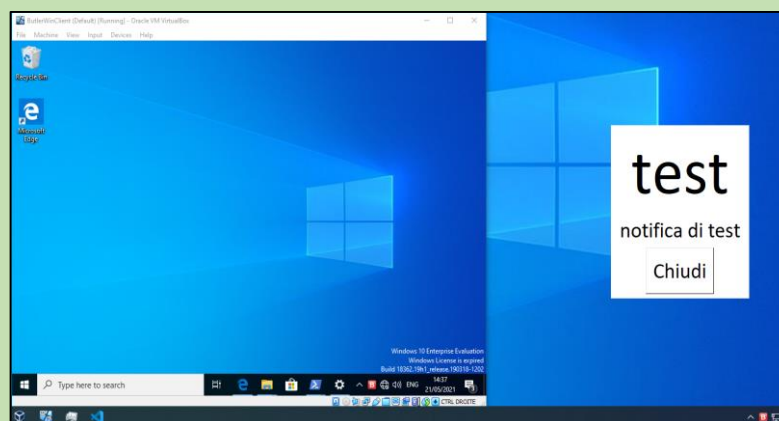
Passato

I Butlers ignorano i comandi dei moduli non attivi:

26-05-2021



La notifica non arriva all'host con l'estensione disattivata:



TC-019	Passato	<p>Il programma si accorge dei dati mancanti ed evita di aprire il modal:</p> <div data-bbox="557 327 1134 504" data-label="Image"> </div>	26-05-2021
TC-020	Passato	<p>I risultati sono uguali a quelli dei singoli test eseguiti, anche con dei clients su Linux. Ad esempio, lo stato dei moduli è rispettato:</p> <div data-bbox="485 629 1211 947" data-label="Image"> </div> <div data-bbox="485 981 1211 1352" data-label="Image"> </div>	26-05-2021
TC-021	Passato	<p>I risultati sono uguali a quelli dei singoli test eseguiti, anche con il server su Linux</p>	26-05-2021

5.3 Mancanze/limitazioni conosciute

Il programma è funziona senza apparenti problemi, e **non ci sono errori noti** nella versione finale nell'ambiente di test usato. Dato il tempo ridotto e gli ostacoli incontrati, però, dal punto di vista del codice è constatabile una **minore solidità** generale: ad esempio, in alcuni punti ci sono delle semplici stringhe usate in condizioni o come indici. Sono state limitate al minimo, ma purtroppo alcune sono risultate necessarie per trovare soluzioni rapide a problemi dalla bassa importanza. Questi parametri hard coded simboleggiano una cura minore nei dettagli e sono dovuti, appunto, unicamente alla mancanza di tempo, ma andrebbero rimossi e organizzati appena possibile.

I programmi hanno incontrato nuove scelte di sviluppo, che hanno sia l'obiettivo di adempiere alle richieste del progetto, sia quello di rendere i software ancora facilmente estensibili con ulteriori moduli. Diventa perciò sempre più necessario l'uso di **test case** appositi per ogni funzione dei programmi, al fine di garantirne la stabilità generale anche in seguito ad eventuali altre modifiche.

La limitazione più importante è la **gestione del modello standard** delle connessioni. Si tratta di un concetto mal interpretato durante l'analisi del QdC, quindi la sua implementazione non risulta molto curata, ma non è nemmeno del tutto assente. Come già descritto, è possibile gestirlo dal database ma dal centro di controllo può solo essere applicato.

Un'altra limitazione è l'uso di IP e porta come identificativi dei Butler in molte parti del programma server. Si tratta di una scelta coerente con lo scorso progetto, ma che necessiterebbe di essere adattata dopo l'implementazione dei moduli dell'inventario e del comportamento, dato che questi sono strettamente legati agli indirizzi MAC dei computer.

Dopo aver constatato la lentezza nell'attivazione di alcuni server flask, è necessario considerare l'idea di cambiare la metodologia di gestione delle threads almeno in parte, per migliorare l'autenticazione tra client e server e anche tutti gli altri componenti asincroni che potrebbero beneficiarne in futuro.

Infine, rimangono le mancanze del progetto precedente, dato che non erano presenti come punto da correggere tra i requisiti: il collegamento all'active directory, una migliore gestione dei permessi all'installazione, dei controlli di validità delle configurazioni, l'accesso dal centro di controllo ai dati dei Butlers non connessi e molti altri piccoli miglioramenti soggettivi che solo un reale utilizzatore o una persona esterna allo sviluppo del programma potrebbe vedere.

6 Consuntivo

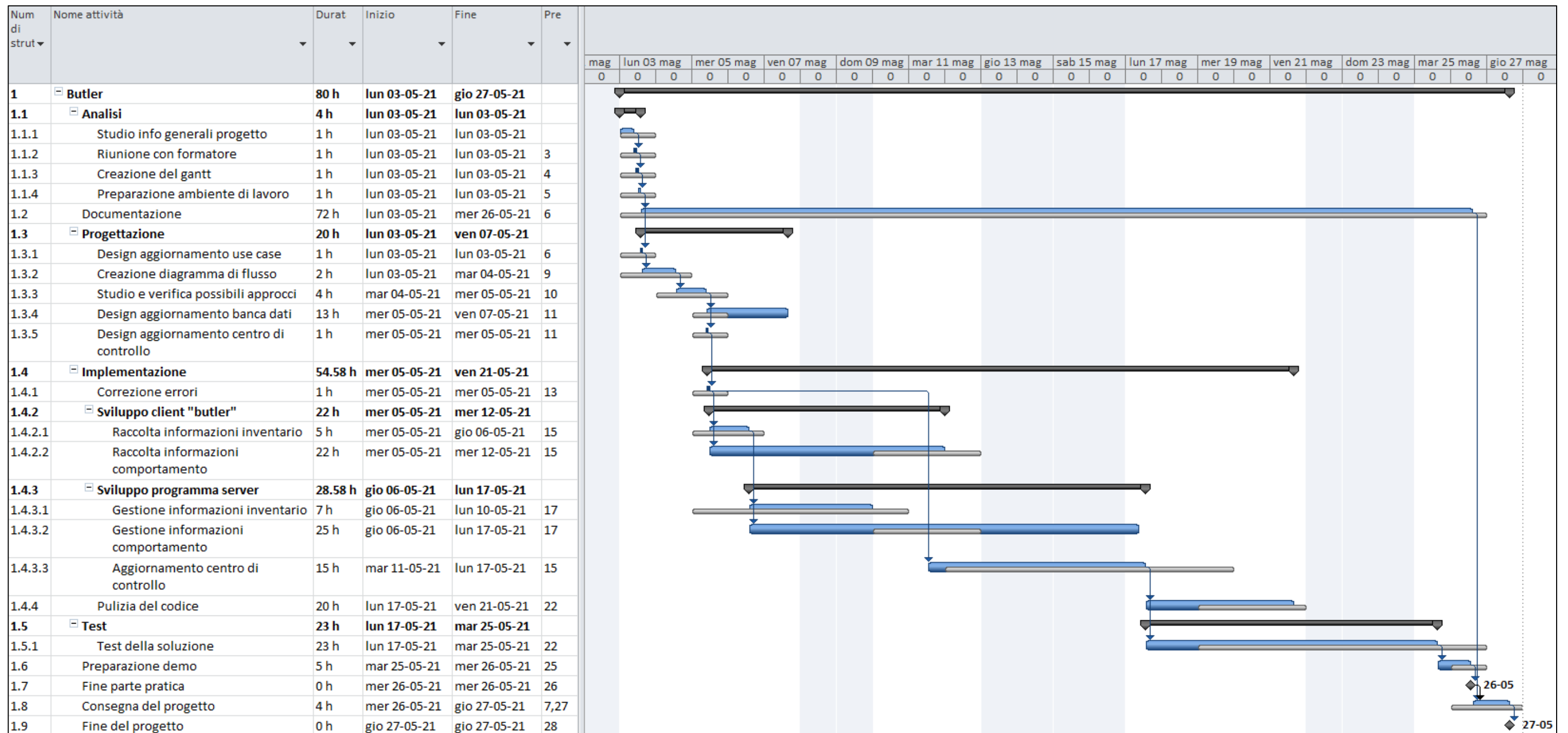


Figura 17 - Il diagramma di gantt consuntivo

I rettangoli grigi rappresentano la pianificazione preventiva, e permettono di distinguere facilmente l'aspettativa contro il tempo impiegato realmente. Il gantt è risultato piuttosto fedele fino alla progettazione, mentre l'implementazione ha da subito visto alcuni cambiamenti per adattarsi agli strumenti usati: la semplicità di psutil ha permesso di sviluppare i moduli client in contemporanea, per poi passare al server solo in seguito; come si può notare a partire dall'attività 1.4.2, le fasi che avrebbero dovuto essere svolte in parallelo **sono state scambiate**. La gestione delle informazioni del comportamento ha richiesto più tempo a causa della difficoltà avute con le connessioni e il database, quindi anche la fase di progettazione del modello dei dati si è allungata. Il tempo è stato recuperato con l'aggiornamento del centro di controllo che è risultato leggermente più breve del previsto. È stato possibile anticipare le fasi di debug e di test di alcune ore rispetto alla pianificazione. In questo modo è stato possibile continuare la documentazione senza altre attività in parallelo nelle ultime ore.

7 Conclusioni

7.1 Sviluppi futuri

Prima di qualunque aggiunta, i programmi dovrebbero subire una fase di consolidamento delle ultime modifiche apportate. Il passaggio più importante è la modifica di ogni indice di dizionario e ogni loop che sfrutta ancora l'identificativo ip:porta per supportare il più sicuro e attualmente più coerente **MAC address**. Questa modifica avrebbe effetto anche sull'interfaccia grafica. In seguito, di conseguenza, dato che i Butlers non sarebbero più riconosciuti dall'IP e che ora gli host sono meglio rappresentati dalle loro macchine fisiche, la ricerca dei destinatari delle notifiche potrebbe essere migliorata supportando non solo IP e sottoreti, ma anche il MAC, il nome utente o qualunque altro parametro dei Butlers come filtro.

Altre modifiche interessanti, seppur secondarie, comprendono l'aggiunta di più parametri disponibili da linea di comando all'avvio dei programmi, un miglioramento del centro di controllo con possibilità di vedere e modificare i dati dei Butlers offline, la possibilità filtrarli e ordinarli e l'aggiunta di diversi altri parametri dell'inventario; non sarebbe complicato aggiungere addirittura una barra di ricerca collegata a psutil per ricavare i risultati di qualsiasi suo metodo. Per quanto riguarda i comandi da parte del server, invece, dato che ora è possibile inviare dati generici si potrebbe aggiungere la possibilità di modificare in maniera remota anche le configurazioni, aggiungendo quindi una fase di scrittura del file JSON.

Come nello scorso progetto, con l'aumento delle funzionalità del programma, diventa sempre più importante l'uso di tecnologie che semplifichino la gestione di tutti i componenti: in particolare, sembra sempre più necessario l'uso di un **framework JavaScript** come Vue.js o AngularJS per gestire l'interfaccia grafica del server, e di una classe come **ThreadPoolExecutor** per una migliore gestione delle threads. Infine, per i servizi REST di flask sarebbe necessario usare un server web con un programma apposito come apache o nginx per sostituire le funzioni interne, consigliate solo per il debug e dalle scarse prestazioni.

7.2 Considerazioni personali

Poter continuare a lavorare sul codice sviluppato personalmente nel progetto precedente è stata una sorpresa positiva, dato che ero piuttosto fiero del lavoro svolto. A differenza di quello che mi sarei aspettato, non è stato per nulla difficile aggiungere nuove parti ai programmi essendo già molto stabili e con pochissimi problemi riportati dalla vecchia implementazione. I requisiti si sono rivelati adatti al tempo fornito. Purtroppo, ho sottovalutato l'uso che avrei dovuto fare del database: il salvataggio delle connessioni per ogni computer ha compromesso il resto del progetto a causa della sua **eccessiva complessità**. La scarsa conoscenza di MongoDB e la mancanza di una progettazione adeguata a riguardo hanno reso ancora più difficile la gestione di un dato (le connessioni di rete) già complesso di base. A causa della sua natura e, nuovamente, del poco tempo a disposizione, è stato difficile testare adeguatamente la sua implementazione.

I due moduli presi singolarmente non avrebbero richiesto troppo lavoro: l'unione dei due, invece, ha richiesto un impegno decisamente maggiore (nonostante non fosse fondamentale) al fine di studiarli, isolarne la logica e implementare una soluzione che prendesse in considerazione non solo questi due casi, ma anche tutti quelli analoghi possibili. A causa della mancanza di risorse, questo è stato l'unico impegno "extra" preso non necessario.

Considerando tutte le mancanze date dal tempo a disposizione limitato, non sono completamente soddisfatto del risultato ottenuto. Si è trattato del primo progetto nel quale non ho potuto dimostrare una completa gestione delle richieste: ho lasciato in sospeso alcuni punti dei quali sono perfettamente a conoscenza ma che non ho avuto modo di trattare adeguatamente. Anche al di fuori dell'implementazione ci sono migliorie varie da poter apportare al metodo di lavoro, a cominciare dalla pianificazione: mi sono reso conto che, probabilmente, **il metodo waterfall non è il più adatto** per progetti del genere. Non avendo mai effettivamente lavorato con altre tipologie di pianificazioni ho dato per scontato l'uso di questo, ma dopo un'analisi oggettiva del lavoro effettuato ritengo che il modello **iterativo/evolutivo** sarebbe stato più adatto vista la libertà maggiore nella gestione delle fasi. La progettazione è stata proporzionata rispetto al tempo fornito, ma avrebbe potuto essere **più esaustiva**: alcuni schemi sono stati creati di fretta, non rappresentano il lavoro che sarebbe poi stato effettuato e, al contrario dello scorso progetto, alcuni non hanno fornito alcun aiuto; sono piuttosto risultati un peso da dover costantemente aggiornare e possibilmente non stravolgere durante l'implementazione.

Una competenza che ho avuto modo di approfondire e di scoprire meglio è l'uso del **debugger** (in questo caso quello di visual studio code): inizialmente non ero convinto della sua utilità considerando che, avendo due programmi che scambiano dati al di fuori del codice, alcuni dati vengono inviati e "sfuggono" al controllo, o necessitano di viaggiare senza interruzioni per non creare malfunzionamenti a causa delle richieste scadute. Ho constatato che la desincronizzazione della comunicazione all'inserimento di un breakpoint, in realtà, non causa sempre problemi di funzionamento, perciò la funzione si è rivelata più utile del previsto.

Il risultato del progetto compone **un'utile aggiunta** al sistema Butler. Ha permesso di ragionare su un codice già scritto e verificarne l'effettiva modularità, permettendo di concludere, in seguito alle modifiche effettuate, la coerenza delle scelte di design prese in precedenza. Rimane un software ancora molto "teorico" la quale reale efficacia andrebbe testata in un ambiente reale e la quale utilità è forse discutibile. Il percorso è comunque risultato molto utile dal punto di vista didattico e, anche se potrebbe non trovare un'applicazione pratica, si è trattato di un'idea stimolante che potrebbe essere riproposta o ripresa in futuro.

8 Glossario

Termine	Spiegazione
AD	L'Active Directory, spesso riferita a quella di Microsoft, è un insieme di servizi per la gestione di reti di computer
Agent	Un software autonomo che esegue operazioni in background senza necessitare il controllo dell'utente. Solitamente per permettere a qualcuno di raccogliere informazioni su un sistema
AJAX	Asynchronous JavaScript and XML è una tecnica di sviluppo web per inviare richieste asincrone ad un server
API	Application Programming Interface rappresenta le procedure che permettono un interfacciamento tra sistemi o applicazioni delle quali non è necessario conoscere il funzionamento specifico per usufruirne (es. librerie).
Butler	Dall'inglese "maggior-domo", è il nome dato al servizio (o agent) che sarà in esecuzione sui clients
Collection	È il corrispettivo delle tabelle in MongoDB ("collezione" in italiano)
Dizionario python	È una serie di dati identificati da stringhe personalizzate, al contrario della lista che usa dei numeri in sequenza
ER	Sta per Entità Relazione, e rappresenta un modello di rappresentare i dati tramite, appunto, delle entità, degli attributi e le relazioni che questi hanno tra loro. È comunemente usato per rappresentare schematicamente i database
Framework	Una struttura concreta o astratta che funge da base per un software e offre dei metodi e delle procedure standard per lo sviluppo
Front-end	La parte di un software con la quale interagiscono gli utenti (di solito una GUI)
Full-duplex	Indica una trasmissione bidirezionale simultanea ⁷ tra due componenti in rete
GUI	Con Graphical User Interface si intende un'interfaccia che permette ad un utente di interagire comodamente tramite icone con un software, al posto di usare unicamente comandi da terminale
HTTP	L'HyperTextual Transfer Protocol è il principale sistema di trasmissione di informazioni in applicazioni web
JSON	JavaScript Object Notation è un formato standard e interpretabile dall'uomo per la memorizzazione di dati. È indipendente dal linguaggio di programmazione
JWT	I JSON Web Token sono uno standard per trasmettere informazioni in modo sicuro poiché firmate
Lista python	È un array standard; una lista di valori di qualsiasi tipo. A differenza del dizionario, gli indici sono numeri

⁷ <https://iate.europa.eu/entry/result/1592645/all>, IATE - Entry ID 1592645, visitato il 10-05-2021

MAC Address	Sta per Media Access Control Address, ed è un identificativo univoco solitamente rappresentato in esadecimale che ogni scheda di rete deve possedere
Modulare	In questo caso, si intende che il programma possa permettere facilmente l'ampliamento senza dover subire ingenti cambiamenti
NoSQL	Un database NoSQL (anche detto “non relazionale”) memorizza i dati in modo diverso da quelli SQL, dato che non usa delle relazioni
PID	Process ID rappresenta l'identificativo assegnato ai processi in molti sistemi operativi
Polling	Si tratta di una tecnica usata da alcuni software che consiste nel leggere di continuo lo stato di un componente esterno in attesa che questo diventi cambi stato
REST	Representational State Transfer è un'architettura basata su HTTP per ricavare risorse in base ad URLs
SHA-2	Secure Hash Algorithm 2 è un insieme di algoritmi di hashing
SPA	Single Page Application è una metodologia di sviluppo che diminuisce i ricaricamenti completi delle pagine web
Stateless	Un protocollo stateless implica che il server non memorizzi nessun identificativo della sessione, quindi le richieste sono indipendenti tra di loro
TCP	Transfer Control Protocol è un protocollo sicuro ed affidabile per lo scambio di dati con controllo degli errori. È opposto a UDP
UDP	User Datagram Protocol è un protocollo veloce per lo scambio di dati che non necessitano di assoluta integrità. È opposto a TCP

9 Bibliografia

9.1 Sitografia

- <https://www.netguru.com/blog/python-pros-and-cons>, *Python Pros and Cons | Netguru Blog on Python*, visitato il 03-05-2021
- <https://stackoverflow.com/questions/38103690/get-system-informationcpu-speed-total-ram-graphic-card-model-etc-under-window>, *python - Get system information(CPU speed-Total RAM-Graphic Card model etc.) under Windows - Stack Overflow*, visitato il 04-05-2021
- <https://stackoverflow.com/questions/3993731/how-do-i-access-netstat-data-in-python>, *How do I access netstat data in Python? - Stack Overflow*, visitato il 04-05-2021
- <https://psutil.readthedocs.io/en/latest/>, *psutil documentation — psutil 5.8.1 documentation*, visitato il 04-05-2021
- <https://www.section.io/engineering-education/what-is-md5/>, *What Is MD5 and Why Is It Considered Insecure? | Section*, visitato il 07-05-2021
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>, *HTTP response status codes - HTTP | MDN*, visitato il 12-05-2021
- <https://serverfault.com/questions/204150/sometimes-powershell-stops-sending-output-until-i-press-enter-why>, *Sometimes PowerShell stops sending output until I press enter. Why? - Server Fault*, visitato il 20-05-2021
- <https://stackoverflow.com/questions/4794244/how-can-i-create-a-copy-of-an-object-in-python>, *oop - How can I create a copy of an object in Python? - Stack Overflow*, visitato il 25-05-2021

9.2 Indice delle figure

Figura 1 - Lo schema degli use case dei due programmi	7
Figura 2 - Il diagramma di gantt preventivo	8
Figura 3 - Il confronto delle connessioni in dettaglio	14
Figura 4 - Il modello dei dati	15
Figura 5 - Il mockup dell'interfaccia	16
Figura 6 - L'UML del programma server	17
Figura 7 - L'UML del programma client	18
Figura 8 - Il diagramma di sequenza	20
Figura 9 - Lo schema dei componenti	21
Figura 10 - Il diagramma delle classi del programma client	25
Figura 11 - Il diagramma delle classi del programma client	31
Figura 12 - La modifica sbagliata senza la chiave aggiuntiva	34
Figura 13 - Un esempio di modifica corretta.....	34
Figura 14 - L'array originale	34
Figura 15 - Lo stile dei Butlers connessi.....	36
Figura 16 - Il modal di dettaglio	38
Figura 17 - Il diagramma di gantt consuntivo	55

10 Allegati

10.1 Fisici

- Abstract
- Quaderno dei Compiti
- Diari di lavoro
- Manuale di installazione e uso

10.2 In Gitsam

- Codice sorgente dei due programmi
- Diagramma di gantt preventivo
- Diagramma di gantt consuntivo
- Diagramma dei componenti
- Diagramma use case
- Diagramma di sequenza REST
- Diagramma di confronto delle connessioni
- Diagramma di flusso Butler
- Diagramma di flusso server
- Diagramma modello dei dati
- Design modifica pannello di controllo
- Diagramma delle classi Butler
- Diagramma delle classi server
- Documentazione HTML del codice