Progetto di informatica 3B
TITLE: AdaptiveHome

Jacopo Rodeschini 1046083
Department of Computer Science
Università degli studi Bergamo
Dalmine, 24044
j.rodeschini@studenti.unibg.it

Contents

Delivery of Informatica 3B (Algoritmi) course project 2020, Dalmine, Italy.

List of Figures

List of Tables

1 Introduzione e System Overview

1.1 Descrizione

AdaptiveHome è un progetto nato per ampliare le potenzialità di vivere in un'abitazione automattizzata adattandosi allo stile di vita dell'utente. L'utente sarà parte di questo processo di integrazione: AdaptiveHome si occuperà della gestione dei segnali di controllo mentre le implementazioni fisiche degli attuatori (lampade / contatti) sono lasciate all'utente. E' lasciata piene libertà all'utente di utilizzare la tecnologia hardware ritenuta più adatta alle sue esigenze come: Raspberry pi, esp, Arduino o addirittura costruirsi lui stesso delle schede di prototipazione embedded. Quello che rimane costante è l'intefaccia con cui AdptiveHome comunica con i

vari dispostivi e i diversi utenti. L'applicazione AdaptiveHome si basa su tecnologie di cloud computing e databases distribuiti (cdn) ed è sviluppata e mantenuta tramite un processo agile AMDD. L'applicazione sarà accessibile agli utenti previa registrazione attraverso il browser google-chrome. Gli utenti possono aggiungere nuove funzioni e nuovi controlli per aggiungere nuovi dispositivi da monitorare. I controlli che l'applicazione può gestire sono: *Routine* e *Fifo*: i primi sono dei controlli periodici che si ripetono con un periodo definito dall'utente (per esempio l'impianto di irrigazione che deve irrigare una volta al giorno), mentre i comandi *Fifo* sono comandi rapidi per attuare un'azione in real-time (come per esempio accendere una lampadina). La gestione delle due diverse tipologie è diversa, in particolare le *Routine* vanno a comandare delle funzioni hardware quando scade un timestamp e se "triggerate" vanno aggiornati i valori sulla base del "periodo" di esecuzione (per esempio ogni giorno), mentre per le *Fifo* è necessario assicurarsi che il comando venga eseguito entro un certo tempo, denominato *threshold*, da quando vengono inserite nel sistema. Se l'istante di esecuzione supera questa soglia di *threshold* il comando viene scartato e segnalato un warning. In ogni caso, sia per le *routine* che per le *fifo*, l'utente rimane in modalità pull e continua a chiedere lo stato dei segnali di controllo all'applicazione, in particolare chiede se ci sono delle *Routine* per le quali è scaduto il timestamp e se ci sono dei comandi *fifo* da eseguire che non hanno ancora superato la *threshold*. Il sistema non sa se il comando, una volta letto, sia stato effettivamente eseguito (ovvero abbiamo comandato un'uscita fisica I/O) per cui fa l'ipotesi che quando il comando viene letto dell'applicazione esso sia anche eseguito (sarà compito dell'utente finale assicurarsi dell'effettiva implementazione fisica). Infine, per non ricevere troppe richieste da parte degli utenti è stata inserita una politica di accettazione della richiesta che mira a limitare il numero di richieste che l'utente può operare.

1.2 Topologia del sistema

In Figura 1 viene riportato il Topology Diagram nel quale di evidenzio come il sistema sia composto da diverse tipologie di dispositivi: il Terminale Utente (browser), l'assistente vocale, un dispositivo IOT, il server applicativo e il server per l'utilizzo di servizi terzi (come per esempio il DBMS). Sul server è implementato il programma che si occupa della business logic. In particolare l'applicazione si occupa di routing delle pagine web richieste dall'utente tramite il dominio AdaptiveHome/. Il dialogo client-server avviene seguendo le regole previste dal protocollo HTTPS. I compiti di autenticazione e gestione dei databases sono delegati ai servizi di terze parti tramite l'utilizzo di APIRestful definite dal provider del servizio. Infine il server applicativo e l'IOT-device si scambiano dati attraverso delle APIRestful messe a disposizione dal server applicativo.

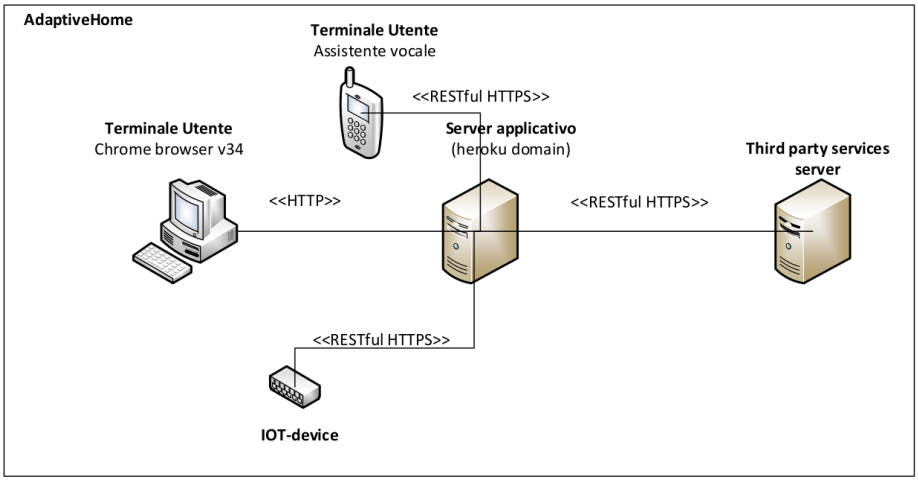


Figure 1: AdaptiveHome: Topology Diagram

1.3 Attori Coinvolti

In questa prima fase di analisi sono stati identificati i possibili attori con cui l'applicazione deve interagire per definire meglio il conteso di lavoro del sistema. In particolare sono stati definiti i seguenti attori:

- **Utente**: Utente con competenze digitali e volte alla "fabbricazione elettronica" (*Maker*) registrato alla piattaforma AdaptiveHome.
- **DBMS**: Piattaforma per integrare l'applicazione in un contesto di cloud storge e cloud computing. Vengono inoltre delegate al DBMS le funzioni di autenticazione da parte degli utenti.
- **Assistente vocale**: Applicazione di terze parti che può interagire con l'applicazione. *Nota: al momento, per la prima versione rilasciata, non è stato implementato l'assistente vocale ma è stato assicurata la piena integrazione dell'assiste vocale al sistema nelle prossime release.*
- **IOT-device**: Piattaforma scelta dall'utente per implementare le funzioni fisiche. E' richiesto che sia connessa alla rete internet e che implementi la chiamate all'API per poter comunicare con l'applicazione.

1.4 Dispositivi

Successivamente all'analisi degli attori sono stati identificati i seguenti dispositivi che faranno parte integrante dell'applicazione:

- **Client**: Un terminale connesso alla rete internet con cui l'utente può accedere alla Dashboard di AdaptiveHome.

- **IOT-device**: Piattaforma scelta dall'utente per implemantare le funzioni fisiche, è richiesto che sia connessa alla rete internet e che implementi l'interfaccia per poter comunicare con l'applicazione. Invia dati (nei limiti consentiti) e riceve segnali di controllo per gli attuatori.

1.5 Tecnologie utilizzate, software di sviluppo e progettazione

Per lo sviluppo del progetto si è fatto uso di diversi software e framework, in particolare, per la scrittura del codice relativo al backend/frontend si è utilizzato l'ide *atom* . Per la simulazione delle API e delle query implementate sul database è stato utilizzato il comando da terminale *curl*.

atom: è un ambiente di sviluppo integrato (IDE) utilizzato nella programmazione. In particolare è stato utilizzato per l'implementazione di file .js, .ejs .css. Oltre a possedere funzionalità "out of the box" molto comode per lo sviluppo software, è estensibile con migliaia di pacchetti relativi ad ogni linguaggio.

1.5.1 Linguaggi

Per lo sviluppo del progetto sono stati utilizzati diversi linguaggi, l'applicazione si avvale di varie tecnologie per il suo funzionamento: javascript, ejs e css.

Javascript: è stato usata sia in backend che in frontend per rendere interattiva l'applicazione. In backend è sto usato il framework nodejs per realizzare il server tramite javascript.

ejs: è un formato di file per la realizzazione dei template della web application. In particolare è possibile insire snippet di codice all'interno di un documento strutturato come html. Per fare il parser dei documento è stato necessario aggiungere un system-engine per elaborare questo tipo di file.

jQuery3 (libreria): è stata utilizzata per manipolare i dati lato client e rendere resposive le pagine html. Inoltre è utilizzata per la gestione di timestamp, il local storage e la comunicazione con il server per la richiesta dei dati.

Versioning del Codice: per tenere traccia delle modifiche del codice è stato utilizzato *git* mentre per condividere e collaborare al progetto è stato usata la web application *github*. Per tenere traccio delle issues è stato utilizzato *github* e *google-keep*.

1.5.2 Tecnologie scelte

In questa sezione vengono riportate in forma tabellare le tecnologie scelte e le relative caratteristiche per le varie componenti, anche in questo caso risulta critico solo in server applicativo in quanto le altre compoenti si trovano collocate presso strutture terze proprietarie.

Table 1: Tecnologie scelte.

Device	Tecnologia
Server Applicativo	Sul server sono state adottate diverse tecnologie in base alle componenti che devono essere realizzare, in particolare si è scelto di procedere utilizzando (1) *nodejs* per realizzare il server. Per l'esecuzione del server su una macchina (host) è stato utilizzato il (2) framework *http*. Per la realizzazione dell'applicazione e stato adottato il (3) framework *express* che permette di concentrarsi più sull'application logic piuttosto che sul codice effettivo e permette un grado di flessibilità molto elevato, lasciando di fatto al programmatore l'onere di progettare tutte le componenti di una web application. Per la realizzazione delle pagine html si è optato per html5,css3, javascript e bootstrap. Come (4) html engine è stato adottato *EJS* (Embedded JavaScript Templating) che permette la creazione di template HTML facilitando l'inserimento dei dati all'interno del tamplate creato. Per la gestione delle date/timestamp è stato utilizzato la (5) libreria *moment.js*, mentre per la gestione delle dipendenze del progetto è stato utilizzato il (6) package manager di default per nodejs, *npm*.
Databases	Il databases è implentato attraverso il componente firebase *Cloud Storage*, di tipo noSQL particolarmente indicato per applicazioni web e per la piena compatibilità di paradigma con javascript e nodej. Inoltre risponde pienamente alle caratteristiche di scalabilità in quanto può allocare o deallocare risorse in funzione del numero di richieste. Per mantenere bassi i costi in fase di produzione e necessario mettere un limite all'accesso al databases. Inoltre è progettato per migrare il sistema verso il paradigma *server-less* attraverso l'uso del componente firebase *cloud-functions*.
Autenticazione	La funzione di Autenticazione è delegata alla componente firebase *Authentication* perchè permette una gestione ottimale degli utenti tramite funzioni di *recupero password*, *autenticazione a due fattori*, *registrazione attraverso l'uso di diversi provider* (github, gmail e altri), *verifica della mail e numero di telefono* e altre funzioni per la gestione degli utenti disponibili per il manager del database. Queste funzioni sono rese necessarie per garantire la massima esperienza d'uso all'utente.
Assistente Vocale*	Per l'assistente vocale è stato scelto di utilizzare *google-assistant* perfettamente integrato nella tecnologia firebase e pienamente compatibile con il framework nodejs.

2 Requisiti funzionali e analisi dei casi d'uso

In seguito alla fase iniziale di analisi dei requisiti sono stati definiti i casi d'uso e i requisiti funzionali necessari per definire come avviene interazione con l'utente sia per specificare le funzioni necessarie all'applicazione per fornire le funzionalità proposte.

2.1 User-Cases Diagram

I casi d'uso sono riportati nell'*User-case Diagram* mostrato in Figura 2 dove è evidenziato come gli attori, identificati al Paragrafo 1.3, interagiscono con sistema informativo.

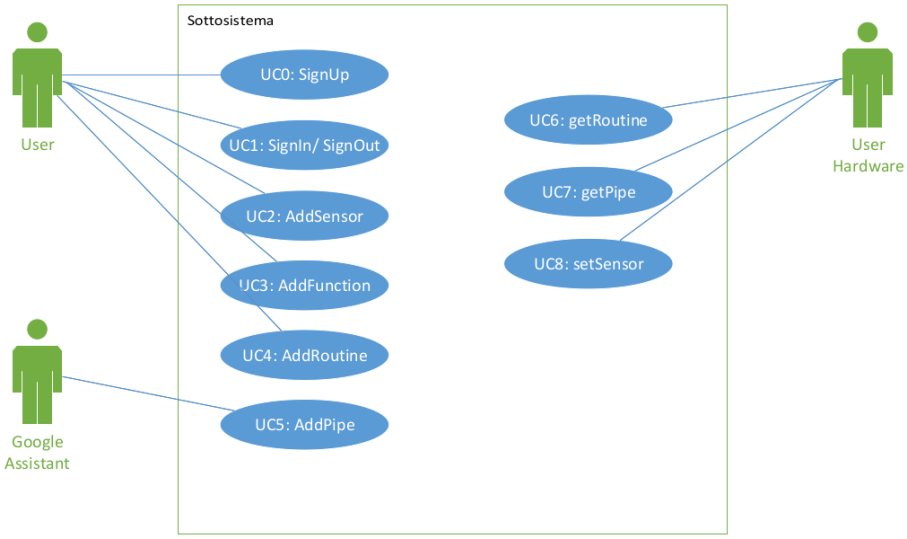


Figure 2: AdaptiveHome: User-case Diagram

2.2 User-Cases Summary

In questa fase dell'analisi sono stati ricavati 8 casi d'uso che implementano le funzionalità principali dell'applicazione. I casi d'uso sono illustrati in tabella nella forma: *Nome*, *ID*, *Tipo*, *Priorità*, *Rischio* e *Breve descrizione* (al Paragrafo 4.1 sono riportati i casi d'uso in modo più rigoroso e dettagliato).

Table 2: User-Cases Summary.

Nome	Id	Tipo	Priorità	Rischio	Descrizione
SignUp	UC0	Funzionale	Alta	Alto	Registrazione dell'utente presso la piattaforma.

Continued on next page

Continued from previous page

Nome	Id	Tipo	Priorità	Rischio	Descrizione
SignIn	UC1	Funzionale	Alta	Alto	LogIn / LogOut dell'utente presso la piattaforma.
AddSensor	UC2	Funzionale	Bassa	Basso	L'utente aggiunge un nuovo sensore di cui vuole visualizzare i dati
AddFunction	UC3	Funzionale	Bassa	Basso	L'utente aggiunge un nuovo segnale di controllo degli attuatori attraverso la dichiarazione di una nuova funzione (questa può essere pilotato da delle *Routine* o dai comandi *Fifo*).
AddRoutine	UC4	Funzionale	Bassa	Basso	L'utente aggiunge una nuova routine di controllo su una determinanta funzione.
AddFifo	UC5	Funzionale	Bassa	Basso	L'utente o Google Assitant aggiunge un nuovo controllo istantaneo.
getRoutine	UC6	Funzionale	Bassa	Basso	L'utente richiede i controlli delle *routine* all'applicazione.
getFifo	UC7	Funzionale	Bassa	Basso	L'utente richiede i controlli *fifo* all'applicazione.
getSensor	UC8	Funzionale	Bassa	Basso	L'utente invia i dati di un sensore precedentemente registrato.

2.3 Requisiti Funzionali

Vengono ora descritti i requisiti funzionali dell'applicazione, ovvero le funzioni che devono essere svolte dall'applicazione ma non direttamente accessibili agli attori coinvolti. Sono riportate in tabella nella forma: *Name*, *ID* e *Descrizione* (al Paragrafo 4.2 sono descritti i requisiti funzionali in modo più rigoroso e dettagliato).

Table 3: Requisiti Funzionali.

Name	**Id**	**Descrizione**
Routine Manager	F01	Quando i dati relativi alle routine sono estratti dal database vengono elaborati dal server. Ogni routine è associata ad una funzione e specifica il timestamp in cui cambiare stato, il valore corrente della funzione e il valore futuro. Quanto il timestamp scade i valori vengono aggiornati e in seguito inviati al client e salavati nel database.
Routine Count	F02	Verifica la frequenza di chiamata delle API per la lettura delle routine. Se il numero di richieste al secondo supera una certa soglia viene generato un warning (una possibile soglia potrebbe essere (6 request)/ora).I valori verranno mostrati tramite l'interfaccia utente.
Fifo Manager	F03	Quando i dati relativi ai comandi rapidi sono estratti dal database vengono elaborati del server. Ogni comando è associato ad una funzione e conserva il timestamp di quando è stato creato e il valore da assegnare alla funzione. Quando il valore del comando rapido viene letto (tramite un' apposita api: .getFifo), se la differenza tra l'istante di lettura e l'istante di creazione supera una certa soglia vene generato un warning e il comando scartato. Quelli che soddisfano il vincolo sul timestamp sono inviati al client. I dati, una volta inviati, sono rimossi dal database.
Fifo Count	F04	Verifica la frequenza di chiamata delle API per la lettura delle *fifo*. Se il numeri di richieste al secondo supera una certa soglia viene generato un warning (una possibile soglia potrebbe essere (10 request)/min). I valori verranno mostrati tramite l'interfaccia utente.
Warning Manager	F05	Si occupa della gestione dei warning, fornisce per ogni utente la lista cronologica dei warning che verrano mostrati tramite l'interfaccia utente.
DashBoard Manager	F06	Si occupa della gestione dei grafici della Dashboard controllabile dall'utente, fornisce per ogni utente i valori da plottare, i grafici e le statistiche.

Continued on next page

Continued from previous page

Name	Id	Descrizione
RoutingConsensus	F07	Si occupa di dare il consenso alla risposta della chiamata dell'api ./getRouting(api-key). In questa funzione è implementata una policy specifica per autorizzare la risposta. Per una prima fase di prototipo questa funzione è realizzata come componente mock e restituisce sempre vero.
FifoConsensus	F08	Si occupa di dare il consenso alla risposta della chiamata dell'api ./getFifo(api-key). In questa funzione è implementata una policy specifica per autorizzare la risposta. Per una prima fase di prototipo questa funzione è realizzata come componente mock e restituisce sempre vero.
SensorsConsensus	F09	Si occupa di dare il consenso alla risposta della chiamata dell'api ./setSensors(api-key,id-key). In questa funzione è implementata una policy specifica per autorizzare la risposta. Per una prima fase di prototipo questa funzione è realizzata come componente mock e restituisce sempre vero.

3 Requisiti non-funzionali

3.1 Requisiti non funzionali

In tabella sono riportati i requisiti non funzionali nella forma: *ID*, *Descrizione* e *Valore (max)* considerati dell'applicazione per garantire la migliore esperienza d'uso all'utente. *(Nota: affinché sia garantita la gestione delle transizioni e la distribuzione del contenuto nei database distribuiti è necessario che si acceda al database con una frequenza massima di una volta al secondo: 1req/sec e che la dimensione del documento non ecceda 1Mb. Al momento questo condizioni non sono controllate).*

Table 4: Requisiti non funzionali.

Id	Descrizione	Valore (max)
NF0	Richiesta dei comandi delle *routine*	1 req/10min
NF1	Richiesta dei comandi rapidi *fifo*	10 req/min
NF2	Lettura dati da sensori	1 req/min
NF3	Usabilità/semplicità dell'interfaccia grafica	2 click/azione

4 Dettaglio casi d'uso

Dopo aver analizzato gli aspetti principali del progetto e definito il perimetro dell'applicazione, per poter procedere all'implementazione software dell'applicazione è stato necessario dettagliare meglio sia i casi d'uso sia i requisiti funzionali che non funzionali. In seguito sono riportati i casi d'uso descritti in modo più rigoroso e dettagliato.

4.1 User-Cases Full

In questo paragrafo viene approfondita in dettaglio la struttura dei casi d'uso, descritti precedentemente, a cui sono state aggiunte le informazioni di: *pre-condizioni*, *trigger*, *post-condizioni*, *standard process*, *alternative process* ed *exceptional process*. Queste informazioni aggiuntive risultano utili durante la progettazione e la programmazione.

Table 5: User case UCO: Registrazione degli utenti nell'applicazione.

SignUp	**UC0**
Descrizione	Registrazione degli utenti nell'applicazione tramite: mail, password.
Attori coinvolti	Utente, (Firebase) Cloud Autentications, Server applicativo
Pre-condizioni	Il sistema deve avere la connessione presso Cloud Autentications. L'utente non è gia registrato all'applicazione
trigger	L'utente vuole registrarsi presso la piattaforma
Post-condizioni	L'utente è registrato nell'applicazione e viene reindirizzato alla Dashboard
Standard process	1) Richiesta pagina AdaptiveHome/Auth/SignUp.html. 2) Completare la form proposta inserendo tutti i campi obbligatori: (mail,psw). 3)Triggerare il bottone SignUp presente nel popUp con il quale i valori sono passati al server. 5) Il server esegue una query presso *firebase* per inserire il nuovo utente. 6) Se l'esito dell'operazione da esito positivo viene inizializzata una sessione utente e generata una api-key (con cui l'utente esegue le chimate alla api). Infine l'utente viene reindirizzato presso la pagina iniziale. 7) Se l'esito è negativo viene visualizzato un messaggio di errore e viene reindirizzato alla pagina di ./SignUp.html
Alternative process	1) è possibile registrarsi tramite un account già esistente google. 2) Seguire i passaggi guidati tramite la google form.

Continued on next page

Continued from previous page

SignUp	**UC0**
Exceptional process	1) Se la pagina ./SignUp.html non è disponibile viene mostrata una pagine di errore (404). 2) Se la connessione al server firebase non è presente, viene visualizzato un messaggio di errore "*si invita a riprovare più tardi*".

Table 6: User case UC1: LogIn degli utenti nell'applicazione.

SignIn	**UC1**
Descrizione	LogIn degli utenti nell'applicazione.
Attori coinvolti	Utente, (Firebase) Cloud Autentications
Pre-condizioni	Il sistema deve avere la connessione presso Cloud Autentications. L'utente è già registrato all'applicazione.
trigger	L'utente vuole fare il logIn presso la piattaforma
Post-condizioni	L'utente è registrato nell'applicazione e viene reindirizzato alla Dashboard
Standard process	1) Richiesta pagina AdaptiveHome/Auth/SignIn.html. 2) Completare la form proposta inserendo tutti i due campi obbligatori: (mail,psw). 3) Triggerare il bottone SignIn presente nel popUp con il quale i valori sono passati al server. 5) Il server esegue una query presso *firebase* per autenticare l'utente. 6) Se l'operazione da esito positivo viene inizializzata una sessione utente e l'utente viene reindirizzato presso alla pagine desiderata. L'interfaccia utente viene personalizzata sulla base delle infomrazioni conosciute dell'utente (User-name, mail, etc...) 7) Se l'esito è negativo viene visualizzato un messaggio di errore. 8) Per fare il logOut è necessario triggerare il bottone Account 9) Triggerare il bottone logOut 10) Rimozione della sessione utente
Alternative process	1) Se è già presente una sessione utente valida, al momento di routing presso il dominio /AdaptiveHome.org l'utente sarà già loggato.
Exceptional process	1) Se la pagina ./SignIn.html non è disponibile viene mostrata una pagine di errore (404). 2) Se la connessione al server firebase non è presente, viene visualizzato un messaggio di errore "*si invita a riprovare più tardi*".

Table 7: User case UC2: L'utente vuole aggiungere un nuovo sensore.

AddSensor	**UC2**

Continued on next page

Continued from previous page

AddSensor	**UC2**
Descrizione	L'utente vuole aggiungere un nuovo sensore (stream di dati) di cui vuole visualizzare i dati.
Attori coinvolti	Utente, (Firebase) Cloud Storage
Pre-condizioni	Il sistema deve avere la connessione presso Cloud Storage. E' presente una sessione utente valide sul client.
trigger	L'utente vuole aggiungere un nuovo sensore (stream di dati). Dalla pagina ./Dasboard.html triggera il bottote Add Sensor, oppure dalla navbar Triggera il bottone Actions e successivamente Add Sensor.
Post-condizioni	Il sensore è aggiunto al database ed è possibile visualizzare i valori nella Dashboard.
Standard process	1) Richiesta pagina AdaptiveHome/AddSensor.html. 2)Completare la form proposta inserendo tutti i campi obbligatori: (name,type,descrizione (max 40 caratteri)). 3) Triggerare il bottone Complete presente nel popUp, con il quale i valori sono passati al server. 4) Il server convalida i parametri passati, se l'esisto è positivo viene generata un id-key univoca associata al sensore (prosegue al passaggio 5), se l'esisto è negativo viene mostrato un messaggio di errore, viene terminato il processo di inserimento e l'utente viene reindirizzato alla pagine ./AddSensor.html 5) Il server esegue una query presso *firebase* per creare un nuovo documento associato all'$_{\text{id}}$-key_ e all'utente. 6) Se l'operazione da esito positivo viene mostrato il messaggio "*operation completed successfully*", se l'esito è negativo viene visualizzato un messaggio di errore "*operation failed*". 8) L'utente è reindirizzato alla pagine ./AddSensor.html. 9) Il valore id-key è visibile all'utente e deve essre usato (in aggiunta all'api-key) per autenticare il sensore in fase di invio dati presso il server.
Alternative process	Nothing
Exceptional process	1) Se la pagina ./AddSensor.html non è disponibile viene mostrata una pagine di errore (404). 2) Se la connessione al server firebase non è presente, viene visualizzato un messaggio di errore "*si invita a riprovare più tardi*".

Table 8: User case UC3: L'utente vuole aggiungere un nuovo segnale di controllo.

AddFunction	**UC3**
Descrizione	L'utente vuole aggiungere un nuovo segnale di controllo in grado di pilotare un dispositivo di I/O.
Attori coinvolti	Utente, (Firebase) Cloud Storage, Server applicativo

Continued on next page

Continued from previous page

AddFunction	**UC3**
Pre-condizioni	Il sistema deve avere la connessione presso Cloud Storage. E' presente una sessione utente valide sul client.
Trigger	L'utente vuole aggiungere un nuovo segnale di controllo. Dalla pagina ./Dasboard.html triggera il bottote Add Function, oppure dalla navbar Triggera il bottone Actions e successivamente Add Function.
Post-condizioni	Il segnale di controllo è aggiunto al database ed è possibile controllare il segnale di controllo inizializzando delle nuove Routine oppure dei comandi rapidi.
Standard process	1) Richiesta pagina AdaptiveHome/AddFunction.html. 2)Completare la form proposta inserendo tutti i campi obbligatori: (name,type,descrizione (max 40 caratteri),value). 3) Triggerare il bottone Complete presente nel popUp, con il quale i valori sono passati al server. 4) Il server convalida i parametri passati, se l'esisto è positivo viene generata un id-key univoco (prosegue al passaggio 5), se l'esisto è negativo viene mostrato un messaggio di errore, viene terminato il processo di inserimento e l'utente viene reindirizzato alla pagina ./AddFunction.html 5) Il server esegue una query presso *firebase* per aggiungere la funzione definita dall'utente. 6) Se l'operazione da esito positivo viene mostrato il messaggio "*operation completed successfully*", se l'esito è negativo viene visualizzato un messaggio di errore "*operation failed*". 8) L'utente è reindirizzato alla pagina ./AddFunction.html. 9) Il valore di id-key è utilizzato dal'applicazione per identificare la funzione, sarà il valore comunicato tramite le API per notificare un cambio di stato del senglae di controllo.
Alternative process	Nothing
Exceptional process	1) Se la pagina ./AddSensor.html non è disponibile viene mostrata una pagine di errore (404). 2) Se la connessione al server firebase non è presente, viene visualizzato un messaggio di errore "*si invita a riprovare più tardi*".

Table 9: User case UC4: L'utente vuole aggiungere una nuova routine.

AddRoutine	**UC4**
Descrizione	L'utente vuole aggiungere una nuova routine per modificare il valore di un segnale di controllo (ovvero di una funzione) ad istanti regolari (ad es: attivare un elettrovalvola dell'impianto di irrigazione al 18:00 ogni 3 giorni).
Attori coinvolti	Utente, (Firebase) Cloud Storage, Server applicativo

Continued on next page

Continued from previous page

AddRoutine	**UC4**
Pre-condizioni	Il sistema deve avere la connessione presso Cloud Storage. E' presente una sessione utente valide sul client.
trigger	L'utente vuole aggiungere una nuova routine. Dalla pagina ./Dasboard.html triggera il bottote Add Routine, oppure dalla navbar Triggera il bottone Actions e successivamente Add Routine.
Post-condizioni	La routine è aggiunta al database ed è possibile visualizzare tutte le routine attive nella Dashboard.
Standard process	1) Richiesta pagina AdaptiveHome/AddRoutine.html. 2)Completare la form proposta inserendo tutti i campi obbligatori: (name,key-function*,descrizione (max 40 caratteri), period). Il valore *key-function* deve essere un *id-key* associato ad una funzione esistente (precedentemente definita dall'utente). Il valore *period* è inserito tramite un calendario 3) Triggerare il bottone Complete presente nel popUp, con il quale i valori sono passati al server. 4) Il server convalida i parametri passati, se l'esisto è positivo server esegue una query presso *firebase* per aggiungere la nuova routine associata all'utente (procede passo 5), se l'esisto è negativo viene mostrato un messaggio di errore, viene terminato il processo di inserimento e l'utente viene reindirizzato alla pagine ./AddRoutine.html. 5) Se l'operazione da esito positivo viene mostrato il messaggio "*operation completed successfully*", se l'esito è negativo viene visualizzato un messaggio di errore "*operation failed*". 8) L'utente è reindirizzato alla pagina ./AddRoutine.html.
Alternative process	Nothing
Exceptional process	1) Se la pagina ./AddSensor.html non è disponibile viene mostrata una pagine di errore (404). 2) Se la connessione al server firebase non è presente, viene visualizzato un messaggio di errore "*si invita a riprovare più tardi*". 3) Se il valore *key-functions* inserito in fase di aggiunta di una nuova routine non è associato a nessuna funzione associata all'utente si ha un errore e il processo di inserimento termina con un messaggio di errore. 4) Se il valore di *key-function* è utilizzato più volte per definire diverse routine (operazione consentita) viene aggiunto un warning nella pila dei warning.

Table 10: User case UC5: L'utente (o Google-Assistant) vuole aggiungere un nuovo comando rapido.

AddPipe	**UC5**
Descrizione	L'utente (o Google-Assistant) vuole aggiungere un nuovo comando rapido per modificare il valore di un segnale di controllo (ovvero di una funzione) (ad es: attivare un elettrovalvola dell'impianto di irrigazione adesso).
Attori coinvolti	Utente (o Google-Assistnat), (Firebase) Cloud Storage, Server applicativo
Pre-condizioni	Il sistema deve avere la connessione presso Cloud Storage. E' presente una sessione utente valida sul client.
trigger	L'utente vuole aggiungere un nuovo comando rapido. Dalla pagina ./Dasboard.html triggera il bottote Add Pipe, oppure dalla navbar Triggera il bottone Actions e successivamente Add Pipe.
Post-condizioni	Il comando rapido è aggiunto al database.
Standard process	1) Richiesta pagina AdaptiveHome/AddPipe.html. 2)Completare la form proposta inserendo tutti i campi obbligatori: (name,key-function*,descrizione (max 40 caratteri),#freshness). Il valore *key-function* deve essere un *id-key* associato ad una funzione esistente (precedentemente definita dall'utente). 3) Triggerare il bottone Complete presente nel popUp, con il quale i valori sono passati al server. 4) Il server convalida i parametri passati, se l'esisto è positivo il server assegna il valore del timestamp al campo #freshness ed esegue una query presso *firebase* per aggiungere la nuova pipe associata all'utente (procede passo 5), se l'esisto è negativo viene mostrato un messaggio di errore, viene terminato il processo di inserimento e l'utente viene reindirizzato alla pagine ./AddPipe.html. 5) Se l'operazione da esito positivo viene mostrato il messaggio "*operation completed successfully*", se l'esito è negativo viene visualizzato un messaggio di errore "*operation failed*". 8) L'utente è reindirizzato alla pagina ./AddPipe.html.
Alternative process	Nothing

Continued on next page

Continued from previous page

AddPipe	**UC5**
Exceptional process	1) Se la pagina ./AddSensor.html non è disponibile viene mostrata una pagine di errore (404). 2) Se la connessione al server firebase non è presente, viene visualizzato un messaggio di errore "*si invita a riprovare più tardi*". 3) Se il valore *key-functions* inserito in fase di aggiunta di una nuova pipe non è associato a nessuna funzione associata all'utente si ha un errore e il processo di inserimento termina con un messaggio di errore. 4) Se il valore di *key-function* è utilizzato più volte per definire diverse routine (operazione consentita) viene aggiunto un warning nella pila dei warning.

Table 11: User case UC6: L'User-Hardware richiede i controlli delle funzioni associate alle routine.

getRoutine	**UC6**
Descrizione	L'User-Hardware richiede i controlli delle funzioni associate alle routine.
Attori coinvolti	User-Hardware (client), (Firebase) Cloud Storage-Authentication, Server applicativo.
Pre-condizioni	Il sistema deve avere la connessione presso Cloud Storage. Il client dispone dell'*api-key* (generata in fase di SignUp). Il client deve poter connettersi al server applicativo. La frequenza di chiamata non deve superare un certo limite.
Trigger	User-Hardware richiede tramite la chiamata all'API ./getRoutine(api-key) i valori delle funzioni (ovvero i segnali di controllo) associate alle routine definite dall'utente.
Post-condizioni	Il client riceve i valori di (*key-function*, current-value, next-value, trigger-time) in formato json per tutte le routine definite in precedenza. [Sarà compito dell'utente implementare le funzioni a livello hardware che in base al timestamp e al next-value realizzeranno il cambio di stato per un dispositivo I/O , per esempio da HIGH (luce accesa) a LOW (luce spenta)].

Continued on next page

Continued from previous page

getRoutine	**UC6**
Standard process	1) Chiamata da parte del client dell'API ./getRoutine(api-key), pove il paramtro *api-key* è la *key* associata all'utente in fase di registrazione. 2) Il server verifica la compatibilità dell'api-key con gli utenti presenti nel databases sfuttanto il servizio firebase Authentication. 3) Se l'esito è positivo viene viene verificato il consenso ad inviare i dati al client (tramite la funzione RoutineConsensus). 4) Se vi è consenso all'invio viene fatta una quesry al databases Cloud Storage per ricostruire i dati relativa alle routing definite dagli utenti. 5) I dati sono quindi inviati in risposta alla chimata di funzione
Alternative process	Nothing
Exceptional process	1) Se il valore *api-key* inserito in fase di richiesta non è associato a nessun utente si ha un errore e il processo di richiesta dati termina (non viene inviata una risposta di errore). 2) Eventuali errori riscontrati vengono aggiunti alla pila dei warning.

Table 12: User case UC7: L'User-Hardware richiede i controlli delle funzioni associate ai comandi rapidi.

getFifo	**UC7**
Descrizione	L'User-Hardware richiede i controlli delle funzioni associate ai comandi rapidi.
Attori coinvolti	User-Hardware (client), (Firebase) Cloud Storage-Authentication, Server applicativo.
Pre-condizioni	Il sistema deve avere la connessione presso Cloud Storage. Il client dispone dell'*api-key* (generata in fase di SignUp). Il client deve poter connettersi al server applicativo. La frequenza di richiesta non deve superare un certo limite.
Trigger	User-Hardware richiede tramite la chiamata all'API ./getFifo(api-key) i valori delle funzioni (ovvero i segnali di controllo) associate alle fifo definite dall'utente.
Post-condizioni	Il client riceve i valori di (*key-function*, current-value, next-value, trigger-time) in formato json per tutte le routine definite in precedenza. [Sarà compito dell'utente implementare le funzioni a livello hardware che in base al trigger-time e al next-value realizzeranno il cambio di stato per un dispositivo I/O , per esempio da HIGH (luce accesa) a LOW (luce spenta)].

Continued on next page

Continued from previous page

getFifo	**UC7**
Standard process	1) Chiamata da parte del client dell'API ./getFifo(api-key), dove il parametro *api-key* è la *key* associata all'utente in fase di registrazione. 2) Il server verifica la compatibilità dell'api-key con gli utenti presenti nel databases sfuttanto il servizio firebase Authentication. 3) Se l'esito è positivo viene viene verificato il consenso ad inviare i dati al client (tramite la funzione FifoConsensus). 4) Se vi è consenso all'invio viene fatta una query al databases Cloud Storage per ricostruire i dati relativa ai comandi rapidi (fifo) definite dagli utenti. 5) I dati sono quindi inviati in risposta alla chiamata di funzione
Alternative process	Nothing
Exceptional process	1) Se il valore *api-key* inserito in fase di richiesta non è associato a nessun utente si ha un errore e il processo di richiesta dati termina (non viene inviata una risposta di errore). 2) Eventuali errori riscontrati vengono aggiunti alla pila dei warning.

Table 13: User case UC8: L'User-Hardware richiede di inviare i dati relativi ad un sensore.

setSensors	**UC8**
Descrizione	L'User-Hardware richiede di inviare i dati relativi ad un sensore. Il formato dei dati è di tipo JSON.
Attori coinvolti	User-Hardware (client), (Firebase) Cloud Storage-Authentication, Server applicativo.
Pre-condizioni	Il sistema deve avere la connessione presso Cloud Storage. Il client dispone dell'*api-key* (generata in fase di SignUp). Il client deve poter connettersi al server applicativo. La frequenza di richiesta non deve superare un certo limite.
Trigger	User-Hardware invia tramite la chiamata all'API ./setSensor(api-key,id-key) i valori del sensore precedentemente definito dall'utente.
Post-condizioni	Il server riceve una risposta in base all'esito di inseriemtno dei dati. I dati possono essere cumulati, ovvero avere piu valori associati allo stesso sensore. L'api supporta un solo sensore.

Continued on next page

Continued from previous page

setSensors	**UC8**
Standard process	1) Chiamata da parte del client dell'API ./setSensor(api-key,id-key), dove il parametro *api-key* è la *key* associata all'utente in fase di registrazione e *id-key* è l'identificativo del sensore generato in fase di definizione del sensore (recuperabile dalla DashBoard). 2) Il server verifica la compatibilità dell'api-key con gli utenti presenti nel databases sfuttanto il servizio firebase Authentication e verifica che id-key del sensore è definito per l'utente 3) Se l'esito è positivo viene verificato il consenso a ricevere i dati inviati dal client (tramite la funzione SensorConsensus). 4) Se vi è consenso alla ricezione viene verificata la truttura del dato: {[time: value,time:value,...]} e viene fatta una query al databases Cloud Storage per salvare i dati associati al sensore id-key. 5) Il server conferma la ricezione dei dati tramite il messaggio {Status:"OK"}.
Alternative process	Nothing
Exceptional process	1) Se il valore *api-key* inserito in fase di richiesta non è associato a nessun utente si ha un errore e il processo di invio dati termina (non viene inviata una risposta di errore). 2) Se il valore *id-key* inserito in fase di richiesta non è associato a nessun sensore si ha un errore e il processo di invio dati termina (non viene inviata una risposta di errore). 3) Eventuali errori riscontrati vengono aggiunti alla pila dei warning. 4) Se il fomrato dei dati non è conforme allo standard viene inviato un messaggio di errore: {Status:"FAILURE"}.

4.2 Requisiti Funzionali-Detail

In questo capitolo vengono definiti nel dettaglio i requisiti non funzionali descritti precedentemente. In tabella sono riportati i dettagli dei requisiti funzionali nella forma: *Descrizione*, *Funzionalità*, *Errori* (o eccezioni) e l'*Error manager* che descrive come viene gestita una possibile situazione di errore.

Table 14: Requisito Funzionale F01: verificare lo stato delle routine.

Routine Manager	**F01**

Continued on next page

Continued from previous page

Routine Manager	**F01**
Descrizione	Si occupa di verificare lo stato delle routine (funzione di triggersul database), in particolare monitora il timestamp del periodo di esecuzione (che indica quando una funzione deve cambiare stato), allo scadere del timestamp i valori vengono agiornati nel modo seguente *current-value* = *next-value*, *next-value* = *!next-value*, *timestamp* = *timestamp* + *period*. I valori di *value* indicano il valore assunto della funzione (dal segnale di controlla), mentre *period* è il periodo di ripetizione della routine.
Funzionalità	1) Update Routine Value.
Errori	Possibili errori: 1) mancata connessione al database.
Error Menager	1) delegare l'errore al warning manage specificando il codice di errore riscontrato.

Table 15: Requisito Funzionale F02: verifica che la frequenza di chimata delle routine.

Routine Count	**F02**
Descrizione	Conta quante volte avviene la chiamata delle api per la lettura dei controlli riferiti alla routine ed elabora alcune statistiche di controllo, in particolare verifica che la frequenza di chiamata sia inferiore (<=) ad una certa soglia (vedere requisiti non funzionali). Se il numero di chiamte/sec eccede la soglia massima viene generato un warning.
Funzionalità	1) Count Request routine API.
Errori	Possibili errori: 1) Mancata connessione al database. 2)Soglia richiest/sec superata.
Error Menager	1) delegare l'errore al warning manage specificando il codice di errore riscontrato.

Table 16: Requisito Funzionale F03: verificare lo stato dei comandi *Fifo*.

Fifo Manager	**F03**

Continued on next page

Continued from previous page

Fifo Manager	**F03**
Descrizione	Si occupa di verificare lo stato dei comandi *Fifo* in particolare monitora il valore di freshness (timestamp di creazione) delle *Fifo* (comando rapido) che indica quando un comando è stato creato. I comandi rapidi sono inseriti in una struttura a coda (ovvero gli elementi sono aggiunti in cascata) e quindi ordinati in base al valore di freshness. Per cui viene verificato che il primo elemento della coda (di ogni utente) non abbia una freshness strettamente superiore (>) ad un certo valore di *threshold*. La soglia di *threshold* indica il tempo limite entro cui il valore della funzione associata alla *Fifo* deve essere letto dall'IOT-device per pilotare il dispositivo fisico. Se è verificato che il valore di freshness > *threshold* il primo elemento della *Fifo* viene scartato e generato un warning, si procede quindi ad analizzare il secondo elemento. In modo iterativo si selezionano i comandi *Fifo* validi che saranno inviati in risposta all'IOT.device.
Funzionalità	1) Check freshness.
Errori	Possibili errori: 1) freshness > *threshold*. 2) mancata connessione al database.
Error Menager	1) delegare l'errore al warning manager specificando il codice di errore riscontrato.

Table 17: Requisito Funzionale F04:verifica che la frequenza di chiamata dei comandi *Fifo*.

Fifo Count	**F04**
Descrizione	Conta quante volte avviene la chiamata delle API per la lettura dei controlli riferiti ai comandi rapidi (*Fifo*) ed elabora alcune statistiche di controllo, in particolare verifica che la frequenza di chiamata sia inferiore (<=) ad una certa soglia (vedere requisiti non funzionali). Se il numero di chiamte/sec eccede la soglia massima viene generato un warning.
Funzionalità	1) Count Request *Fifo* API.
Errori	Possibili errori: 1) Mancata connessione al database. 2) Soglia richiest/sec superata.
Error Menager	1) Delegare l'errore al warning manage specificando il codice di errore riscontrato.

Table 18: Requisito Funzionale F05: gestione dei warning generati.

Warning Manager	**F05**
Descrizione	Si occupa di gestire tutti i warning generati in fase di esecuzione. Ad ogni warning è identificato da un codice univoco. I warning sono ordinati in base all'istante di creazione e possiedono una descrizione informale dell'anomalia riscontrata. Per ogni utente viene gestita la coda dei warning che devono essere mostrati all'utente, inoltre vengono elaborati i dati per estrarne delle caratteristiche quantitative come il numero totale di warning e altre informazioni.
Funzionalità	1) Agigungere un nuovo warning nel database. 2) Manage coda dei warning. 3) Estrapolare statistiche di controllo (numero totale di warning).
Errori	Possibili errori: 1) Mancata connessione al database.
Error Menager	1) Viene gestita una semplice stuttura locale dei warning pendenti (ovvero che non sono stati ancora salvati sul database). Nel momento in cui la connessione è ristabilita vengono inserite le tuple pendenti nel databases.

Table 19: Requisito Funzionale F06: costruzione della deshboard.

DashBoard Manager	**F06**
Descrizione	Contiene tutte le funzioni necessarie per la costruzione della deshboard, ad esempio la creazione dei grafici personalizzati e le statistiche degli utenti. E' stato realizato un modulo apposito per gestire la comlessità della gestione grafica, in questo modo è possibile gestire in maniera ottimale anche eventuali modifiche (o upgrade) delle singole funzioni di visualizzazione senza compromettere la businness logic. Inoltre, in questo modo è possibile ridurre i tempi di attesa di invio della pagina, gestendo il traffico dati (per la creazione della dashboard) tramite chiamate asincrone. Per cui il traffico dati inizierà non appena la pagina sarà scaricata sul browser. Poiché la quantità di dati influisce molto sui tempi di attesa da parte degli utenti, la richiesta dei dati viene fatta per un solo sensore alla volta, solo su richiesta (ad esempio clicca su $next_{plot}$). I dati sono resi persistenti nel borwser attraverso l'uso del *local storage* sul browser. I passaggi per l'invio e il salvataggio dei dati sul browser sono descritti nella sezione *protocollo di richiesta*.

Continued on next page

Continued from previous page

DashBoard Manager	**F06**
Protocollo di richiesta	1) Verificare la sessione utente (Valida -> 2, Scaduta-> 4). 2) Sessione valida: verificare che nel *local storage* ci siano dei dati per i diversi sensori (lista di oggetti che descrivono i sensori). Se ci sono dei dati, chiedere attraverso le API definite nella componente *DashBoard* solo i dati con timestamp di arrivo maggiore (>) rispetto al timestamp di arrivo dell'ultimo dato disponibile nel *local storage* (fase di aggiornamento dei dati). 3) Ripetere il punto (2) per tutti i sensori presenti nel *local storage*. 4) Chiedere i dati dei sensori mancanti attraverso l'invio di una richiesta di dati con l'invio della lista dei sensori già noti. Le funzioni presenti nel modulo Dashboard verificano la lista dei sensori noti (che puo essere *null* nel caso in cui si abbiano i dati salvati in local storage) confrontandola con i sensori presenti nel database. 5) Se vi sono dei sensori nuovi vengono inviati i dati relativi ad un solo sensore, nella forma: {status: "DATA",sensor:id-key,data:[...]}. 5.1) Se un sensore presente nella lista dei dati noti nel *local storage* non è presente nei dati noti nel database il server risponde con una richiesta di eliminazione dei dati presenti nel *local storage* riferiti al sensore tramite la risposta {status: "DELETE",sensor:id-key}. 5.2) Il client elimina i dati del sensore specificato tramite *id-key*. 6) Il client riceve i dati, li memorizza nel *local storage* aggiungendoli alla lista o creando la lista se non è già presente. 7) Continuare a chiedere i dati dei sensori finché non si ha una risposta *null* da parte del server. 8) Quando il client riceve *null* significa sono stati inviati tutti i dati nuovi e sono stati aggiornati quelli precedentemente salvati nel *local storage*.
Errori	Possibili errori: 1) Mancata connessione al database. 2) Connessione protetta sul browser (non v'è possibilita di salvare i dati).
Error Menager	1) Delegare l'errore al warning manage specificando il codice di errore riscontrato.

Table 20: Requisito Funzionale F07: policy per il consenso all'invio dei dati.

Routing Consensus	**F07**

Continued on next page

Continued from previous page

Routing Consensus	**F07**
Descrizione	In questa funzione è implentata una policy (specifica) di sicurezza: verifica alcune regole per la corretta comunicazione dei dati e se soddisfatte da il consenso all'invio dei dati. Viene chiamata quando l'IOT-device fa richiesta di ricevere i controlli delle *Routine*. La funzione è binaria (SI/NO). E' stato realizzato un componente apposito per fare in modo che al sistema risultino trasparenti eventuali modifiche della policy.
Funzionalità	1) Verifica frequenza di richiesta dati. 2) Verifica del dispositivo (OS/IP) usato per la richiesta dei dati.
Errori	Possibili errori: 1) Mancata connessione al database. 2) Policy di verifica ha dato esito negativo.
Error Menager	1) Delegare l'errore al warning manage specificando il codice di errore riscontrato. 2) Non viene inviata nessuna risposta a fronte della richiesta.

Table 21: Requisito Funzionale F08: policy per il consenso all'invio dei dati.

Pipe Consensus	**F08**
Descrizione	In questa funzione è implentata una policy (specifica) di sicurezza: verifica alcune regole per la corretta comunicazione dei dati e se soddisfatte da il consenso all'invio dei dati. Viene chiamata quando l'IOT-device fa richiesta di ricevere i controlli delle *Fifo*. La funzione è binaria (SI/NO). E' stato realizzato un componente apposito per fare in modo che al sistema risultino trasparenti eventuali modifiche della policy.
Funzionalità	1) Verifica frequenza di richiesta dati. 2) Verifica del dispositivo (OS/IP) usato per la richiesta dei dati.
Errori	Possibili errori: 1) Mancata connessione al database. 2) Policy di verifica ha dato esito negativo.
Error Menager	1) Delegare l'errore al warning manage specificando il codice di errore riscontrato. 2) Non viene inviata nessuna risposta a fronte della richiesta.

Table 22: Requisito Funzionale F09: policy per il consenso all'invio dei dati.

Sensor Consensus	**F08**

Continued on next page

Continued from previous page

Sensor Consensus	**F08**
Descrizione	In questa funzione è implentata una policy (specifica) di sicurezza: verifica che alcune regole per la corretta comunicazione dei dati e se soddisfatte da il consenso all'invio dei dati. Viene chiamata quando l'IOT-device invia dati di sensori *sensori* precedentemente registrati nella piattaforma. La funzione è binaria (SI/NO). E' stato realizzato un componente apposito per fare in modo che al sistema risultino trasparenti eventuali modifiche della policy.
Funzionalità	1) Verifica frequenza di invio dati. 2) Verifica del dispositivo (OS/IP) usato per la richiesta dei dati. 3) Verifica la grandezza (Kbyte) dei dati inviati.
Errori	Possibili errori: 1) Mancata connessione al database. 2) Policy di verifica ha dato esito negativo.
Error Menager	1) Delegare l'errore al warning manage specificando il codice di errore riscontrato.

4.3 Business logic manager

Alla luce di una più approfondita analisi sui requisiti funzionali, per comprende meglio il meccanismo di gestione delle *Routine* e delle *Fifo* è stato realizzato il diagramma riportato in Figura 3 che mostra la logica di funzionamento della gestione dei dati. Successivamente saranno dettagliati i comportamenti delle singole funzioni.

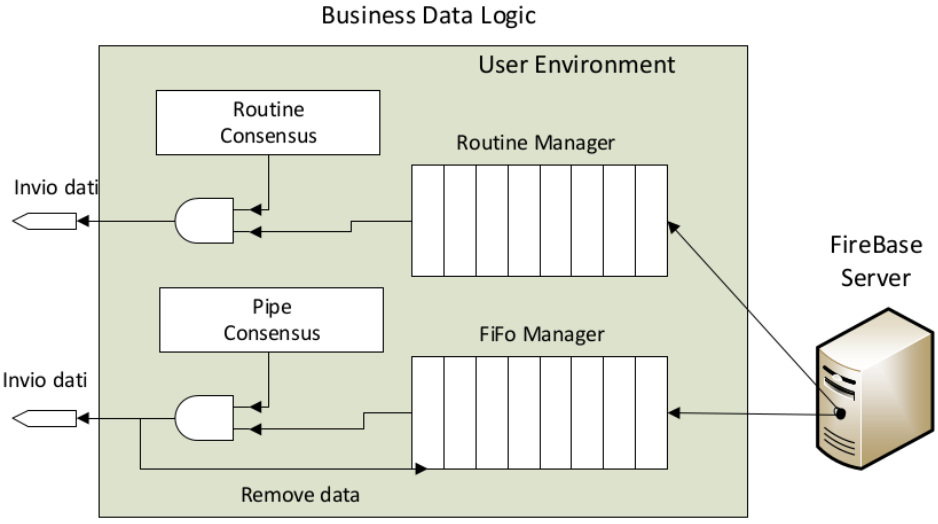


Figure 3: AdaprtiveHome: Data Business Logic.

Dal punto di vista logico, per ogni utente, possiamo modellare il comportamento della gestione di *Routine* e delle *Fifo* come la gestione di due strutture dati. In particolare il gestore delle Routine (Routine manager) quando viene fatta richiesta di invio delle routine di un utente, esegue una query sul database selezionando solo le istanze delle

routine per cui il timestamp di transizione sia scaduto (rispetto al momento della richiesta). Se vi sono *Routine* restituite dal database vengono aggiornati i valori associati alle *Routine* inviati all'IOT-device che ne ha fatto richiesta e fatto l'update delle tuple nel database. Per ridurre i tempi di latenza (e ridurre operazioni cpu consuming come la gestione del formato json) vengono inviate all'IOT-device solo le routine per cui vi sia stato un cambiamento di stato. Le routine sono persistenti e non vengono rimosse quando vengono lette. Per il trasferimento dei dai è necessario che ci sia il consenso alla transizione da parte della funzione: *Routine consensus*. Il gestore delle Fifo, lavora in modo diverso da quanto visto per le Routine, in particolare verifica che il valore di *freshness* associato ad un comando rapido non sia superiore ad una certa *threshold*. La struttura è simile ad una coda per cui i comandi rapidi sono ordinati per timestamp crescente. Nel momento in cui il valori di *freshness* > *threshold* la tupla viene rimossa e segnalato un warning (ovvero, non siamo stati in grado di eseguire il comando rapido entro un tempo definito). Viceversa, se c'è consenso all'invio da parte della funzione *Fifo consensus* il valore è inviato all'IOT-device in risposta ad una chiamata. Quando i dati relativi alle Pipe sono inviati i dati sono rimossi dal database.

4.4 Error Code

In questo paragrafo vengono riportati tutti i codici d'errore che possono verificarsi durante il funzionamento dell'applicazione. In tabella sono riportati gli errori nella forma: *ID*, *Name* e Descrizione.

Table 23: Error code.

ID	Descrizione	Gestione
ER0	Mancata connessione al database	Tramite delle strutture dati locali, gestite del *warning-mangarer*
ER1	Frequenza [richieste-{routine}/sec > thr]	warning-manger
ER2	Frequenza [richieste-{pipe}/sec > thr]	warning-manger
ER3	Timestamp [freshness > thr]	warning-manger
ER4	Connessione protetta del browser	warning-manger
ER5	Esito necativo policy sicurezza per la richiesta della routine	warning-manger
ER6	Esito necativo policy sicurezza per la richiesta della pipe	warning-manger
ER8	Valore di key-function è utilizzato più volte per definire diverse routine	warning-manger

Continued on next page

Continued from previous page

ID	Descrizione	Gestione
ER9	Valore di key-function è utilizzato più volte per definire diverse fifo	warning-manger
ER10	Valore di id-key sensore non è associato a nessun sensore	warning-manger
ER11	Formato dati non conforma allo standard	warning-manger
ER12	Dimensione documento troppo elevata	warning-manger

5 Modello a componenti dell'architettura

Alla luce di quanto visto nei capitoli precedenti vengono riportate in questo capitolo le "4" viste della *software architecture*. Queste saranno una guida di riferimento per la realizzazione software delle componenti. In particolare sono realizzate le viste di: *Activity Diagram*, *Component Diagram*, *Module diagrams* e *Deployment Diagram*.

5.1 Activity Diagram

In questi diagrammi sono descritti per via grafica i casi d'uso mostrando come avviene l'interazione tra l'utente a l'applicazione. Il processo descritto è stato ripreso da quando visto nella descrizione dei casi d'uso, con l'aggiunta di informazioni utili in fase di programmazione. Inoltre questa rappresentazione risulta utile per definire il grado di coesione tra le funzioni implementate nei diversi moduli, in particolare le funzioni che gestiscono il flusso di attività tra l'utente e l'applicazione sono state raggruppate nello stesso modulo (es: le funzioni per il logIn sono implementate nello stesso modulo). I diagrammi delle attività sono riportati in forma tabellare.

Table 24: Activity Diagram UC1: Registrazione dell'utente presso la piattaforma.

Name	Data Type

Continued on next page

Continued from previous page

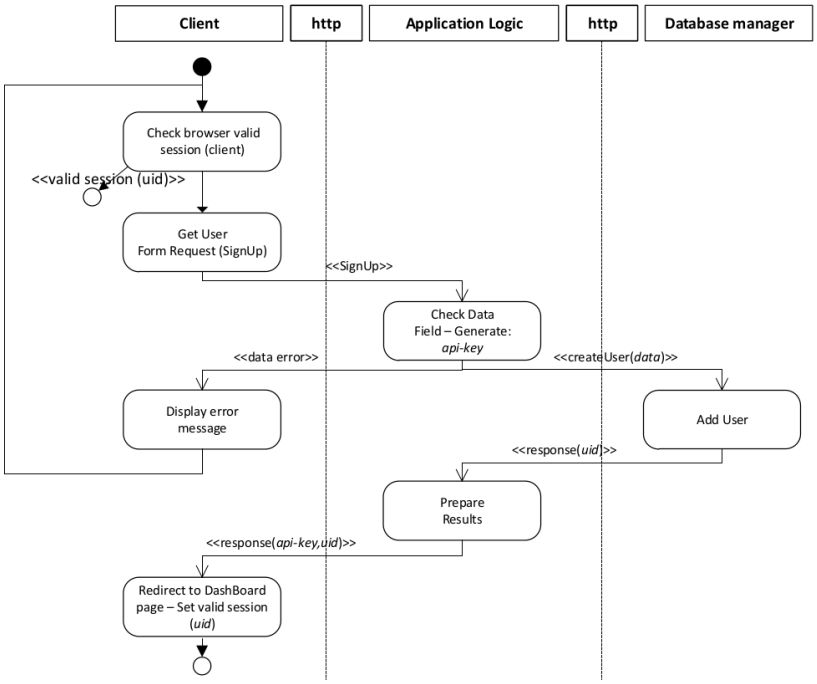
Name	Data Type
UC0	
SignUp	Registrazione dell'utente presso la piattaforma.

Table 25: Activity Diagram UC2: LogIn / LogOut dell'utente presso la piattaforma.

Name	Data Type

Continued on next page

Continued from previous page

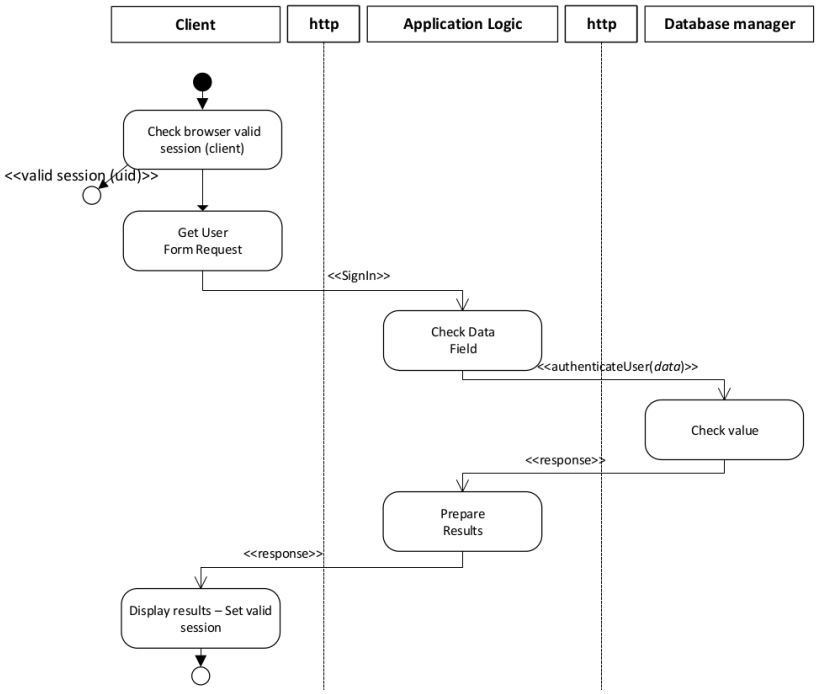
Name	Data Type
UC1	
SignIn	LogIn / LogOut dell'utente presso la piattaforma.

Table 26: Activity Diagram UC3: L'utente aggiunge un nuovo sensore.

Name	Data Type

Continued on next page

Continued from previous page

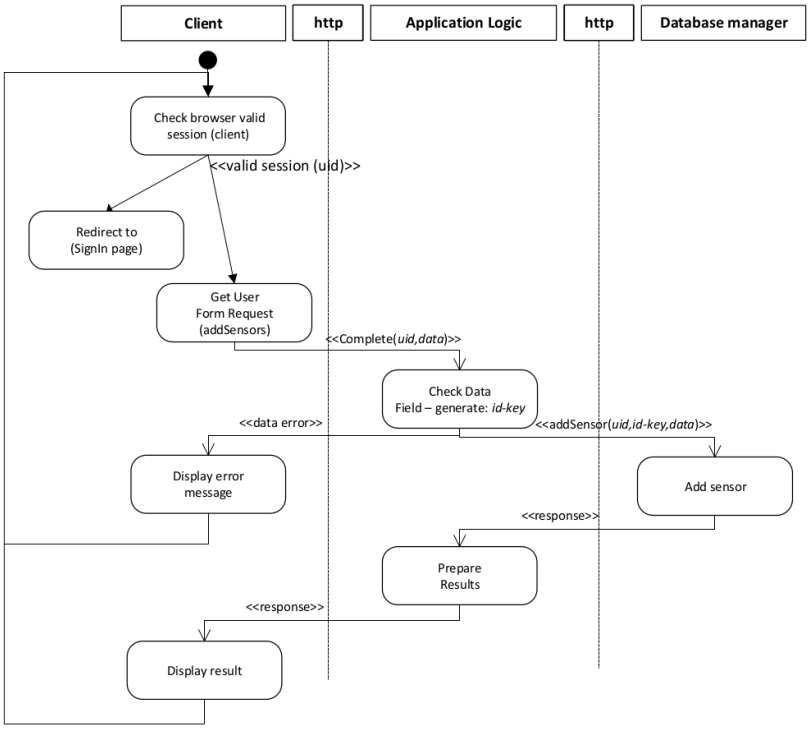
Name	Data Type
UC2	
AddSensor	L'utente aggiunge un nuovo sensore di cui vuole visualizzare i dati.

Table 27: Activity Diagram UC4: L'utente aggiunge un nuovo segnale di controllo.

Name	Data Type

Continued on next page

Continued from previous page

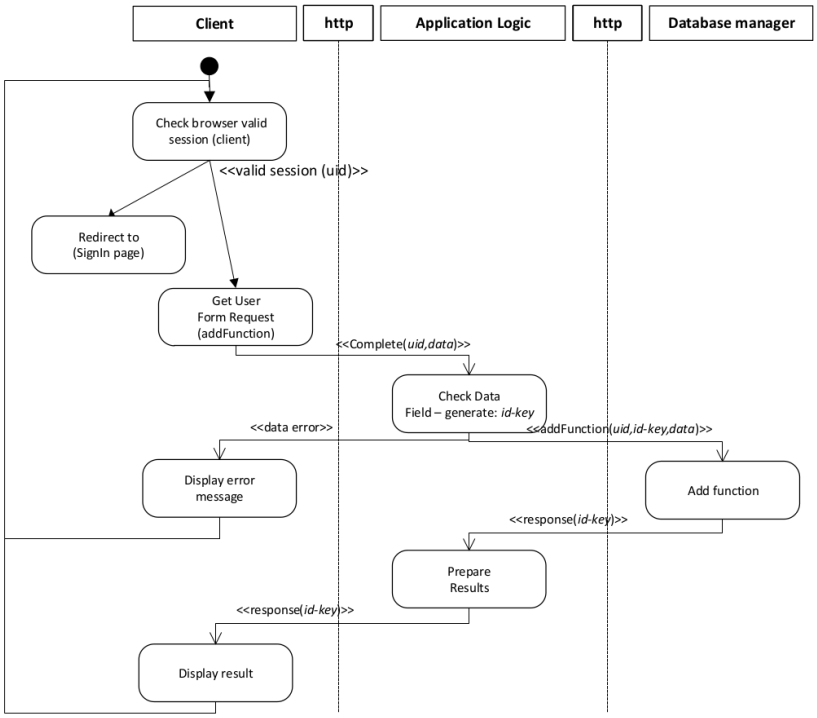
Name	Data Type
UC3	
AddFunction	L'utente aggiunge un nuovo segnale di controllo degli attuatori attraverso la dichiarazione di una nuova funzione.

Table 28: Activity Diagram UC4: L'utente aggiunge una nuova routine.

Code	Data Type

Continued on next page

Continued from previous page

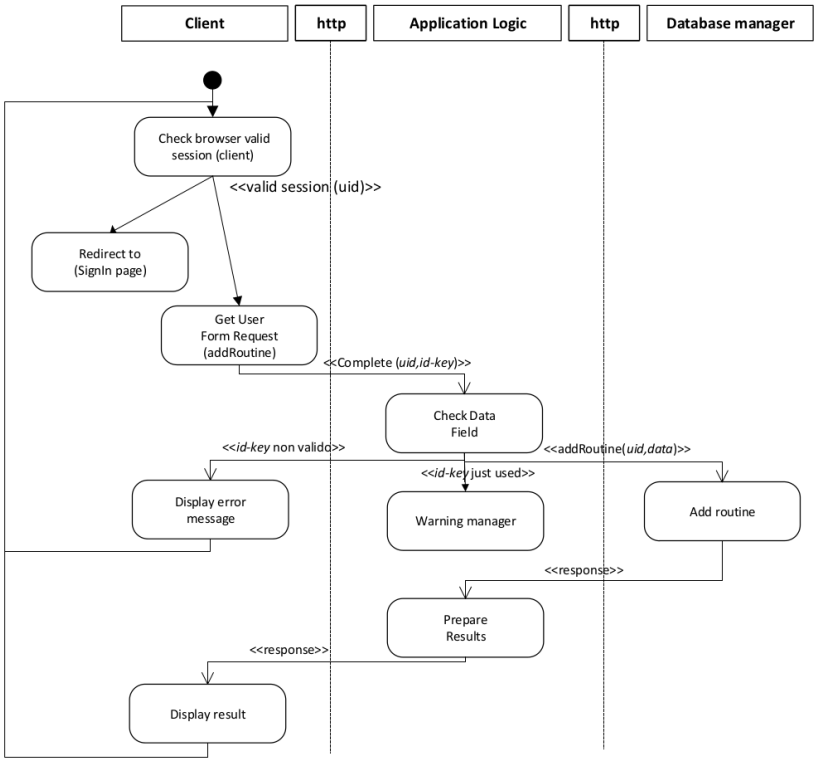
Code	Data Type
UC4	
AddRoutine	L'utente aggiunge una nuova routine su una deternimanta funzione.

Table 29: Activity Diagram UC5: L'utente (o Google-Assistant) vuole aggiungere un nuovo comando rapido.

Name	Data Type

Continued on next page

Continued from previous page

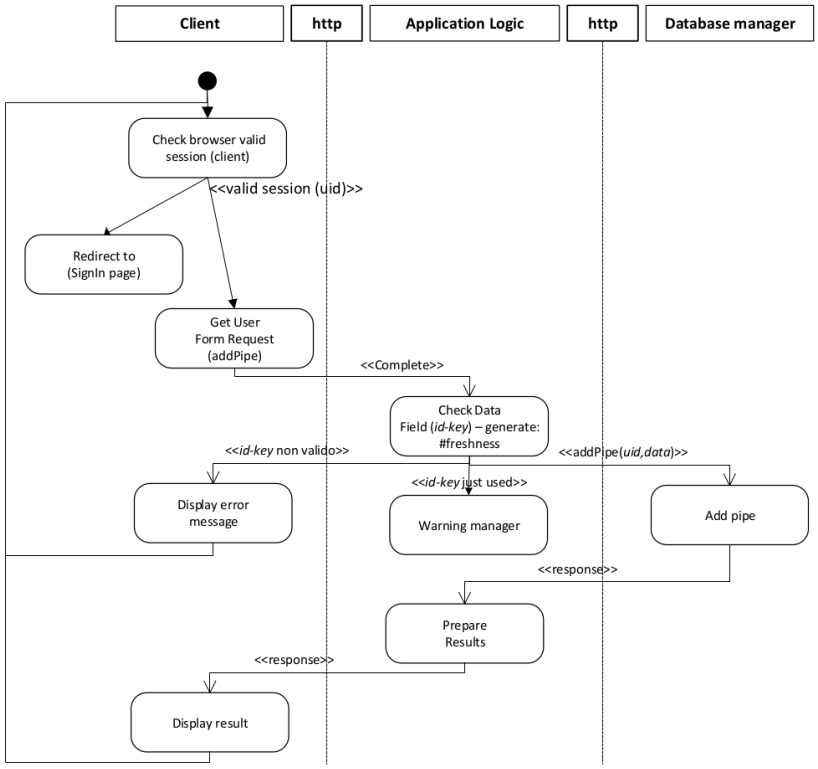
Name	Data Type
UC5	
AddFifo	L'utente o Google-Assitant aggiunge un nuovo controllo istantaneo.

Table 30: Activity Diagram UC6: L'User-Hardware richiede i controlli delle funzioni associate alle routine.

Name	Data Type

Continued on next page

Continued from previous page

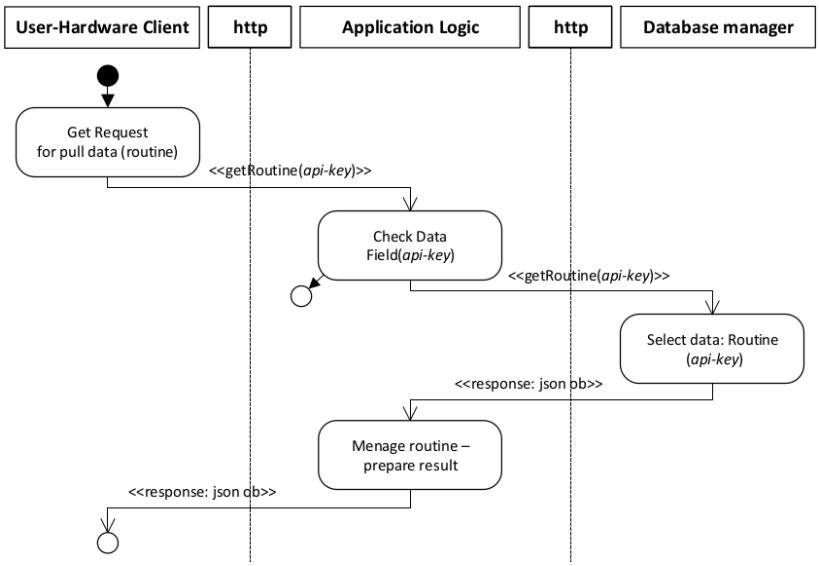
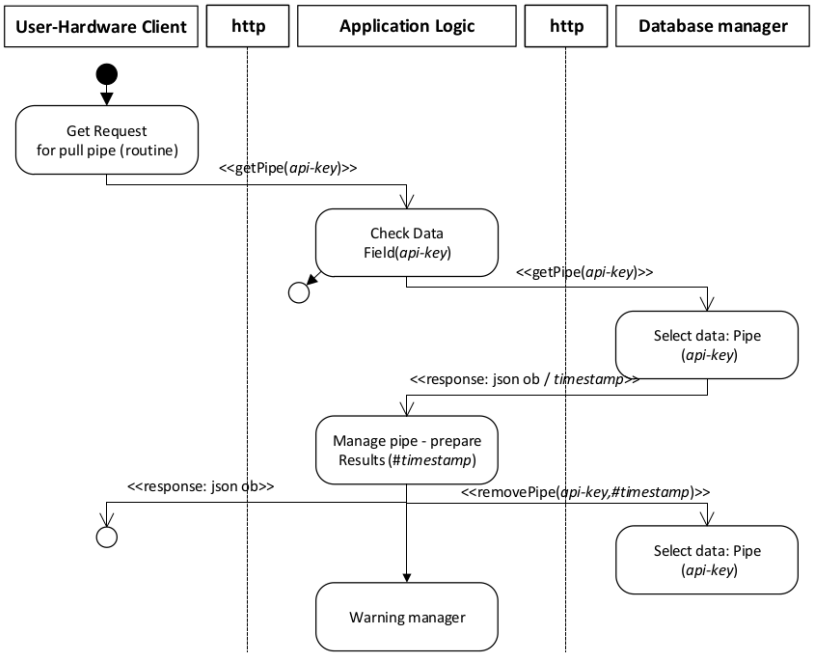
Name	Data Type
UC6	
getRoutine	L'User-Hardware richiede i controlli delle funzioni associate alle routine.

Table 31: Activity Diagram UC7: L'User-Hardware richiede i controlli delle funzioni associate ai comandi rapidi.

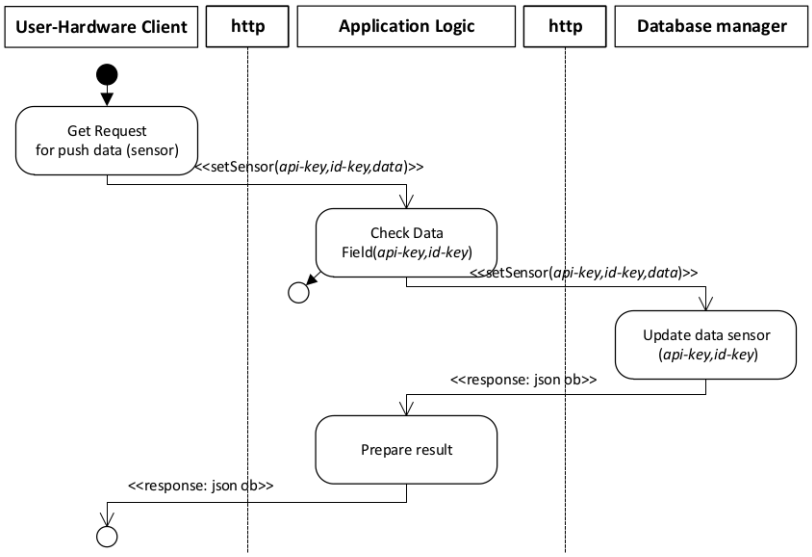
Name	Data Type
UC7	

Continued on next page

Continued from previous page

Name	Data Type
getFifo	L'User-Hardware richiede i controlli delle funzioni associate ai comandi rapidi.

Table 32: Activity Diagram UC7: User case UC8: L'User-Hardware richiede di inviare i dati relativi ad un sensore.

Name	Data Type
UC8	
setSensor	User case UC8: L'User-Hardware richiede di inviare i dati relativi ad un sensore.

5.2 Component Diagram

In Figura 4 è mostrato il diagramma delle componenti del sistema, in particolare sono riportati i sottosistemi: *Client*, *AdaptiveHome* (dove è implementata l'application logic dell'applicazione) e il sottosistema *DBMS Firebase*.

5.3 Module Diagram

In questo capitolo viene descritto il diagramma delle componenti dell'applicazione. Poichè è utilizzato il framework nodejs le componenti sono organizzate secondo moduli all'interno dei quali sono implementate le funzioni fornite da quel modulo. Nel diagramma delle componenti sono descritte oltre che le funzioni implementate all'interno dei moduli anche le inter-dipendenze con le altri componenti. Essendo una applicazione web-based sono state definite le interfaccie di comunicazione (API) tra la web-application e gli attori esterni ovvero il prototipo delle funzioni

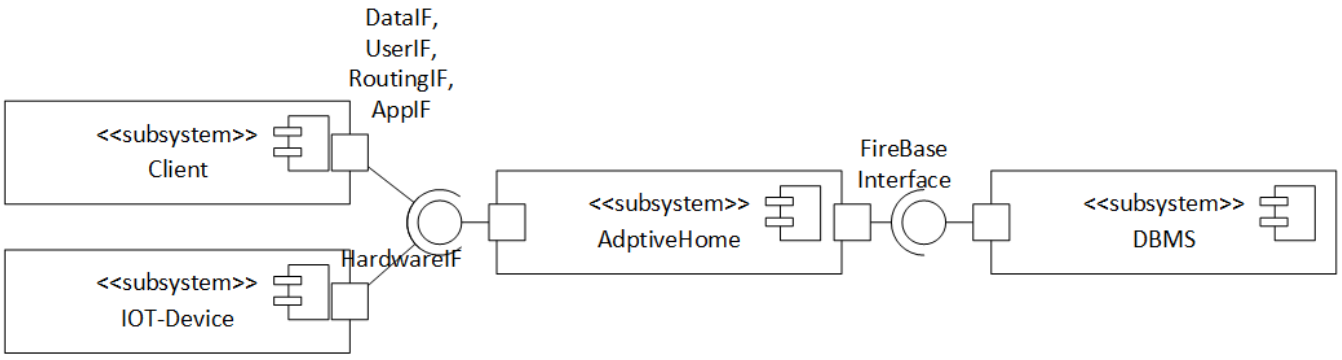


Figure 4: AdaptiveHome: Component Diagram.

visibili all'esterno del dominio applicativo. L'effettiva esecuzione è demandata a specifici componenti implementati nel server applicativo. Per mantenere un alto grado di manutenibilità e compatibilità con un approccio *server-less* a *microservizi* i componenti hanno una struttura semplice, ovvero un componente implementa un solo modulo.

Per una definizione completa dei moduli e degli oggetti utilizzati, in ordine vengono riportati: il *diagramma delle componenti* con le funzioni implementate all'interno del componente, i *tipi di dati* utilizzati (*Oggetti Javascript*) per scambiare informazioni/dati tra moduli e interfacce infine sono riportati i tipi di dato utilizzati nell'applicazione e nelle API (descritte dettagliatamente al Paragrafo 6.1). A seguito a queste considerazioni viene proposto in Figura 5 il diagramma delle componenti. Per garantire il massimo grado di manutenibilità e usabilità sono state create piccole interfacce (con poche funzioni) e le funzioni sono state raggruppate per coesione funzionale (come suggerito dall'*Activity Diagram*).

Alla luce delle considerazioni fatte e per una migliore gestione delle risorse (e quindi la possibilità di gestire più utenti) conviene delegare a funzioni di trigger sul database lo sviluppo delle funzioni *Routine Manager* e *Pipe Manager*, in questo modo quando il client esegue una pull request, i dati sono già pronti per essere inviati perché non necessitano di nessuna altra valutazione da parte del server applicativo (il server applicativo aggiunge comunque dei dati di controllo nella risposta e verifica il consenso alla transizione). Per cui risulta conveniente istallare le componenti di gestione del database (specificate nei requisiti funzionali) direttamente nella componente FireBase - Cloud Functions, ma essendo un prodotto "pay to go" per una prima fase di prototipazione è stato scelto di implementare le funzioni all'interno dell'applicazione. L'obiettivo nelle versioni successive (in base al carico di utenti) è quello di migrare le funzioni implementate verso un approccio *server-less* e delegare sempre più funzioni verso i server proprietari Google Cloud Function per garantire una migliore esperienza d'uso. Per cui le componenti sono state sviluppate cercando di dividere le funzioni che si occupano dei dati (*database manager*) da quelle che si occupano dell'application logic e di avere il minor grado di accoppiamento tra le componenti.

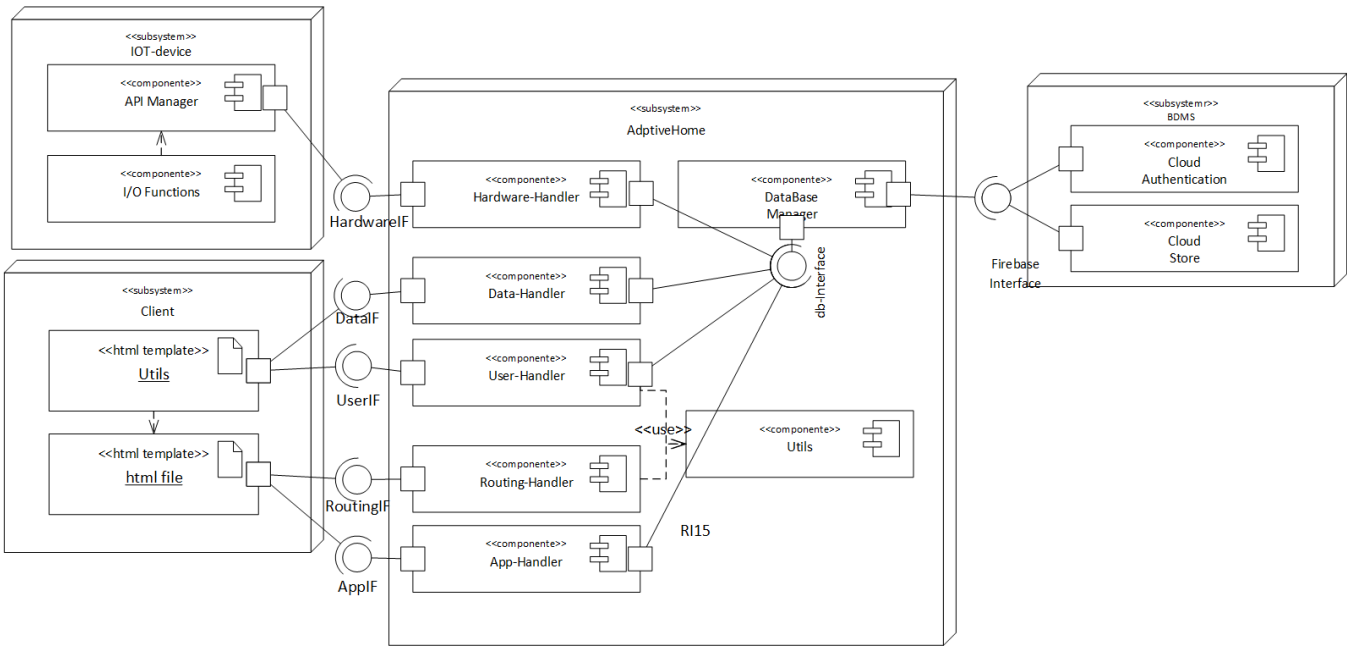


Figure 5: AdaptiveHome: Component Diagram (top level)

Table 33: Descrizione componenti.

Componente	Descrizione
Hardware-handler	Modulo creato per rispondere alla richieste dell'iot-device. In particolare gestisce i dati delle routine/fifo da inviare all'iot-device e pilotare di conseguenza i dispositivi fisici. Tutte le richieste che vengono gestite da questo componente hanno l'url: <url:http://domain_app/hardware/:APIKEY/>.

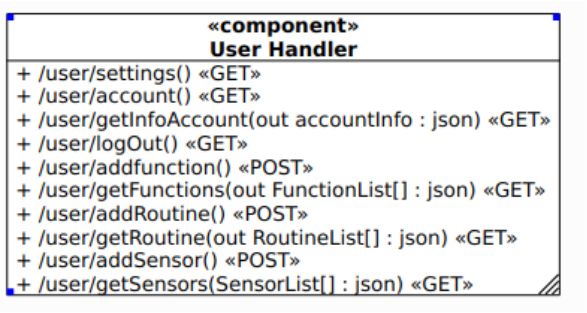
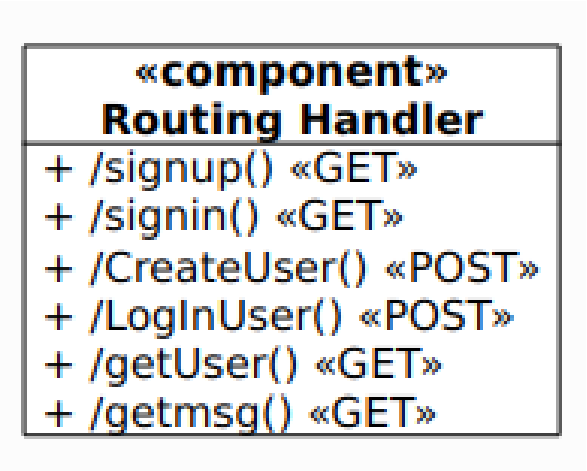
Continued on next page

Continued from previous page

Componente	Descrizione
User-handler	Modulo che gestisce le richieste dell'utente quanto usa la piattaforma web (as esempio per verificare i dati di autenticazione). Tutte le richieste che vengono gestite da questo componente hanno l'url: <url:http://domain_app/user/..>. Perchè una richiesta per questo modulo sia valida è necessario che l'utente abbia una sessione attiva valida. Per cui è necessario che faccia il LogIn prima di effettuare queste chiamate. Per cui il modulo *User-Handler* è visibile solo agli utenti che sono registrati (in questo modo è possibile proteggere anche pagine che sono visibili sono se tenti della piattaforma, come per esempio la dashboard). Per fare questo sono stati implementati dei middleware con lo scopo di verificare le sessioni attive, se non c'è una sessione attiva si viene reindirizzati alla pagina *home.html*. L'utilizzo di funzioni di middleware semplificano la gestione della richiesta e assicurano che tutte le richieste gestite da quel modulo soddisfino certi requisiti.
Data-handler*	Modulo usato dall'applicazione per le elaborazione dei dati provenienti dai sensori. In particolare vengono messe a disposizione tutte le funzioni per il corretto funzionamento della DashBoard. Tutte le richieste che vengono gestite da questo componente hanno l'url: <url:http://domain_app/data/..>. *Al momento questa componente non è implementata.*
App-handler*	Modulo per la gestione di informazioni generali dell'applicazione, come ad esempio il numero totale di utenti, o il numero totale di richieste erogate. *Al momento questa componente non è implementata.*
Database-manager	Modulo che gestisce tutte le richieste verso il databases, in particolare mette a disposizione funzioni specifiche per estrarre dati dal database. Utilizza chiamate asincrone per ridurre i tempi di latenza.
Routing-handler	Modulo che si occupa del routing dei template html dei domini per cui non è richiesta nessuna sessione attiva (Ad esempio la pagine ./home.html).

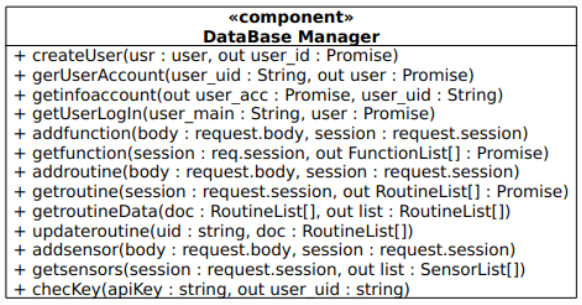
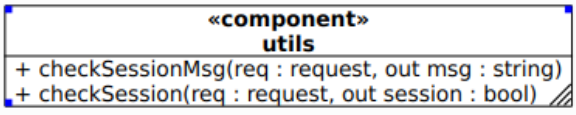
Vengono ora riportate in forma tabellare le funzioni implementate all'interno dei singoli componenti

Table 34: Descrizione funzioni implementate all'interno dei componenti.

Component	Struttura
HardWare handler	
User handler	
Routing handler	

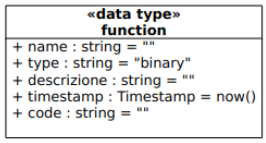
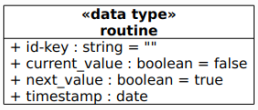
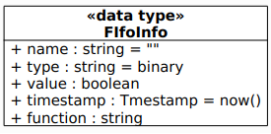
Continued on next page

Continued from previous page

Component	Struttura
Database manager	
Utils	

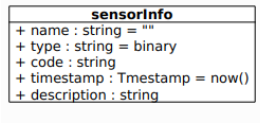
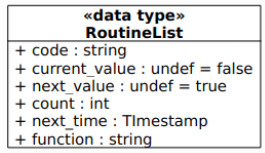
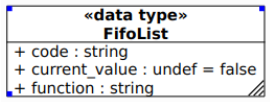
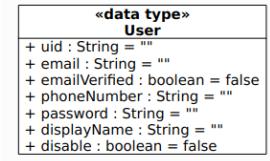
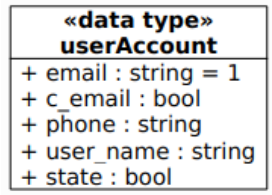
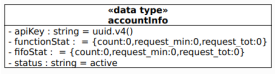
Tipi di dato utilizzati nell'applicazione

Table 35: Tipi di dato utilizzati nei componenti.

Data Type	Descrizione	Struttura
function	Oggetto che descrive le caratteristche di una funzione.	
routineInfo	Oggetto che descrive i valori associati ad una singola routine.	
fifoInfo	Oggetto che descrive le informazioni associate ad una singola fifo.	

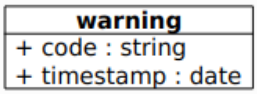
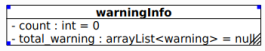
Continued on next page

Continued from previous page

Data Type	Descrizione	Struttura
sensorInfo	Oggetto che descrive le informazioni associate ad un singolo sensore.	
RoutineList	Oggetto che descrive una routine quando è passata all'iot-device.	
FifoList	Oggetto che descrive una fifo quando è passata all'iot-device.	
User	Oggetto che descrive i dati dell'applicazione associati all'utente.	
userAccount	Oggetto che descrive i dati di account associati all'utente.	
accountInfo	Oggetto che descrive le statistiche dell'applicazione associate all'utente.	

Continued on next page

Continued from previous page

Data Type	Descrizione	Struttura
warning	Oggetto che descrive i Warning.	
waringInfo	Oggetto che descrive una collezione di warning per un utente.	

5.4 Deployment Diagram

A seguito delle analisi condotte nei capitoli precedenti, viene riportato in Figura 6 il *deployment diagram*. L'unico componente che risulta interessante è il Server Applicativo che dovrà essere raggiungibile dall'esterno tramite un dominio pubblico e in funzione del numero di utenti poter scalare su più calcolatori l'applicazione per garantire delle prestazioni ottimali. E' stato scelto di ospitare il server web all'interno di un container cloud fornito dalla piattaforma *heroku*. I restanti componenti non risultano interessanti in quanto l'IOT-device può essere installato in un qualsiasi dispositivo che abbia la connessione ad internet (non è necessario che l'indirizzo ip sia pubblico, ne statico e può essere installato in reti private), mentre i servizi offerti dalla piattaforma Firebase sono già ospitati all'interno dell'infrastruttura proprietaria del provider Google.

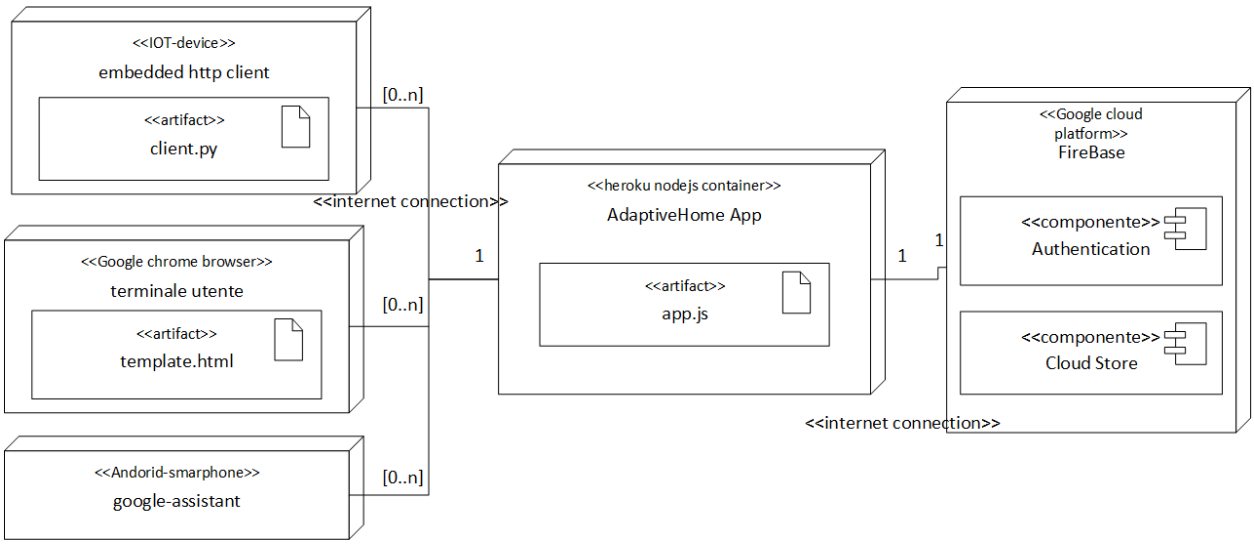


Figure 6: AdaptiveHome: Deployment Diagram.

6 Implementazione

6.1 API

Al momento non sono implementate le API relative alla gestione delle Fifo

6.1.1 GET: /getInfoAccount (User Handler)

La richiesta per richiedere tutte le informazioni sull'account dell'utente deve essere fatta al seguente url: <url: https://adeptivehome.domain.com/user/settings>. Per chiamare questa API è necessario avere una sessione attiva valida.

Formattazione richiesta:

```
HEADERS:
    Content Type : application/x-www-form-urlencoded
SESSION:
    uid: {{UID}}
```

La risposta è in formato json:

```
[
.
.
{
RoutingStat:
    {count:{{number}},request_min:{{number}},request_tot:{{number}}},
FifoStat:
    {count:{{number}},request_min:{{number}},request_tot:{{number}}},
apiKey: {{APIKEY}},
status: {{Active/Idle}},
},
.
.
]
```

è possibile testare il corretto funzionamento dell'API tramite il seguente processo:

```
1: open browser
2: SignIn: https://dry-island-85561.herokuapp.com/
3: open new windows and open:
   https://dry-island-85561.herokuapp.com/user/getInfoAccount

{"RoutingStat":{"count":0,"request_min":0,"request_tot":0},
```

```
 "apiKey":"13cb7435-d956-451c-834e-b2c0afdef600","status":
 "Active","FifoStrat":{"count":0,"request_tot":0,"request_min":0
}}
```

6.1.2 POST: /addFunction (User Handler)

La richiesta per aggiungere una nuova funzione deve essere fatta al seguente url: <url: https://adeptivehome.domain.com/user/addFunction>. Per chiamare questa API è necessario avere una sessione attiva valida.

Formattazione richiesta:

```
HEADERS:
    Content Type : application/json
SESSION:
    uid: {{UID}}
BODY:
    name:{{String}}
    code:{{String}}
    type:{{String}}
    description:{{String}}
```

Se l'aggiunta di una nuova funzione va a buon fine si viene reindirizzati alla pagina *settings* e compare la scritta *Correct update databases*.

è possibile testare il corretto funzionamento dell'API tramite il seguente processo:

```
1: open browser
2: SignIn: https://dry-island-85561.herokuapp.com/
3: type: https://dry-island-85561.herokuapp.com/user/settings
4: press button >> +Functions <<
```

6.1.3 GET: /getFunctions (User Handler)

La richiesta per richiedere tutte le funzioni definite dell'utente deve essere fatta al seguente url: <url: https://adeptivehome.domain.com/user/getFunctions>. Per chiamare questa API è necessario avere una sessione attiva valida.

Formattazione richiesta:

```
HEADERS:
    Content Type : application/x-www-form-urlencoded
```

```
SESSION:
    uid: {{UID}}
```

La risposta è in formato json:

```
[
.
.
{
timestamp:{{timestamp}},
description: {{String}},
name: {{String}},
type: {{enum(Bynary,Discrete,Analog)}},
code: {{String}}
},
.
.
]
```

è possibile testare il corretto funzionamento dell'API tramite il seguente processo:

```
  1: open browser
  2: SignIn: https://dry-island-85561.herokuapp.com/
  3: open new windows and type:
  https://dry-island-85561.herokuapp.com/user/getFunctions

[{"timestamp":"2021-06-19T14:38:16.952Z","description":"",
  "name":"Prova","type":"Binary","code":"FN01"},
 {"code":"FN02","type":"Binary","name":"Prova","description":"",
  "timestamp":"2021-06-19T21:03:10.688Z"},{"type":"Binary",
 "description":"","name":"Prova","timestamp":"2021-06-19T21:02:46.832Z",
 "code":"FN03"},
 {"timestamp":"2021-06-26T12:31:37.461Z","code":"FN04","type":"Binary",
  "description":"","name":"nEW"}]
```

6.1.4 POST: /addRoutine (User Handler)

La richiesta per aggiungere una nuova routine deve essere fatta al seguente url: <url: https://adeptivehome.domain.com/user/addRoutine>. Per chiamare questa API è necessario avere una sessione attiva valida.

Formattazione richiesta:

```
HEADERS:
    Content Type : application/json
SESSION:
    uid: {{UID}}
BODY:
    name:{{String}}
    function_code:{{String}}
    code:{{String}}
    days:{{Number}}
    hours: {{Number}}
    type:{{String}}
    current_value:{{True/false}}
    next_value:{{True/false}}
    description:{{String}}
```

Se l'aggiunta di una nuova routine va a buon fine si viene reindirizzati alla pagina *settings* e compare la scritta *Correct update databases*.

è possibile testare il corretto funzionamento dell'API tramite il seguente processo:

```
1: open browser
2: SignIn: https://dry-island-85561.herokuapp.com/
3: type: https://dry-island-85561.herokuapp.com/user/settings
4: press button >> +Routine <<
```

6.1.5 GET: /getRoutine (User Handler)

La richiesta per richiedere tutti i controlli riferiti alle routine definite dell'utente deve essere fatta al seguente url: <url: https://adeptivehome.domain.com/user/getRoutine>. Per chiamare questa API è necessario avere una sessione attiva valida.

Formattazione richiesta:

```
HEADERS:
    Content Type : application/x-www-form-urlencoded
SESSION:
    uid: {{UID}}
```

La risposta è in formato json:

```
[
.
```

```
.
{
name:{{String}},
day:{{Number}},
hours:{{Number}},
timestamp:{{timestamp}},
count: {{Number}},
next_time: {{String}},
start:{{timestamp}}
type: {{enum(Bynary,Discrete,Analog)}},
current_value:{{True/False}}
next_value:{{True/False}}
code: {{String}}
},
.
.
]
```

è possibile testare il corretto funzionamento dell'API tramite il seguente processo:

```
    1: open browser
    2: SignIn: https://dry-island-85561.herokuapp.com/
    3: open new windows and type:
    https://dry-island-85561.herokuapp.com/user/getRoutine

[{"name":"Prova","days":"1","timestamp":{"_seconds":1624040586,
"_nanoseconds":809000000},"function":"F01","hours":"2","count":18,
"code":"FN01","next_time":"27/Jun 08:00","description":"","start":
{"_seconds":1623997380,"_nanoseconds":0},"current_value":false,
"next_value":true,"type":"Binary"},{"count":5,"function":"F01",
"days":"1","next_value":true,"type":"Binary","current_value":false,
"description":"","code":"FN02","timestamp":{"_seconds":1624115917,
"_nanoseconds":393000000},"next_time":"27/Jun 08:10","hours":"4",
"start":{"_seconds":1623658200,"_nanoseconds":0},"name":"Prova"}]
```

6.1.6 POST: /addSensor (User Handler)

La richiesta per aggiungere un nuovo sensore deve essere fatta al seguente url: <url: https://adeptivehome.domain.com/user/addSensor>. Per chiamare questa API è necessario avere una sessione attiva valida.

Formattazione richiesta:

```
HEADERS:
    Content Type : application/json
```

```
SESSION:
    uid: {{UID}}
BODY:
    name:{{String}}
    code:{{String}}
    description:{{String}}
```

Se l'aggiunta di una nuovo sensore va a buon fine si viene reindirizzati alla pagina *settings* e compare la scritta *Correct update databases*.

è possibile testare il corretto funzionamento dell'API tramite il seguente processo:

```
1: open browser
2: SignIn: https://dry-island-85561.herokuapp.com/
3: type: https://dry-island-85561.herokuapp.com/user/settings
4: press button >> +Sensor <<
```

6.1.7 GET: /getSensors (User Handler)

La richiesta per richiedere tutti i sensori definiti dall'utente, deve essere fatta al seguente url: <url: https://adeptivehome.domain.com/user/getSensors>. Per chiamare questa API è necessario avere una sessione attiva valida.

Formattazione richiesta:

```
HEADERS:
    Content Type : application/x-www-form-urlencoded
SESSION:
    uid: {{UID}}
```

La risposta è in formato json:

```
[
.
.
{
name:{{String}},
descriptions: {{String}}
timestamp:{{timestamp}},
type: {{enum(Bynary,Discrete,Analog)}},
code: {{String}}
},
.
```

```
.
]
```

è possibile testare il corretto funzionamento dell'API tramite il seguente processo:

```
    1: open browser
    2: SignIn: https://dry-island-85561.herokuapp.com/
    3: open new windows and type:
    https://dry-island-85561.herokuapp.com/user/getSensors

[{"type":"Analog","description":"Sensore di temperatura camera",
  "name":"Temperature","timestamp":{"_seconds":1625219552,
  "_nanoseconds":417000000},"code":"T01"}]
```

6.1.8 (IOT-device): /getRoutine

La richiesta per richiedere le routine su cui v'è stato un cambio di stato deve essere fatta al seguente url: <url: https://adeptivehome.domain.com/haerware/getRoutines/:APIKEY>. E necessario avere l'APIKEY generata in fase di registrazione alla piattaforma. Nel caso è possibile recuperarlo nella pagine *account.html* una volta che si è registrati.

Formattazione richiesta:

```
HEADERS:
    Content Type : application/x-www-form-urlencoded
PARAMS:
    apiKey: {{APIKEY}}
```

La risposta è in formato json:

```
[
.
.
{
current_value: (true/false),
next_value: (true/false),
next_time: timestamp,
count: int,
function: string,
code:string
},
.
```

```
.
]
```

è possibile testare il corretto funzionamento dell'API tramite il seguente comando da cli:

```
$ curl -i https://dry-island-85561.herokuapp.com
/hardware/getRoutines/13cb7435-d956-451c-834e-b2c0afdef600

[{"current_value":false,"next_value":true,"next_time":
"2021-06-27T08:00:00.000Z","count":18,"function":"F01",
"code":"FN01"},{"current_value":false,"next_value":true,
"next_time":"2021-06-27T08:10:00.000Z","count":5,
"function":"F01","code":"FN02"}]
```

Una volta che i dati sono letti, se eseguimo di nuovo il comando, non verrranno più restituiti. Questo perchè il valore di next-time è stato aggiornato e non verranno più verificati degli update su quella tupla.

```
$ curl -i https://dry-island-85561.herokuapp.com/
   hardware/getRoutines/ 13cb7435-d956-451c-834e-b2c0afdef600

[]; // lisa vuota
```

6.1.9 (IOT-device): /getFifo

Al momento non risulta implementata

6.2 Organizzazione del codice

In questa rappresentazione viene mostrata la struttura delle cartelle dove è inserito il codice.

```
src/
 app.js
 bin
    www.js
 checkConfig.json
 IOT-Client
    IOT-Client.iml
    main.py
 lib
    database_manager
```

```
        adaptivehome-firebase.json
        database_manager.js
package.json
package-lock.json
public
   Image
      home_1.jpg
      home.jpeg
      home_wets.png
      house-house.png
      SignUp.webp
   vendors
       chart.js
       css
       Date-Picker
       font-awesome
       iconfonts
       js
       mdi
          README
routes
   app_interface.js
   data_interface.js
   hardware_interface.js
   routing.js
   user_interface.js
   utils.js
views
    account.ejs
    error.ejs
    home.ejs
    index.ejs
    javascripts
       chart.js
       dashboard.js
       misc.js
       off-canvas.js
    pricing.ejs
    Settings
       addfunctions.ejs
    settings.ejs
    SignIn.ejs
    SignUp.ejs
    stylesheets
       demo_1
          style.css
```

```
          style.css.map
        fonts
          Roboto
        home.css
        shared
            style.css
            style.css.map
    Template
 footer.ejs
 head.ejs
 header.ejs

93 directories, 3455 files
```

6.3 Dipendenze dei moduli

In questo capitolo vengono riportate le dipendenze tra i moduli creati. Per la creazione di questi diagrammi sono stati utilizzati appositi tool di analisi del codice. Per avere una rappresentazione più compatta è stata inserito il vincolo di: *–max-depth 3* ovvero è stata fatta la ricerca solo fino al terzo livello di profondità partendo dalla radice. Per cui sono stare necessarie diverse viste. Nelle tabelle sottostanti sono riportate in ordine di dettaglio.

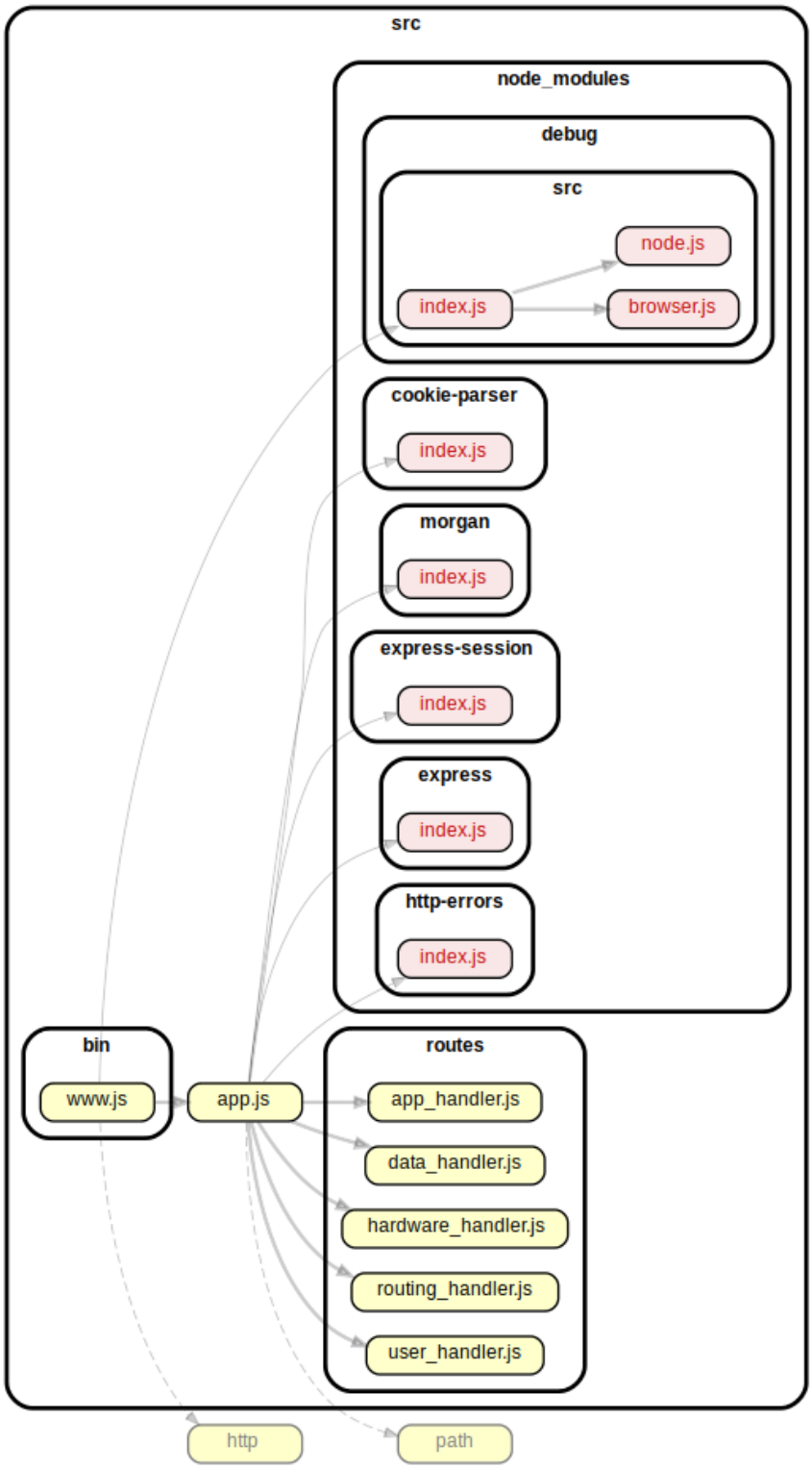
Table 36: Interdipendenze tra i moduli dell'applicazione (top-level).

Component diagram

Continued on next page

Continued from previous page

Component diagram

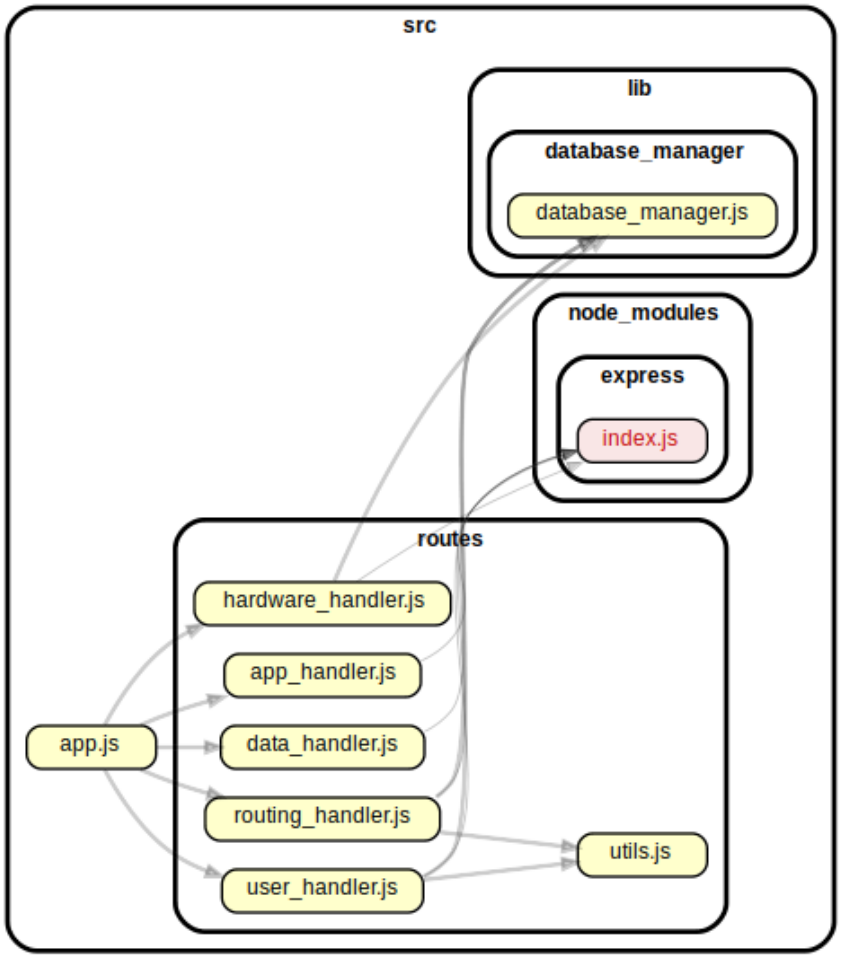


Continued on next page

Continued from previous page

Component diagram
In questo diagramma sono riportate le interdipendenze tra i moduli dell'applicazione. Inoltre sono riportate anche le dipendenze dalle librerie. Il componente *www.js* si occupa di esportare il componente contenente l'applicazione (app.js) e di creare il server mettendo l'app in ascolto su una porta http. Il componente *app.js* è decisamente più laborioso, in quanto carica tutti i moduli, definisce gli *url* delle API e configura l'applicazione.

Table 37: Interdipendenze tra i moduli dell'applicazione.

Component Diagram

In questo diagramma vengono dettagliate le dipendenze tra le componenti all'interno del package *routes*. In particolare lo schema segue quanto visto per il diagramma della componenti riportato nei precedente capitoli.

7 Deploy

7.1 Installazione del software

L'installazione è suddivisa in due fasi, la prima per installare il server applicativo mentre la seconda per installare l'applicazione all'interno dell'IOT-device.

7.2 Server

In questo paragrafo viene descritto il processo di deploy del server

7.2.1 Prerequisiti

- git
- nodejs
- npm
- heroku cli
- heroku account

7.2.2 Download source

clone repository

```
$ git clone https://github.com/jacopoRodeschini/AdaptiveHome.git
$ cd AdaptiveHome/
```

installare le dipendenze (locale)

```
$ npm install --save
```

7.2.3 Host on Heroku

Loggarsi alla piattofaroma heroku per l'esecuzione di heroku cli

```
$ heroku login
$ ...
$ ...
```

Installare il progetto in un container cloud (heroku provider)

```
$ git push heroku main
$ ...
$ ...
```

Istanziare le variabili di ambiente (environment)

```
$ heroku config:set NODE_ENV='production'
$ heroku config:set PRIVETE_KEY='QE34&&7/1234?'
$ heroku config:set NPM_CONFIG_PRODUCTION=false
$ heroku config:set NODE_PORT=3000
```

assegnare le risorse al server e verificarne l'allocazione

```
$ heroku ps:scale web=<<number_worker>>
$ heroku logs --tail
$ heroku ps
```

accedere al browser e verificare il funzionamento

```
$ ping <<AdaptiveHome.domain.com>>
$ heroku open // <<repository_heroku.git>>

browser goto -> http://<<repository_heroku.git>>
```

7.3 IOT-device

In questo viene riportata la guida di installazione dell'applicazione all'interno del device IOT che deve essere in grado di poter comunicare con il server applicativo ed interagire con le API esposte dell'applicazione. E' lasciata piene libertà all'utente di sviluppare ed installare il client come meglio crede per risolvere esigenze particolari (come per esempio avere diversi IOT-device configurabili). La piattaforma mette a disposizione diversi software "pronti all'uso", in particolare viene fornito un programma sviluppato *uPython* per essere compatibile con molti device economici (come la famiglia ESP) poiché è una versione minimalista di python, ottimizzata per piattaforme embedded. In questa fase viene mostrato il processo di installazione dell'applicativo sul dispositivo ESP8266.

7.3.1 Prerequisiti

- Device: ESP8266
- Update del firmaware (uPython) Istruzioni: Rodeschini
- esptool, ampy framework Istruzioni: Rodeschini

7.3.2 Test API

```
$ curl -i http://<heroku_url>/hardware/getRoutines/
```

7.3.3 Download client

```
$ git clone https://github.com/jacopoRodeschini/SoftwareLab.git
```

Connettere il dispositivo target (ESP8266) alla porta USB e verificare che sia correttamente riconosciuto

```
$ ls /dev/tty*
```

7.3.4 Upload firmware

```
$ cd SoftwareLab/stc/IOT-Client/
$ ampy --port /dev/PORT run main.py
```