UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica applicata

Corso di Laurea in Ingegneria Informatica

ELABORATO DI TESI

SVILUPPO DI UNA WEB APPLICATION PER IL MONITORAGGIO INTELLIGENTE DEL CONSUMO ENERGETICO CON CASI D'USO IOT

Docente Relatore
Prof.ssa Lidia Fotia

Candidato
Francesca Melloni

Azienda Ospitante
Ericsson Telecomunicazioni S.p.A.

Matricola
0612705287

Anno Accademico 2023/2024

dedica

Sommario

Introduzione

Nella realtà in cui ci troviamo è sempre più di interesse l'immediata ottimizzazione delle risorse energetiche.

Nonostante nella nostra società stia crescendo costantemente la consapevolezza su questo argomento e le energie rinnovabili vengano incentivate finanziariamente, siamo ancora lontani dal vivere in comunità che siano alimentate solo da fonti di energia pulita.

Per ridurre l'impatto ambientale e i costi a esso legato si può pensare ad un'opzione che richiede la condivisione delle risorse energetiche rinnovabili e la giusta ripartizione del loro prezzo. Tale opzione permetterebbe di far nascere delle reti comunitarie che siano più resilienti e orientate verso un approccio consapevole dell'energia con l'obiettivo di innescare una maggiore consapevolezza e un impegno proattivo per un futuro più sostenibile dal punto di vista ambientale.

Il presente elaborato rappresenta la continuazione di un progetto già avviato, il quale si proponeva il medesimo obiettivo. Sono state implementate e perfezionate diverse funzionalità, introducendo inoltre casi d'uso reali in grado di simulare una piccola parte della comunità oggetto del nostro studio. L'intento è stato quindi quello di creare scenari concreti dai quali estrapolare suggerimenti intelligenti per promuovere un utilizzo consapevole dell'energia, come precedentemente discusso.

Nel capitolo 1 vengono presentate le tecnologie utilizzate per la progettazione e lo sviluppo della soluzione proposta. Ognuna di queste viene descritta in termini di caratteristiche, di vantaggi e di svantaggi, concentrandosi in particolar modo sul motivo per cui sono state scelte.

Nel capitolo 2 viene spiegato quale è stato l'approccio alla soluzione in tutti gli aspetti. Lo sviluppo del progetto è avvenuto su 3 fronti: casi d'uso, front-end e back-end. Ognuno di questi si è basato su uno studio dei requisiti svolto in una fase iniziale che ha fornito le basi per una progettazione sempre focalizzata sulle esigenze dell'utente.

Nel capitolo 3 sono state descritte nel dettaglio le implementazioni dei tre aspetti portanti del progetto. Per i casi d'uso è stata spiegata la logica di implementazione della comunicazione con il server oltre che alla funzionalità del singolo caso. Nel caso del back-end e del front-end sono state inserite diverse figure che mostrano il codice di implementazione e le informazioni annesse, in modo che fosse chiaro il procedimento utilizzato. Infine, viene mostrato un esempio del funzionamento dell'intera applicazione svolto durante la fase di testing.

Nel capitolo 4 vengono fatte una serie di considerazioni sul problema affrontato e sulla soluzione adottata. Vengono tratte le conclusioni del lavoro svolto e vengono discusse le possibili applicazioni future.

Capitolo 1 - Panoramica delle tecnologie utilizzate

1.1 Tecnologie di backend

Le tecnologie di backend utilizzate nel contesto del mio progetto hanno avuto un ruolo importante per la gestione della logica di business, l'elaborazione delle richieste dei client e la gestione dei dati di un'applicazione web. Di seguito sono descritte le principali tecnologie impiegate illustrandone le caratteristiche e i vantaggi principali.

1.1.1 Flask

Flask [1] è un micro-framework per lo sviluppo web in Python , progettato per essere semplice e flessibile. I suoi maggiori vantaggi riguardano la leggerezza, la modularità e la facilità con cui permette di avviare rapidamente progetti di sviluppo web. Essendo un framework basico consente agli sviluppatori di scegliere liberamente i componenti da integrare senza imporre particolari strutture o componenti. Flask è infatti ideale per applicazioni che richiedono una rapida prototipazione e uno sviluppo agile.

Nel mio progetto questo framework è stato utilizzato per realizzare un'API RESTful che consente la gestione dei dati energetici nel cloud. L'API implementa tre operazioni principali:

- Inserimento di dati: l'endpoint /report che permette di inserire i dati nel database MongoDB. Un client invia una richiesta POST con un determinato formato JSON e questi vengono inseriti nel database 'energy_in_cloud' nella collezione 'data'. Il successo dell'operazione restituisce un messaggio di conferma.
- Recupero di tutti i dati: l'endpoint /alldata risponde a richieste GET restituendo tutti i dati presenti nella collezione 'data'. I dati restituiti tramite documento vengono convertiti in una lista di dizionari Python, in particolare l'ID del documento viene convertito in stringa per facilitare la serializzazione in JSON.
- Cancellazione di tutti i dati: L'endpoint /delete-all-data gestisce richieste DELETE, elimina quindi tutti i dati dalla collezione 'data'. Al termine dell'operazione, viene restituito un messaggio che indica quanti documenti sono stati eliminati.

1.1.2 MongoDB

MongoDB [2] è un database NoSQL orientato ai documenti, noto per la sua scalabilità e flessibilità nella gestione dei dati. A differenza dei database relazionali tradizionali che utilizzano tabelle e righe, MongoDB memorizza i dati in documenti con schemi dinamici.

Con il termine NoSQL non ci riferiamo a nessun linguaggio specifico, è piuttosto un termine che viene utilizzato per descrivere un insieme di tecnologie volte alla persistenza dei dati, che operano in modo diverso rispetto ai database relazionali. Questo significa che i database NoSQL presentano una vasta gamma di caratteristiche : i dati possono essere archiviati come coppia chiave-valore, oppure possono essere conservati in tabelle e campi che non rappresentano uno schema rigido, oppure ancora in modelli che non supportano vincoli di integrità referenziale. Tutto ciò permette di gestire dati complessi e in continua evoluzione in maniera più intuitiva, rendendo MongoDB particolarmente adatto per applicazioni moderne che richiedono maggiore agilità nella gestione delle informazioni.

Nel mio progetto, MongoDB è stato utilizzato per memorizzare e gestire i dati raccolti dal framework Flask. La sua configurazione è stata effettuata tramite Docker che ha semplificato notevolmente la gestione dell'ambiente di sviluppo e di produzione. Il database del progetto è chiamato ‘energy_in_cloud’ e fa parte di una collezione denominata ‘data’.

1.1.3 Docker

Docker [3] è una piattaforma per la containerizzazione delle applicazioni, che permette di creare, distribuire ed eseguire software in ambienti isolati chiamati container. Un container possiede tutto il necessario per eseguire un'applicazione: dal codice, alle librerie, alle impostazioni di sistema. Docker consente agli sviluppatori di costruire applicazioni in modo rapido e affidabile migliorando l’efficienza e la velocità delle applicazioni. L’utilizzo dei container sta diventando sempre più popolare perché le organizzazioni stanno passando allo sviluppo dei cloud e di ambienti multi cloud ibridi. Gli sviluppatori possono quindi creare container Docker e possono lavorare direttamente con funzionalità di sistemi operativi non propri. Per garantire la sua portabilità, Docker utilizza funzionalità di isolamento del kernel LINUX. Così facendo un’applicazione containerizzata funzionerà allo stesso modo su tutti gli ambienti che sono compatibili con Docker. Di seguito sono riportate le 5 componenti principali di questa piattaforma:

- Docker Image: Un'immagine di Docker è un pacchetto che contiene il codice sorgente dell'applicazione a cui si riferisce. Comprende anche tutti gli strumenti, le dipendenze e le librerie di cui il codice ha bisogno per essere eseguito come container.
- Docker Engine: è il nucleo centrale di Docker e contiene un server deamon, un'API e un'interfaccia per la gestione diretta dei container.
- Docker Container: quando un'immagine Docker è in esecuzione diventa un Container Docker e le sue impostazioni possono essere eventualmente modificate dagli utenti.
- Docker Hub: è un registro dove si possono trovare tutte le immagini Docker che sono disponibili per il download. Qualunque utente può condividere la propria immagine o può scaricare una già presente.
- Deamon Docker: è un servizio che ha il compito di gestire e di creare le immagini Docker. Tale servizio viene eseguito su un server denominato host Docker e funge da centro di controllo di Docker stesso.

Grazie alla sua portabilità, Docker semplifica la distribuzione del software.
Nel mio progetto è stato utilizzato per containerizzare sia l'applicazione Flask che il database MongoDB permettendomi di avere un ambiente di sviluppo e produzione consistente e isolato.

1.1.4 RESTful API

Un'API [4] (Application Programming Interface), è un insieme di protocolli e di strumenti che permettono a più applicazioni di comunicare ed interagire tra loro. In pratica, un'API definisce quali sono i metodi e le procedure che consentono a due applicazioni diverse di scambiarsi dati e funzionalità, permettendo così un lavoro coordinato. Le API hanno il ruolo di ponte, Figura 1, tra chi fornisce le informazioni e chi le riceve, definendo il contenuto della chiamata del consumatore e della risposta del produttore per facilitare la comunicazione. Seguono il principio di un'architettura REST, che rende i servizi web leggeri, scalabili e interoperabili. Tale architettura si basa sul principio di statelessness secondo cui ogni richiesta API inviata dal client al server deve includere, oltre che hai dati, una serie di informazioni per poterlo comprendere e per permettere di processare la risposta. Inoltre, per essere considerata RESTful [5] , un'API deve seguire vincoli architetturali come la memorizzazione dei dati nella cache e un'interfaccia uniforme per i componenti.

Nel progetto svolto le API sono state utilizzate per la comunicazione tra il front-end e il back-end, consentendo scambi di dati affidabili. Questo tipo di approccio ha reso il sistema più flessibile e ha favorito una maggiore modularità del codice.

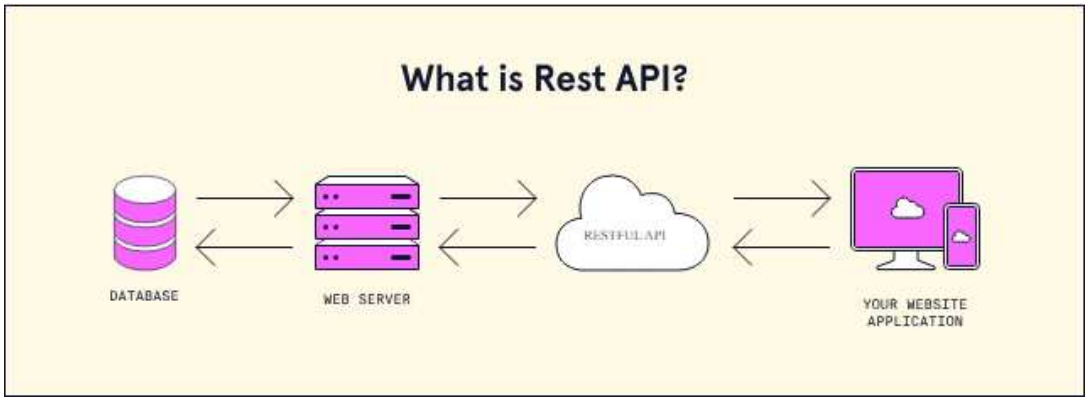

Figura 1 - Come funziona una RESTful API [6]

1.1.5 HTTP

HTTP (Hypertext Transfer Protocol) [7] è un protocollo di comunicazione per il trasferimento di dati nel WWW (World Wide Web). HTTP è un protocollo stateless che facilita lo scambio di informazioni tra il client e il server tramite una serie di richieste e di risposte. Le principali richieste utilizzate con HTTP sono di vario genere, le più frequenti sono GET,POST,PUT,DELETE. Essendo tale protocollo alla base delle API RESTful ed essendo inoltre semplice da utilizzare e compatibile con la maggior parte dei protocolli e formato dati esistenti, HTTP è stato utilizzato nel mio progetto per la comunicazione tra le schedine ESP32 e il server Flask. Le richieste HTTP inviate dall'ESP32 al server hanno permesso l'inserimento, il recupero e la cancellazione dei dati nel database.

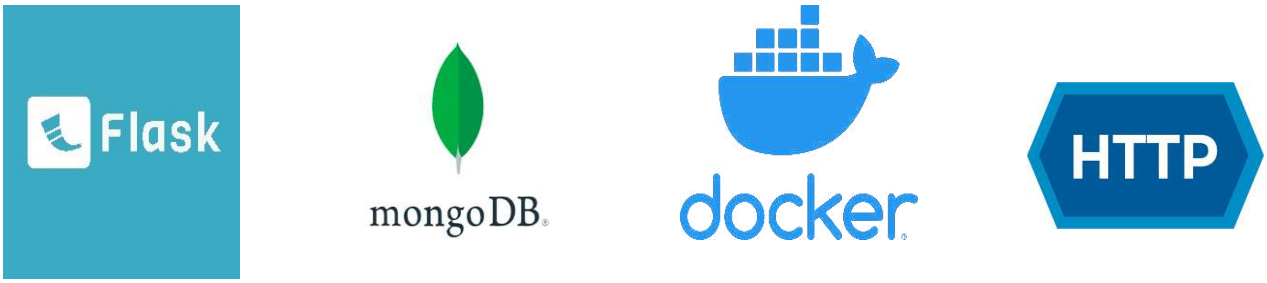

Figura 2 - Tecnologie di back-end utilizzate

1.2 TECNOLOGIE DI FRONTEND

Le tecnologie descritte di seguito sono state fondamentali per lo sviluppo di un' interfaccia utente che fosse interattiva e reattiva.

1.2.1 React

React [8] è una libreria JavaScript che serve per la costruzione di interfacce utente attraverso la creazione di componenti riutilizzabili che semplificano lo sviluppo e la manutenzione delle applicazioni web. Essendo un'applicazione a pagina singola il contenuto della pagina viene caricato direttamente dai componenti di React e non viene inviata una richiesta al server ogni volta che avviene un reindirizzamento. La sintassi che viene utilizzata per la creazione di applicazioni React è chiamata JSX (JavaScriptXML).

Nel mio progetto, React è stato utilizzato appunto per sviluppare la dashboard, che è la parte dell'applicazione con cui gli utenti interagiscono maggiormente. La struttura basata su componenti ha permesso di creare un'interfaccia modulare, scalabile e facilmente modificabile. Ogni sezione della dashboard, come i grafici, le tabelle e i moduli di input, è stata implementata come un componente React separato e solo in un secondo momento è stato integrato con gli altri.

1.2.2 Fetch

Fetch [9] è una tecnologia per fare richieste HTTP. È una funzione JavaScript che permette di fare chiamate asincrone al server per recuperare o inviare dati, senza dover ricaricare l'intera pagina. Questo ha reso l'applicazione web più dinamica e moderna facendo sì che l'interfaccia utente si aggiornarsi in risposta alle interazioni dell'utente. Fetch rappresenta una valida alternativa a XMLHTTPRequest per la gestione di chiamate HTTP perché ha una sintassi più semplice e integrata meglio nel modello ad oggetti di JavaScript.

Nel mio progetto, Fetch è stato utilizzato per comunicare con l'API dell'applicazione Flask.

Ad esempio, quando un utente interagisce con la dashboard, come il caso di una richiesta di aggiornamento delle informazioni, Fetch viene utilizzato per inviare tale richiesta al server.

1.3 Ambienti di sviluppo

Gli ambienti di sviluppo integrato (IDE) sono utilizzati per lo sviluppo efficiente e collaborativo del software. Un buon ambiente di sviluppo è fondamentale per la scrittura del codice perché

questi offrono funzionalità come il debug o il compilamento automatico che sono di grande supporto per gli sviluppatori.

1.3.1 Visual Studio Code

Visual Studio Code (VS Code) [10] è un editor noto per essere leggero, ma estremamente potente, grazie alla vasta gamma di estensioni disponibili che ampliano le sue funzionalità di base. Le caratteristiche chiave di VS Code includono:

- un'interfaccia chiara e intuitiva che consente agli sviluppatori di potersi concentrare sul codice e di non avere distrazioni
- la formattazione automatica del codice
- il rilevamento degli errori
- il supporto per il completamento automatico
- altre funzionalità per velocizzare il lavoro degli sviluppatori

Questo strumento si è rivelato essenziale per il mio progetto, in quanto mi ha permesso di scrivere e modificare il codice Python dell'applicazione Flask con grande efficienza. Ha permesso di configurare e gestire i file Docker direttamente dall'editor semplificando la gestione degli ambienti di sviluppo e produzione. Inoltre, la gestione dei pacchetti tramite il file requirements.txt è stata resa semplice e intuitiva. VS Code è uno strumento decisamente versatile, capace di adattarsi alle diverse esigenze del progetto, rendendo lo sviluppo più fluido e organizzato.

1.3.2 GitHub

GitHub [11] è una piattaforma di hosting per il controllo del codice sorgente e la collaborazione, basata su Git [12]. Fornisce molte funzionalità per Lo sviluppo dei progetti software, che vanno dalla semplice gestione del codice sorgente alla possibilità di collaborare su larga scala con altri sviluppatori. Il successo di Git è dovuto al fatto che, a differenza dei sistemi di controllo centralizzati come CVS o SVN che offrono un unico repository centrale con l'intera cronologia delle versioni, Git ha un'architettura distribuita. Inoltre offre sicurezza garantendo che ogni modifica venga apportata in modo autenticato. Questo permette di tenere traccia di quali modifiche sono state effettuale, da chi e quando.

Nel mio progetto, GitHub ha svolto un ruolo cruciale per la condivisione del codice con i miei colleghi tramite repository privati. La revisione del codice è stata strutturata grazie alla possibilità di creare e gestire pull request, migliorando la qualità del software prodotto.

Inoltre, le funzionalità di issue tracking hanno aiutato a tenere traccia dei bug e delle richieste di nuove funzionalità, rendendo il processo di sviluppo più organizzato e gestibile.

1.3.3 Thonny

Thonny [13] è un ambiente di sviluppo integrato (IDE) per Python, in particolare della versione Python 3.10; dunque, basta un semplice programma di installazione per poterlo usare. Offre inoltre una funzionalità di debugger semplice da utilizzare perché segue la struttura del programma oltre che le linee di codice. Thonny si è rilavato uno strumento ideale per la prototipazione e la sperimentazione rapida, permettendomi di testare le funzionalità direttamente sulle schede ESP32 senza la complessità di ambienti di sviluppo più avanzati.
Nel mio progetto, Thonny è stato utilizzato principalmente per la scrittura del codice relativo ai casi d'uso realizzati sulle due schedine fornite. La sua interfaccia "user-friendly" ha facilitato l'implementazione e il debugging del codice.

Figura 3 - Ambienti di sviluppo utilizzati

2. Progettazione della soluzione

2.1 Introduzione alla progettazione della soluzione

Il progetto sviluppato si basa su un lavoro svolto precedentemente che si proponeva di progettare a grandi linee una soluzione approssimata per la condivisione e la ripartizione equa dell'energia che viene utilizzata da una comunità. In realtà, il concetto alla base è quello di condividere nel complesso la stessa energia pulita senza preoccuparsi di quale sia il momento migliore della giornata per 'consumare'.

L'obiettivo finale, infatti, è stato quello di riuscire a ripartire equamente i costi che vengono attribuiti ai diversi utenti considerando solo il rapporto tra il consumo del singolo e il consumo totale, trascurando quale fosse la fonte di energia. Per far ciò è stata condotta un'analisi dettagliata dei requisiti e delle specifiche basata su uno studio dettagliato e approfondito delle esigenze dell'utente.

Per questa prima fase l'approccio si è basato su un modello a cascata in cui il processo è strutturato in modo che ogni fase successiva dipende dalla corretta implementazione di quella precedente. Il modello a cascata vuole che ogni fase debba essere completata prima dell'inizio della fase successiva in modo che l'input di ognuna sia l'output della fase precedente. L' output di ogni fase consiste in un insieme di documenti, verificati e approvati dall'intero gruppo che lavora sul progetto, che identificano la fase e gli obiettivi raggiunti. I vantaggi di questo modello è che vengono distinte tra di loro le fasi rendendo l'output di ognuna facilmente verificabile.

2.1.1 Analisi dei requisiti e delle funzionalità

Il punto di partenza per la progettazione è stata l'identificazione e la raccolta dei requisiti funzionali e non funzionali. Si è cercato quindi di individuare ogni funzionalità e servizio che l'applicazione web avrebbe dovuto fornire per soddisfare l'obiettivo del progetto e le necessità dell'utente che lo userà. Questa fase prende il nome di Ingegneria dei requisiti e serve a trovare i servizi che il cliente richiede, i vincoli sotto i quali il sistema deve operare e le descrizioni delle funzionalità che questo deve fornire. Capire quali sono gli obiettivi da raggiungere è fondamentale per riuscire a ripartire le attività e per organizzare il lavoro affinché sia possibile concluderlo entro il termine stabilito.

Dall'analisi condotta sono state individuate una serie di funzionalità:

- Creazione e gestione di una GUI semplice ed intuitiva cosicché fosse possibile per gli utenti visualizzare l'andamento dei consumi.
- Avere la possibilità di visualizzare il resoconto sul consumo totale di energia e la ripartizione in termini di costi per i vari utenti.
- Poter tenere traccia anche per periodi passati della quantità di energia consumata sia quotidianamente che mensilmente.

Per facilitare la visualizzazione dei requisiti principali sono stati usate delle tecniche di modellazione che hanno permesso la descrizione delle interazioni tra gli utenti e il sistema: i casi d'uso [14]. Questi ultimi insieme alle storie utente [15] permettono di racchiudere diversi scenari in un caso unico e di mostrare come il sistema è capace di gestire delle eccezioni o delle situazioni inaspettate.

Un esempio di uno dei casi d'uso utilizzato è il seguente:

CASO D'USO 1	Richiesta dati sul consumo di energia
DESCRIZIONE	Come abitante di un condominio voglio visualizzare i dati relativi al consumo di energia di un giorno in particolare di un mese in modo che possa sapere quanto ho consumato.
ORDINARY SEQUENCE	1.L'attore clicca sul giorno nel datapacker della dashboard che gli permette di visualizzare il grafico relativo a quel giorno. 2.La dashboard richiede tali dati al server tramite una richiesta GET. 3.Il server recupera i dati dal database e li restituisce alla dashboard. 4. La Dashboard mostra i dati all'utente.

Tabella 1 – Caso d'uso 1

I casi d'uso sono caratterizzati da descrizioni in linguaggio naturali che servono per identificare:

- Funzionalità principali
- Pre-condizioni e post-condizioni
-Flusso di eventi (scenari)

-Condizioni di errore e flussi alternativi

Una volta scelti gli attori del sistema (utenti che interagiscono) e gli scenari che si vogliono considerati allora vengono sviluppati i casi d'uso e vengono messe in conto tutte le casistiche possibile affinché alcuna situazione possa risultare inaspettata.

Per rendere il caso d'uso ancora più chiaro possiamo usare un Sequence Diagram [14] .

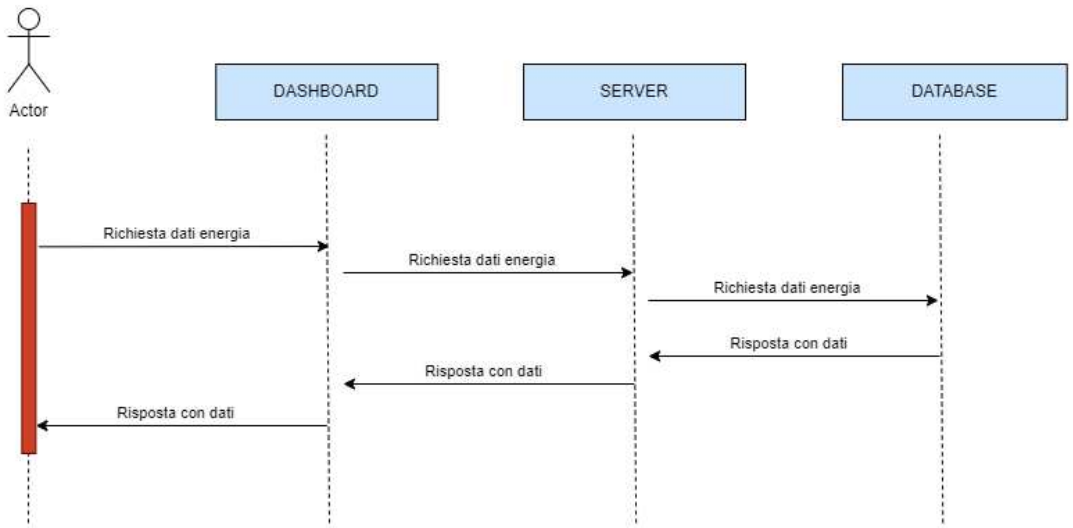

Figura 4 - Sequence Diagram

2.2 Progettazione dei casi d'uso

La progettazione dei casi d'uso è stata una fase cruciale per la realizzazione della soluzione in quanto essa ha permesso la produzione di dati reali che sono stati poi analizzati ed elaborati per arrivare all'obiettivo finale. Il kit usato per creare i casi d'uso è stato il Super Starter Kit UNO R3 Project [16] che conteneva tutti i componenti necessari per l'implementazione di piccoli casi in miniatura che potessero simulare un consumo di energia costante. Nel seguito verranno presentati i casi realizzati e dopo verranno descritti i singoli componenti utilizzati.

CASO 1: 8 LED in parallelo, collegati a un chip convertitore da seriale a parallelo chiamato 74HC595, che possono essere accesi in sequenza da un bottone con configurazione pull-down e spenti tutti insieme da un secondo bottone con configurazione pull-up. L'utilizzo del chip rende leggermente più lento il processo che comanda il led ma permette di collegare gli 8 pin direttamente a esso senza doverne utilizzare 8 diversi sulla schedina esp32.

CASO 2: Controllo di un motore DC facendo uso di un modulo di alimentazione elettrica per la corrente di alimentazione del motore che viene collegato a un chip L293D. Il motore si accende solo mentre viene tenuto premuto un bottone con configurazione pull-up.

In entrambi i casi d'uso è stata utilizzata la stessa logica per inviare un messaggio contenente i dati necessari per l'elaborazione delle informazioni da fornire alla dashboard. Tale logica sarà descritta nel capitolo successivo.

2.2.1 Componenti utilizzati

LED (Light Emitting Diodes): è un componente che ha due terminali, un Anode (+) e un Catode (-), che emette luce con un intensità proporzionale alla quantità di corrente che lo attraversa. In un LED la corrente può scorrere solo dall'anodo al catodo altrimenti esso si comporterà come un circuito aperto.

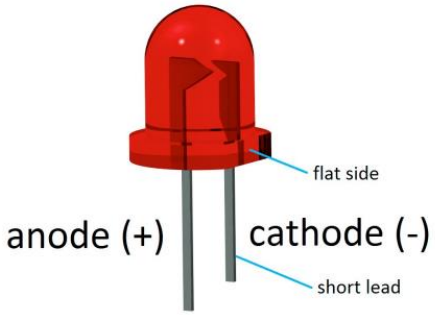

Figura 5 - Componente: LED

Sul datasheet di un led è possibile leggere diverse specifiche tra cui le più importanti:

- La tensione diretta (V_F): Tensione ai capi del led che è quasi indipendente dalla corrente.
- Corrente massima (I_F): Limite della corrente che può attraversare il led affinché questo non venga danneggiato in modo permanente.

Nel caso d'uso 1 sono stati usati in serie con i led dei resistori da 220 ohm per assicurarci che la corrente che avrebbe attraversato i led fosse sempre minore di 20mA (I_F dei led usati). Il calcolo eseguito per scegliere tale valore di resistenza è stato il seguente:

Supposto di volere un valore pari 5mA per la corrente, sappiamo che $V_R = R * I$ (legge di Ohm).

Nel nostro caso $V_R = V_{HIGH} - V_F$ (Legge di Kirchoff)

Quindi $(V_{HIGH} - V_F) = R * I \rightarrow R = (V_{HIGH} - V_F) / I$

Essendo $V_{HIGH} = 3.3V$, $V_F = 2.2V$ e $I = 20mA$

$R = (3.3 - 2.2) V / 0.005 A = 220 \Omega$

CHIP 74HC595: Questo chip è un registro a scorrimento che permette di comandare separatamente ognuna delle sue 8 zone di memoria. Ognuna di queste può registrare valori pari a 1 o a 0. Per poter cambiare i valori vanno utilizzati i pin 'Data' e 'Clock' sul chip. Quando entrambi vengono inseriti, si abilita il pin 'Latch' e il chip memorizza tutti i valori copiandoli sul

registro di latch. Questo chip ha un pin di abilitazione output OE che viene utilizzato per abilitare o disabilitare tutti gli output in una unica volta.

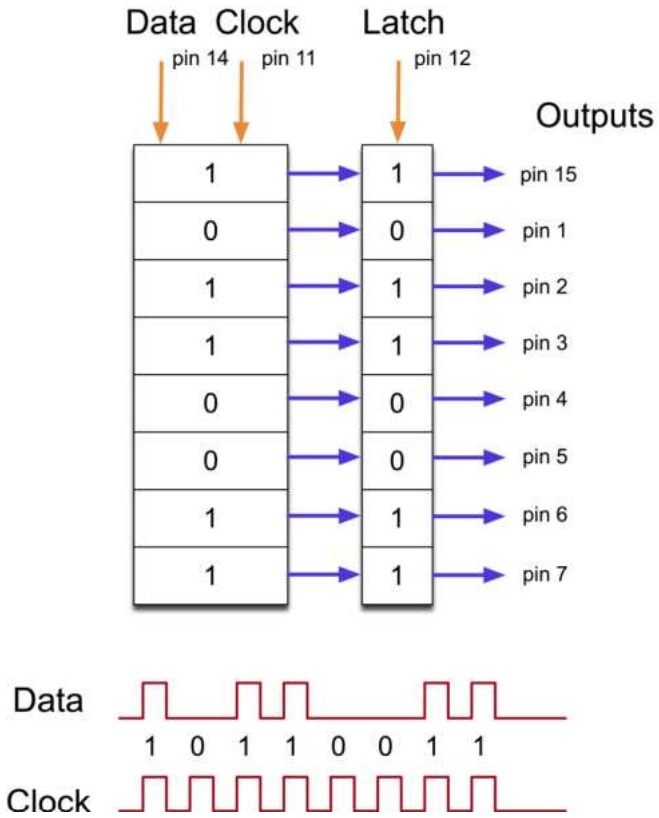

Figura 6 – Output di uno ShiftRegister 74HC595

BOTTONE: Un pulsante funge da interruttore che chiude il circuito attraverso i suoi terminali. Oggi vengono utilizzati pulsanti che dispongono di resistori incorporati la cui modalità di lavoro può essere impostata direttamente dal pin. Le due modalità, entrambe utilizzate nei casi d'uso, sono:

- PULL-UP (resistore di pull-up): quando il bottone viene rilasciato, il resistore porta la tensione del pin di ingresso al livello alto.
- PULL-DOWN (resistore di pull-down): quando il bottone viene rilasciato, il resistore porta la tenzione del pin di ingresso a massa.

MOTORE DC: Un motore DC è un motore a corrente continua che produce una forza meccanica che permette il movimento. I magneti che si trovano all'interno del motore formano il campo magnetico secondo la forma di Lorentz.

MODULO DI ALIMENTAZIONE ELETTRICA: Il motore DC utilizza più energia di quella che può fornire direttamente una schedina ESP32. Infatti, se proviamo a collegare il motore

direttamente alla scheda è molto probabile che questa si rovini permanentemente. Per risolvere questo problema viene utilizzato un modulo di alimentazione elettrica in modo che l'alimentazione fornita al motore sia quella di una batteria esterna. Questo modulo permette di impostare il voltaggio di outoput spostando il ponticello ai pin corrispondenti.

Figura 7 – Componente: modulo di alimentazione elettrica

L293D: Questo chip permette di controllare due motori indipendentemente. Al suo interno contiene una serie di componenti che permettono di decidere il verso del motore e la sua velocità. L293D è servito per fornire una corrente guida bidirezionale con un voltaggio compreso tra 4.5V e 36V e una corrente pari a 600mA. Con i giusti dati in input, ogni paio di attuatori forma un full-H che innesca un certo senso di rotazione.

Figura 8 – Pinout del chip L293D

ACS712: Questo sensore di corrente utilizza l'effetto Hall per rilevare il valore della corrente. Nel progetto viene utilizzato per leggere la corrente del circuito e quindi per calcolare il consumo di energia da inviare.

2.3 Progettazione del front-end

La progettazione del front-end ha rappresentato una fase importante nello sviluppo dell'applicazione web. In questo capitolo di vuole descrivere il processo di progettazione che ha preceduto quello dell'implementazione del front-end, con particolare attenzione alla User Expierence (UX) e all'interfaccia utente (UI).

La prima parte di progettazione consiste in un'analisi delle esigente dell'utente. In questa fase infatti vanno comprese le aspettative dell'applicazione e vanno tenuti presenti i requisiti venuti fuori dall'analisi svolta precedentemente. Gli utenti preferiscono un'interfaccia intuitiva, reattiva e gradevole alla vista, che abbia una navigazione semplice e un veloce accesso ai contenuti.

In un secondo momento è stata decisa l'architettura delle informazioni che ha visto protagonista l'organizzazione e la strutturizzazione delle informazioni che sarebbero state mostrate all'interno dell'interfaccia utente. Questa fase ha richiesto anche uno studio dei colori, elemento chiave nella progettazione del front-end, che ha permesso una buon contrasto e quindi una buona leggibilità. Il layout dell'interfaccia è stato progettato in modo che fosse flessibile e reattivo, utilizzando un approccio mobile-first in modo che potesse garantire che l'applicazione fosse accessibile su dispositivi di diverse dimensioni. Si è utilizzata una struttura a griglia per l'allineamento degli elementi con ampi margini e spaziature per evitare che le informazioni si sovrapponessero.

L'ultima fase è stata quella della prototipazione che ha coinvolto strumenti come Figma per creare wireframe e mockup interattivi. La realizzazione di questi prototipi ha permesso di svolgere test di usabilità prima della fase di sviluppo.

In sintesi la progettazione del front-end è stata un processo iterativo iniziato dall'analisi delle esigente degli utentii fino alla creazioni di prototipi. Il risultato è stato un ottima base per la fase di implementazione descritta nel prossimo capitolo.

2.4 Progettazione del back-end

Il back-end svolge un ruolo fondamentale nello sviluppo di un'applicazione perché è il responsabile di processare ed elaborare i dati che recupera, manipola e memorizza nel database. Il back-end, infatti, gestisce tutte quelle che sono le richieste provenienti dal front-end dell'applicazione, garantendo che siano indirizzate correttamente ai componenti giusti.

La progettazione del back-end si è ovviamente basata sull'analisi dei requisiti e delle funzionalità volte a soddisfare l'utente, in particolare l'attenzione è ricaduta sulla sicurezza e sulla stabilità delle tecnologie utilizzate. Durante questa fase è stato svolto uno studio attento delle innovazioni tecnologiche più recenti in modo da adottare soluzioni che siano all'avanguardia e che potessero garantire la longevità del prodotto. Questo approccio ha reso

il prodotto innovativo e ha inoltre permesso al team di rimanere aggiornato sugli sviluppi tecnologici.

Il prototipo è stato suddiviso in tre parti: il client, il server e il database. Ognuno di questi componenti è stato containerizzato utilizzando Docker in modo da ottimizzare l'implementazione e la gestione del sistema. L'utilizzo di Docker ha aumentato la scalabilità e la portabilità del sistema, permettendo al prototipo di funzionare in ambienti anche diversi tra loro.

Uno degli strumenti più importanti del back-end è il database. Per il progetto è stato scelto MongoDB, in cui i dati vengono memorizzati come documenti JSON in modo che le query sottoposte siano rapide. Il server è stato implementato con Flask, insieme alle API, in modo che si occupi di elaborare le richieste provenienti dal client e di recuperare i dati provenienti dal database. Le operazioni sul database sono infine gestite tramite le API, con endpoint dedicati per ogni tipo di operazione.

3. Implementazione e descrizione delle funzionalità

Questo capitolo vuole descrivere la logica utilizzata per implementare il sistema partendo da quella dei casi d'uso per simulare situazioni reali fino a quella del back-end e del front-end.

3.1 Sviluppo dei casi d'uso

Nello sviluppo dei casi d'uso è stata scelta una comunicazione in locale per la comunicazione tra i componenti. Tale scelta è scaturita dalla necessità di garantire tempi di risposta rapidi e una maggiore affidabilità e sicurezza della trasmissione dei dati. In particolare:

- Tempi di risposta rapidi: Una comunicazione locale diminuisce la latenza perché i dati non devono attraversare la rete internet globale.
- Affidabilità della rete: Rispetto a una connessione internet, una rete locale è meno soggetta a interruzioni perché non è influenzata da fattori esterni (congestione dei dati, attacchi informatici ecc.).
- Sicurezza dei dati: La trasmissione di dati attraverso una rete internet può esporli a rischi di sicurezza come intercettazioni o rischi non autorizzati.

Per la comunicazione tra i componenti del sistema è stato scelto il protocollo HTTP. Questo protocollo ha una serie di vantaggi che si sono rilevati ideali per l'implementazione:

- Compatibilità: HTTP è uno standard universale, garantisce quindi compatibilità tra diversi dispositivi permettendo ai componenti di variare per il sistema operativo e hardware.
-Efficienza nella trasmissione dei dati: HTTP permette di scambiare dati in formato JSON. Questo formato è ideale per l'invio di comandi e la ricezione di risposte tra dispositivi, riduce il carico sulla rete e migliora, nel complesso, le prestazioni del sistema.
- Semplicità dell'implementazione: Il protocollo HTTP consente una rapida scrittura del codice grazie alle molteplici librerie che offrono la maggior parte dei linguaggi di programmazione.

Questa soluzione ha permesso un controllo completo sulla rete e la flessibilità necessaria per adattarsi a diverse esigenze applicative.

3.1.1 Implementazione del codice relativo ai casi d'uso

Il codice è stato strutturato secondo moduli e classi al fine di migliorare la modularità, la leggibilità e la manutenibilità. Ogni classe si riferisce a un singolo componente e contiene le funzioni necessarie per gestirlo.

3.1.1.1 Caso d'uso 1

Il primo caso d'uso, la cui progettazione è descritta nel capitolo 2 paragrafo 2.2, è diviso nelle seguenti classi: boot.py, main.py, ShiftRegister.py. Si descrivono nel seguito le funzioni più rilevanti.

```
def updateShiftRegister(self, leds):
    self.latch_pin.value(0)
    for i in range(8):
        self.clock_pin.value(0)
        self.data_pin.value(leds & (1 << i))
        self.clock_pin.value(1)
    self.latch_pin.value(1)
```

Figura 9 - Funzione updateShiftRegister() della classe ShiftRegister.py

Nella figura [9] è riportata la funzione updateShiftRegister della classe ShiftRegister usata per accendere e spegnere i led. Il suo funzionamento è il seguente:

- self.latch_pin.value(0) : Imposta il pin di latch a basso per iniziare la trasmissione dei dati.
- for i in range(8): Itera su ciascun bit (LED) da inviare al registro a scorrimento
    - self.clock_pin.value(0) : Imposta il pin di clock a basso
    - self.data_pin.value(leds & (1 << i)) : Scrive il bit corrente sul pin dei dati
    - self.clock_pin.value(1) : Alza il pin di clock per memorizzare il bit corrente
- self.latch_pin.value(1) : Alza il pin di latch per caricare i dati nel registro a scorrimento

Nel main vengono poi definiti i pin di tutti i componenti del sistema e inizializzate le variabili necessarie per il calcolo del consumo. La funzione che calcola il consumo del sistema è leggi consumo() che fa uso della funzione read_current(). Quest'ultima restituisce il valore della corrente che viene letto dal sensore di corrente ACS712.

3.1.1.2 Caso d'uso 2

Il secondo caso d'uso, anch'esso descritto nel capitolo 2 paragrafo 2.2, è diviso nelle seguenti classi: boot.py, main.py, motor.py. Si descrivono nel seguito le funzioni più rilevanti.

```
def motorMove(self, speed, direction, freq):
    self.pwm_pin.freq(freq)
    duty_cycle = int(speed / 100 * 65536)
    if duty_cycle > 65535:
        duty_cycle = 65535
    self.pwm_pin.duty_u16(duty_cycle)
    if direction == 1:
        self.cw_pin.value(1)
        self.acw_pin.value(0)
    else:
        self.cw_pin.value(0)
        self.acw_pin.value(1)
```

Figura 10 – Funzione motorMove() della classe Motor.py

Nella figura [7] è riportata la funzione motorMove della classe motor usata per gestire il motore. Il suo funzionamento è il seguente:

- self.pwm_pin.freq(freq): Imposta la frequenza del PWM al valore specificato,
- duty_cycle = int(speed / 100 * 65536) : Calcola il ciclo di lavoro del PWM basato sulla velocità (percentuale).
- if duty_cycle > 65535: Verifica che il ciclo di lavoro non superi il massimo consentito (16-bit, cioè 65535).
- self.pwm_pin.d.uty_u16(duty_cycle): Imposta il ciclo di lavoro del PWM (duty cycle) calcolato.
- if direction == 1: se direzione è 1, imposta il motore a girare in senso orario o altrimenti in senso antiorario impostando i relativi pin a 1 o a 0.

Per controllare la velocità del motore viene utilizzato il PWM che determina la quantità di potenza fornita permettendone il controllo. Sono usate le stesse funzioni descritte nel paragrafo precedente per leggere il valore della corrente e per calcolare il consumo totale.

3.1.2 Implementazione del codice relativo alla richiesta post tramite HTTP

Il codice implementato per la gestione delle richieste HTTP e per creare i dati nel formato voluto si trova nel file 'main.py'. All'interno del file sono presenti le funzioni necessarie per creare, inviare e gestire le richieste HTTP che vengono fatte dalla schedina esp32 verso il server, formattate correttamente secondo le esigenze del database. Per poter gestire le richieste e la formazione dei dati sono state importate le seguenti librerie: urequests, ujson e time.

- [17] 'urequests' per gestire le richieste HTTP

- [18] ‘ujson’ per la manipolazione dei dati in formato JSON

- [19] ‘time’ per permettere la sincronizzazione temporale

Per la formazione del pacchetto dei dati in formato JSON è stata usata la funzione ‘create_data()’ che ha come valore di ritorno un dizionario ordinato contenente le seguenti informazioni : lo user, il giorno, la data, l’ora e il consumo.

```
def get_current_date(): #data corrente in formato 'YYYY-MM-DD'
    current_time = time.localtime()
    return '{}-{:02d}-{:02d}'.format(current_time[0], current_time[1], current_time[2])

def get_current_hour(): # ora corrente in formato 'HH:MM'
    current_time = time.localtime()
    return '{:02d}:{:02d}'.format(current_time[3]-1, current_time[4]+32)

def create_data(energia):
    data = OrderedDict()
    data["name"] = user_name
    data["day"] = get_current_date()
    data["data"] = [{"hour": get_current_hour(), "value": energia}]
    return data
```

Figura 11 - Funzioni per creare il pacchetto data nel main

Mentre lo user è una variabile statica, la data e l’ora sono state recuperate utilizzando la funzione localtime() della libreria time.

Infine, per inviare la richiesta è stata utilizzata la funzione:

‘urequests.post(url, data=ujson.dumps(og), headers=headers)’

dove:

- url: *'HTTP://192.168.137.1:5000/report'* cioè, l'indirizzo del server a cui vogliamo inviare i dati con la porta e l’end point relativo.

- data=ujson.dumps(og): il pacchetto di dati (og) in formato JSON

- headers : Il tipo di contenuto.

Per gestire eventuali errori durante l’invio, tale funzione si trova all’interno di un blocco try-catch.

```python
def http_post(url, og):
    headers = {'Content-Type': 'application/json'}
    try:
        print("Sending data:", og)
        response = urequests.post(url, data=ujson.dumps(og), headers=headers)
        print('Status:', response.status_code)
        print('Content:', response.text)
        response.close()

    except Exception as e:
        print("Exception during HTTP POST request:", e)
```

Figura 12 - Funzione HTTP_post() nel main

3.2 Sviluppo del front-end

Il front-end è stato implementato utilizzando React.js in modo da avere componenti separati che fossero riutilizzabili in momenti diversi.

Nel progetto è stato creato un componente principale utilizzato per integrare tutti i componenti secondari descritti nel paragrafo seguente.

3.2.1 Creazione dei componenti

Tutti i componenti secondari che compongono il front-end sono raggruppati in una cartella denominata 'components'. In questo paragrafo ne vengono descritte le singole funzionalità e ne viene spiegato il codice.

Il primo componente, 'Fetch.js', è utilizzato nel progetto per recuperare i dati dal database. Esso contiene la funzione fetch che effettua le richieste HTTP da un client web a un server, utilizzando delle API.

```javascript
const Fetch = async () => {
    try {
        const response = await fetch('http://localhost:5000/alldata');
        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }
        const data = await response.json();
        return data;
    } catch (error) {
        console.error('There was a problem with the fetch operation: ', error);
    }
};

export default Fetch;
```

Figura 13 - Funzione fetch del modulo Fatch.js

Nella figura [13], viene mostrata la funzione fetch che viene utilizzata per inviare una richiesta GET all'URL HTTP://localhost:5000/alldata del server locale. Nel caso di un'esecuzione della

funzione andata a buon fine, questa restituisce i dati. In caso contrario viene visualizzato a schermo un messaggio d'errore. Gli altri componenti sono stati creati in modo da riuscire a soddisfare le funzionalità che sono venute fuori dallo studio delle esigenze del cliente individuate durante l'analisi dei requisiti. Queste sono:

- Filtraggio Temporale: opzione per selezionare giorno, mese e anni di cui visualizzare i dati di consumo.
- Visualizzazione Grafica: creazione di grafici facili da interpretare che rappresentino i dati relativi ai consumi energetici di tutti gli utenti. Tali grafici saranno campionati per una visualizzazione ancora più dettagliata e leggibile.
- Decremento della batteria: decremento di una batteria in relazione al consumo effettuato dagli utenti nel corso del tempo.
- Tasto per generare un file: bottone per avere la fattura mensile in formato txt.

Per la selezione della data è stato implementato un tag input. Gli utenti possono specificare il giorno di interesse per il monitoraggio dei consumi energetici. I dati vengono filtrati in base alla data e vengono poi passati al componente del grafico, che li utilizza per creare una rappresentazione visiva dei consumi energetici nel tempo. In questo modo tutti gli utenti possono visualizzare l'andamento dei propri consumi. Inoltre, per agevolare l'utente nel monitoraggio, sono state aggiunte funzionalità come la visualizzazione della somma giornaliera e mensile dei consumi energetici.

3.2.2 Integrazione dei componenti

Nel file SelectDay.js sono stati integrati tutti i componenti e sono state create alcune funzioni che permettessero il comportamento descritto. La prima parte del codice inizializza lo stato dei componenti con i dati necessari e imposta gli effetti collaterali. Vengono infatti inizializzati: *data* per memorizzare i dati, *selectedDay* per la data selezionata, *batteryCharge* per il livello di carica della batteria, *powerToPay* per l'energia da pagare, e *monthlySum* per la somma mensile del consumo energetico. Quando l'utente seleziona un giorno viene chiamata la funzione fetchData() per aggiornare i dati e questi vengono filtrati solo per la data scelta, Figura 14.

```
const filtered = data?.filter(item => item.day === selectedDay);
```

Figura 14 - Filtraggio dei dati in base alla data selezionata

Per la visualizzazione dei dati sul grafico è necessario che questi siano aggregati. Per far ciò viene creato un array ordinato di tutti i dati per ora e per utente, Figura 15.

```
const allData = {};
filtered?.forEach(item => {
  item.data.forEach(d => {
    if (!allData[d.hour]) {
      allData[d.hour] = { hour: d.hour };
    }
    allData[d.hour][item.name] = parseFloat(d.value);
  });
});

const allDataArray = Object.values(allData).sort((a, b) =>
    new Date(`1970-01-01T${a.hour}:00`).getTime() - new Date(`1970-01-01T${b.hour}:00`).getTime());
```

Figura 15 - Aggregazione dei dati per la visualizzazione grafica

Altre funzioni sono utilizzate per gestire i cambiamenti della somma giornaliera e mensile. Una delle funzionalità più importanti dell'applicazione è la generazione del file relativo alla fattura. Per generare tale file è stata implementata la funzione 'generateInvoice' che crea un oggetto 'Date' basato sul giorno selezionato ed estrae il mese. Individuato il mese di cui si vuole la fattura, la funzione assegna ad una variabile il valore totale dell'energia consumata nel mese e converte poi in un numero a virgola fissa il valore di powerToPay. Per la parte del file che ripartisce equamente i costi, viene fatto un calcolo in base al consumo dei singoli utenti considerando solo i dati che appartengono al mese della fattura. La creazione del file avviene nel seguente modo:

- **fileName**: Crea il nome del file basato sul mese.
- **element**: Crea un elemento a (link) per simulare il download del file.
- **file**: Crea un oggetto Blob contenente il contenuto della fattura.
- **URL.createObjectURL(file)**: Genera un URL temporaneo per il file Blob.
- **element.click()**: Simula un click sul link per avviare il download del file.
- **document.body.removeChild(element)**: Rimuove l'elemento dopo il download.

```
const fileName = `FATTURA_${month.toUpperCase()}.txt`;
const element = document.createElement("a");
const file = new Blob([fileContent], { type: 'text/plain' });
element.href = URL.createObjectURL(file);
element.download = fileName;
document.body.appendChild(element);
element.click();
document.body.removeChild(element);
```

Figura 16 - Creazione del file in formato txt

Infine, troviamo la parte di rendering del componente JSX.

3.3 Sviluppo del back-end

La sezione di back-end è formata dall'insieme di diverse cartelle, contenenti i pacchetti e le dipendente dei framework utilizzati. Tutte le dipendenze devono essere installate ogni volta che il progetto viene avviato in modo da garantirne il corretto funzionamento. Il progetto è stato in gran parte semplificato grazie all'utilizzo di Docker, il quale ha evitato l'istallazione di una macchina virtuale con il sistema operativo UNIX. La scelta di utilizzare Docker ha ottimizzato il processo di sviluppo, consentendo una gestione efficiente dell'ambiente di sviluppo e riducendo la complessità del deployment del prototipo. Come già anticipato nel paragrafo relativo allo sviluppo dei casi d'uso, il prototipo viene eseguito in un ambiente locale, ciò implica che i dati presenti nel database saranno salvati sulla macchina locale.

3.3.1 Costruzione e avvio del server

Dopo aver installato sulla macchina l'ultima versione di Python, è stato installato il gestore dei pacchetti Python e successivamente sono stati installati i pacchetti fondamentali per lo sviluppo del progetto tramite il comando 'python3 -m ensurepip'. Il passo successivo è stata l'introduzione dei pacchetti nel codice. Il primo importato è stato flask che ha permesso la creazione di un server e la sua esecuzione. È stato importato il versatile pacchetto pymongo, il quale si è dimostrato determinante per stabilire la connessione al database e per eseguire operazioni su di esso. Per far sì che il server accettasse richieste cross-origin, è stato incluso flask_cors . L'utilizzo di Flask come framework ci ha fornito una solida base su cui costruire il back-end del prototipo, consentendoci di gestire facilmente le richieste HTTP. Il modulo pymongo, poi, si è rivelato uno strumento fondamentale per interagire con il database MongoDB, garantendo un'efficace gestione dei dati e delle operazioni di query.

```
def post(self):

    content_type = request.headers.get('Content-Type')
    if(content_type == 'application/json'):
        try:
            json_data = request.get_json()
            db.insert_one(json_data)
            return jsonify({"success": True, "message": "Data insert successfully"})
        except Exception as e:
            return jsonify({"success": False, "error": str(e)})
    else:
        return 'Content-Type not supported!'
```

Figura 17 - Metodo per una richiesta POST

```python
def get(self):
    try:
        cursor = db.find({})
        data_list = []

        for document in cursor:
            document['_id'] = str(document['_id'])
            data_list.append(document)

        return data_list
    except Exception as e:
        return jsonify({"success": False, "error": str(e)})
```

Figura 18 - Metodo per una richiesta GET

```python
def delete(self):
    try:
        result = db.delete_many({})
        return jsonify({"success": True, "message": f"{result.deleted_count} records deleted"})

    except Exception as e:
        return jsonify({"success": False, "error": str(e)})
```

Figura 19 - Metodo per una richiesta DELETE

Nella Figura [17] è definito il metodo “post”, invocato quando si riceve una richiesta POST. Il funzionamento è il seguente: se il tipo di contenuto è JSON, il metodo estrae i dati dalla richiesta, li inserisce nel database e restituisce un messaggio JSON di successo. Se si verifica un errore durante l'elaborazione della richiesta, viene restituito un messaggio JSON di errore. Invece, nella Figura [18] è definito il metodo GET, che viene chiamato quando si riceve una richiesta GET. Questo metodo crea un cursore per scorrere tutta la collezione nel database e successivamente, itera su ciascun dato restituito dal cursore aggiungendolo alla lista creata precedentemente. Infine, restituisce la lista di dati come risposta.

Infine, nella Figura [19] viene mostrato il metodo DELETE che permette di cancellare tutti i dati presenti nel database e restituisce il numero dei elementi eliminati in caso di successo o un messaggio di errore in caso di malfunzionamento.

La definizione di classe è stata fondamentale per la creazione degli endpoint. Sono stati creati endpoint distinti per ciascun metodo:

- l'endpoint /report è stato associato al metodo POST e consente l'inserimento dei dati
- l'endpoint /alldata è stato collegato al metodo GET e permette il recupero di tutti i dati disponibili
- l’endpoint / delete-all-data è stato collegato al metodo DELETE e permette di eliminare tutti i dati presenti nel database.

Per avviare il server in locale è stato utilizzato il metodo run() che resta in ascolto sul numero di porta passato come parametro. Per poter accedere o caricare i dati è sufficiente collegarsi all'URL HTTP://localhost:5000 con i relativi endpoint.

3.3.2 Docker-file e Docker-compose

Al fine di creare un ambiente riproducibile su qualsiasi sistema host, indipendentemente dalla sua configurazione, è stato scritto un Dockerfile. Quest'ultimo contiene le istruzioni utili per la creazione dell'immagine Docker dell'applicazione. Nel Dockerfile vengono definite le dipendenze dell'applicazione, come riportato in Figura [20], viene configurato l'ambiente di esecuzione e sono specificati i comandi per avviare l'applicazione.

```
dnspython==2.6.1
Flask==3.0.2
Flask-RESTful==0.3.10
flask_cors==4.0.0
pymongo==4.6.2
```

Figura 20 - Docker file

```
FROM python:latest

ADD . /app

WORKDIR /app

COPY requirements.txt .
COPY api.py .

RUN pip3 install -r requirements.txt

COPY . .

ENV MONGO_URI="mongodb://localhost:27017/"

EXPOSE 5000

CMD ["python3", "api.py"]
```

Figura 21 - Dipendenze dell'applicazione nel file requirements.txt

Nella Figura [21] è riportata la struttura del Dockerfile.

• *FROM python:latest*: indica che l'immagine base da utilizzare è l'ultima versione dell'immagine ufficiale di Python. Tale immagine include sia l'interprete Python che le librerie.

• *ADD . /app*: copia i file della directory del progetto in una nuova chiamata /app.

• *WORKDIR /app*: imposta /app come directory principale. Quindi, tutte le operazioni successive saranno eseguite in questa directory.

• *COPY requirements.txt .:* copia il file requirements.txt nella directory /app .

• *RUN pip3 install -r requirements.txt:* esegue il comando pip3 install -r requirements.txt nel container Docker. Così facendo vengono istallate le dipendenze Python elencate nel file requirements.txt.

• *ENV MONGO_URI="mongodb://mongodb:27017/":* imposta una variabile di ambiente nel container Docker chiamata MONGO_URI. Questa variabile può essere utilizzata dal codice Python per connettersi a MongoDB.

• *EXPOSE 5000*: indica che il container Docker sarà in ascolto sulla porta 5000. Questa è la porta su cui il servizio web Flask sarà disponibile.

• *CMD ["python3", "api.py"]:* è l'istruzione che viene eseguita quando il container Docker viene avviato. In questo caso, avvia lo script Python “api.py” utilizzando la versione tre dell'interprete Python.

In sintesi, Docker file crea un ambiente Docker in cui c’è Python e le dipendenze installate, copia i file di codice sorgente nel container, imposta una variabile di ambiente per la connessione a MongoDB e avvia lo script Python quando il container viene eseguito. Per containerizzare il database e creare un file che potesse gestire entrambi i container è stato costruito il Docker-compose file.

```yaml
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "5000:5000"
    depends_on:
      - mongodb

  mongodb:
    image: mongo:latest
    environment:
      MONGO_URL: mongodb://localhost:27017/
    ports:
      - "27017:27017"
    volumes:
      - mongodb-data:/data/db

volumes:
  mongodb-data:
    driver: local
```

Figura 22 - Docker-compose

Nella Figura [22] viene mostrato il codice del docker-compose. La prima riga specifica la versione di Docker Compose utilizzata, a seguire vengono elencati i servizi che compongono l'applicazione: "app", ovvero il server, e il database "mongodb". Per entrambi i servizi vengono specificate alcune indicazioni.

Per il servizio "app":

• il percorso (context) in cui si trova il Dockerfile per costruire l'immagine del container

• la porta su cui il container verrà eseguito a partire dalla porta su cui è in ascolto l'host

• le dipendenze dal servizio "mongodb"

La dipendenza fa in modo che il servizio app verrà avviato solo dopo che il servizio "mongodb" è stato avviato.

Per il servizio "mongodb":

• l'immagine Docker da utilizzare per il servizio "mongodb" è l'ultima versione dell'immagine ufficiale di MongoDB

• una variabile d'ambiente dove viene salvato l'URL del database a cui collegarsi

• la porta su cui il container verrà eseguito a partire dalla porta su cui è in ascolto l'host

• il volume dove vengono conservati i dati in modo da ritrovarli anche dopo il riavvio dei container.

Infine, viene definito il volume a cui il servizio "mongodb" fa riferimento. Il volume viene chiamato "mongodb-data" ed utilizza l'archiviazione locale.

3.4 Testing

La fase di testing ha visto protagonista due diversi casi:

TEST 1: Caso in cui l'utente vuole vedere l'andamento dei consumi in un giorno del mese in cui la batteria non è stata ancora esaurita.

TEST 2: Caso in cui l'utente vuole vedere l'andamento dei consumi in un giorno del mese in cui la batteria è stata esaurita.

La scelta dei casi è volta a mostrare la diversa visualizzazione grafica della batteria e del file di fatturazione generato dalla pressione sul bottone. Nella Figura[23] è riportato la stampa creata nel codice su Thonny per ogni richiesta post delle schedine per la verifica del corretto invio dei dati.

```
Sending data: OrderedDict({'name': 'User1', 'day': '2024-07-03', 'data': [{'value': 0.02935, 'hour': '09:45'}]})
Status: 200
Content: {
  "message": "Data insert successfully",
  "success": true
}
```

Figura 23 - Output stampa di verifica dopo richiesta POST

3.4.1 Test 1

Ci troviamo nel caso in cui l'utente voglia visualizzare l'andamento del consumo il giorno 3 giugno 2024. Ciò che appare sulla dashboard è riportato nella Figura [24]. Come già spiegato nei capitoli precedenti, sul grafico possiamo vedere l'andamento dei consumi di entrambi gli utenti che vengono distinti da due colori diversi. Accanto al grafico visualizziamo la somma del consumo giornaliero e quella del consumo mensile. Inoltre, la batteria riportata sulla destra mostra il suo livello di 'carica' sia dalla stringa sottostante sia dal colore blu che la riempe in proporzione. In basso la stringa 'Energia da pagare:' riporta la quantità di energia consumata al seguito dell'esaurimento della batteria, in questo caso 0.0. Il bottone 'FATTURA', quando premuto, genera un file denominato 'FATTURA_MESE.txt', Figuta 25.

Figura 24 - Test 1, Dashboard

FATTURA
Mese: giugno
Data: 03/06/2024
Totale energia consumata: 0.69 kWh
Totale energia da pagare: 0.00 kWh

Totale energia consumata da User1: 0.32 kWh
Totale energia consumata da User2: 0.37 kWh

RIPARTIZIONE EQUA DEL PAGAMENTO:
User1: 0.00 kWh
User2: 0.00 kWh

Figura 25 - Test 1, file FATTURA_GIUGNO.txt

3.4.2 Test 2

Per il secondo test vengono mostrate solo le figure in quanto il processo segue lo stesso ragionamento descritto nel paragrafo precedente.

Figura 26 - Test2, Dashboard

FATTURA
Mese: luglio
Data: 02/07/2024
Totale energia consumata: 2.95 kWh
Totale energia da pagare: 1.95 kWh

Totale energia consumata da User1: 1.15 kWh
Totale energia consumata da User2: 1.80 kWh

RIPARTIZIONE EQUA DEL PAGAMENTO:
User1: 0.76 kWh
User2: 1.19 kWh

Figura 27 - Test 2, file FATTURA_LUGLIO.txt

4. Conclusioni e prospettive future

Il progetto in questione, che aveva l'obiettivo di gestire i consumi energetici degli utenti, ha raggiunto risultati significativi. Durante lo sviluppo e la realizzazione del prototipo siamo riusciti a raggiungere la maggior parte degli scopi prefissati.

Il fulcro del lavoro era quello di sviluppare un sistema capace di monitorare il consumo energetico di ogni utente in maniera efficiente, offrendo a tutti possibilità di visualizzare i propri dati di consumo in qualunque momento. Questo processo ha permesso oltretutto di sensibilizzare gli utenti e di aumentare la loro consapevolezza riguardo al proprio utilizzo energetico, favorendo magari dei comportamenti sostenibili.

Il prossimo obiettivo è riuscire a incrementare l'utilizzo dell'energia rinnovabile e a diminuire l'utilizzo della dipendenza dalla rete elettrica tradizionale. Questa soluzione porterebbe a notevoli vantaggi economici e alla riduzione delle bollette energetiche.

Nonostante il risultato ottenuto sia ottimale, c'è ancora molto lavoro da fare e miglioramenti da attuare. Uno dei prossimi passi è l'ottimizzazione dell'uso dell'energia passando attraverso l'intelligenza artificiale, parte del progetto ancora in sviluppo. Le aspettative sono promettenti e il risultato sarà quello di avere dei veri e propri consigli da parte del prototipo affinché gli utenti siano consapevoli di quale sia il momento migliore per 'consumare', cioè il momento in cui l'energia rinnovabile è disponibile. L'intelligenza artificiale offre grandi vantaggi nella gestione energetica: potrebbe riuscire a prevedere le condizioni meteorologiche e quindi ottimizzare l'uso dell'energia rinnovabile, potrebbe contribuire alla manutenzione degli impianti, suggerendo quando è il momento per effettuare interventi di manutenzione.

In una realtà più vicina a quella del prototipo realizzato, l'AI potrebbe essere usata all'interno di comunità come i condomini per facilitare la condivisione dell'energia rinnovabile tra i residenti. Potrebbe essere possibile prevedere la domanda di energia e quindi capire quando l'energia prodotto la supera, evitando sprechi.

L'energia in eccesso potrebbe essere poi redistribuita agli utenti che ne hanno più bisogno, in questo modo sarà possibile massimizzare l'utilizzo delle risorse rinnovabili .

Inoltre, punto chiave del prossimo sviluppo, l'intelligenza artificiale potrebbe cambiare il flusso energetico in base alle esigenze e preferenze degli abitanti. Per esempio, un utente potrebbe preferire utilizzare l'energia rinnovabile in orari specifici e potrebbe ricevere suggerimenti personalizzati su come farlo in modo efficiente.

In conclusione, il progetto realizzato può essere un’ottima base da cui partire per integrare l'intelligenza artificiale. Possiamo considerare questo punto di partenza come un trampolino di lancio verso un utilizzo efficiente, intelligente e sostenibile dell’energia, offrendo agli utenti strumenti avanzati per gestire e capire i propri consumi.

BIBLIOGRAFIA E SITOGRAFIA

[1] FLASK. [Online]. Available: https://it.wikipedia.org/wiki/Flask_(informatica).

[2] MongoDB. [Online]. Available: https://www.mongodb.com/.

[3] Docker. [Online]. Available: https://www.docker.com/.

[4] API. [Online]. Available: https://it.wikipedia.org/wiki/Application_programming_interface.

[5] Rest. [Online]. Available: https://it.wikipedia.org/wiki/Representational_state_transfer.

[6] [Online]. Available: https://raw.githubusercontent.com/Codecademy/articles/0b631b51723fbb3cc652ef5f009082aa71916e63/images/rest_api.svg.

[7] HTTP. [Online]. Available: https://it.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[8] React. [Online]. Available: https://it.legacy.reactjs.org/.

[9] Fetch. [Online]. Available: https://it.javascript.info/fetch.

[10] V. Code. [Online]. Available: https://code.visualstudio.com/.

[11] GitHub. [Online]. Available: https://github.com/.

[12] [Online]. Available: https://git-scm.com/.

[13] Thonny. [Online]. Available: https://thonny.org/.

[14] [Online]. Available: https://it.wikipedia.org/wiki/Caso_d%27uso_(informatica).

[15] [Online]. Available: https://www.productheroes.it/cosa-sono-user-story/.

[16] [Online]. Available: https://www.3djake.com/elegoo/uno-r3-super-starter-kit.

[17] [Online]. Available: https://pypi.org/project/requests/.

[18] [Online]. Available: https://pypi.org/project/ujson/.

[19] [Online]. Available: https://docs.python.org/3/library/time.html.

[20] [Online]. Available: https://raw.githubusercontent.com/Codecademy/articles/0b631b51723fbb3cc652ef5f009082aa71916e63/images/rest_api.svg.

[21] [Online]. Available: https://en.wikipedia.org/wiki/Sequence_diagram.

INDICE DELLE FIGURE

RINGRAZIAMENTI

Grazie.