ISTITUTO TECNICO DEL SETTORE TECNOLOGICO “ENEA MATTEI”

Via Tirano n.53 - 23100 SONDRIO

Esame di Stato 2020/2021

Della Matera Lorenzo – 5E informatica

LEDLAMPS

Titolo elaborato

Realizzazione di un prototipo di lampada creata con strisce led multicolori, programmabili con schede Arduino o simili e controllabili attraverso una applicazione per dispositivi mobili e/o una applicazione web.

SOMMARIO

1. PREMESSA

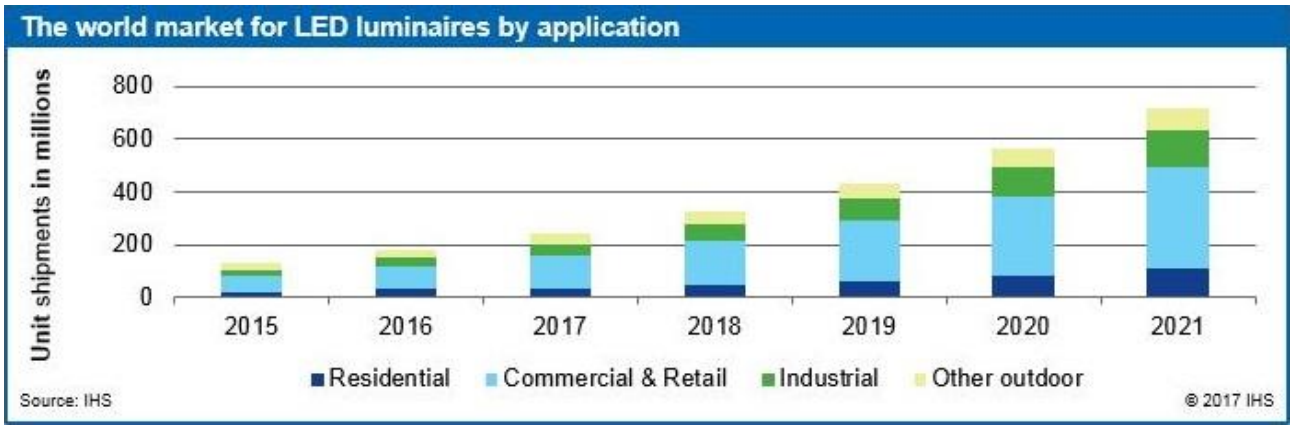
Lo sviluppo della tecnologia ha rivoluzionato la vita di tutti giorni, non solo nel settore terziario, ma anche nella quotidianità. In particolare, oggi si parla di **Internet of Things** o ancora di Internet delle Cose, per definire gli oggetti intelligenti (i cosiddetti "smart objects") che ci circondano nella vita di tutti i giorni, e non stiamo parlando soltanto di computer, smartphone e tablet. L'Internet of Things nasce proprio qui: dall'idea di portare nel mondo digitale gli oggetti della nostra esperienza quotidiana.

In questi anni le tecnologie IoT si sono moltiplicate e sviluppate, così come si sono profondamente evoluti i numerosi ambiti applicativi: casa intelligente, smart building, smart metering, smart factory, auto intelligenti, smart city, e via a seguire con smart environnment, smart agricolture, smart logistics, smart lifecycle, smart retail e smart health. Tutti ambiti resi possibili dall'interconnessione degli oggetti intelligenti.

Nell'ambito dell'IoT, la **domotica** è attualmente uno dei settori maggiormente in espansione, dove vengono concentrati buona parte degli investimenti. La "smart home" propone soluzioni per la gestione in automatico e/o da remoto degli impianti e degli oggetti connessi dell'abitazione, con il fine di ridurre i consumi energetici e migliorare il comfort, la sicurezza dell'abitazione e delle persone al suo interno, oltre a consentire un notevole incremento delle prestazioni e delle possibilità offerte dai diversi impianti presenti nell'abitazione.

In particolare, per quanto riguarda l'illuminazione, le **lampade a Led** hanno sostituito in gran parte le vecchie luci alogene per il loro alto risparmio energetico e per l'alta sostenibilità nei confronti dell'ambiente. Oltre al led normale, oggigiorno si sta diffondendo anche il **Led RGB**.
Il Led è una tecnologia che oggi viene usata per produrre qualsiasi tipo di luce grazie alla sua sostenibilità e alla sua durata. I Led tradizionali sono quelli che possono emettere un solo tipo di luce, di solito bianca e vengono impiegati moltissimo nella produzione di luci per ambienti. La tecnologia LED, che sta prendendo piede nel mercato, sfrutta una serie di diodi montati su strisce che emettono luce consumando pochissima corrente. In questo caso, i Led RGB sono nient'altro che tre tipi di Led montati sulla striscia: Led rosso, verde e blu. Miscelando i colori con le diverse intensità e sovrapponendo quindi un colore sull'altro, si può avere una luce caratterizzata da diverse tonalità che illuminano l'ambiente in modo elegante. Rispetto alla luce bianca, il Led RGB permette di dare un tocco in più all'ambiente ed è possibile cambiare la luce in modo che rispecchi i propri stati d'animo o per dare uno stile in più. Inoltre, la loro struttura molto sottile e leggera le rende facilmente fissabili su ogni supporto.

Negli ultimi dieci anni, il mercato dell'illuminazione si è evoluto verso l'eco sostenibilità, investendo sempre di più nella tecnologia Led.



Rispetto alla tecnologia tradizionale a incandescenza, i Led consumano decisamente meno elettricità e sono anche molto più versatili e resistenti. La durata rispetto a quella tradizionale è molto più elevata, e stiamo parlando di 30.000 e 50.000 ore. Per quanto riguarda il rendimento ed i consumi, sono entrambi migliori nei Led, siccome non necessitano di tanta elettricità per funzionare e riescono ad illuminare anche meglio. Anche se il loro prezzo d'acquisto è più elevato delle altre lampadine sul mercato, è anche vero che, con il tempo, il risparmio è assicurato, sia dal punto di vista dell'impatto ambientale, che economico. L'altro svantaggio è il piccolo raggio di illuminazione, ma per questo le industrie hanno pensato di costruire delle strisce a Led.

Torna all'indice

2. INTRODUZIONE

All'interno del settore sempre più in sviluppo della domotica, ho realizzato un prototipo di lampade utilizzando delle strisce LED multicolore. Le lampade sono installate su un supporto verticale e possono essere controllate via Wi-Fi, previa registrazione, sia da una **applicazione** per dispositivi **Android** dedicata, sia da una **pagina web** intuitiva. Il progetto comprende due lampade LED e un dispositivo gateway con microfono integrato che ne gestisce il controllo.

Questo tipo di lampade è adatto a personalizzare l'illuminazione della propria abitazione in modo creativo, riducendo sprechi di energia elettrica grazie alla sostenibilità di un prodotto LED. Il vantaggio di questo prodotto è che può essere controllato anche a distanza: non è necessario essere a casa per accendere, spegnere o modificare il colore delle proprie lampade, ma basta avere installata sul proprio smartphone l'applicazione dedicata oppure avere una connessione internet per accedere alla pagina web che le controlla. Questa possibilità implica però che sia presente una rete Wi-Fi domestica che permetta alle lampade di raggiungere Internet. Tuttavia, il prodotto è in grado di funzionare anche senza rete internet, dato che l'illuminazione può essere controllata localmente.

Data la versatilità delle strisce LED, il prodotto può essere utilizzare in vari ambiti oltre alla semplice illuminazione delle abitazioni private. Ad esempio si presta al mondo dello spettacolo per illuminare teatri e palchi in occasione di concerti, esibizioni e altri eventi. Oppure può essere usato in centri benessere, considerando i benefici della **cromoterapia** a livello psicologico.

Le funzionalità implementate sono numerose e vanno dall'illuminazione colorata fino a giochi di luce reattivi al suono. Di seguito tutte le modalità:

- **Accensione**, **spegnimento** e regolazione **luminosità**:
- Colorazione personalizzata:
  - **Custom color**: tutta la lampada è accesa di un solo colore;
  - **Single color fade**: la lampada si accende e spegne sfumando lentamente il colore;
  - **Double color fade**: la lampada sfuma lentamente da un colore all'altro;
  - **Double color**: metà lampada di un colore, metà dell'altro colore;
  - **Fix colors** (almeno 2 lampade): una lampata di un colore, una di un altro;
  - **Swapping colors**: l'illuminazione della lampada si alterna tra due colori;
- Giochi di luce semplici:
  - **Chill fade**: la lampada sfuma lentamente tra il rosa e l'azzurro;
  - **Color wipe**: illuminazione in modo ciclico di rosso, verde e blu;
  - **Color flash**: illuminazione alternata di rosso, verde e blu;
  - **Rainbow cycle**: illuminazione ciclica di tutti i colori dell'arcobaleno;
  - **Rainbow fade**: la lampada sfuma lentamente tutti i colori dell'arcobaleno;
  - **Rainbow chase**: ogni led si accende e spegne alternativamente dei colori dell'arcobaleno;
  - **Strobe**: accensione e spegnimento rapido e simultaneo alternando i colori dell'arcobaleno;
  - **Fire**: animazione dei led che simula una fiamma;
  - **Bouncing balls**: animazione dei led che simula delle palle rimbalzanti;
  - **Fill random**: animazione discendente in cui i led della lampada si illuminano uno ad uno di colori casuali;
  - **Twinkle**: i led si accendono in una posizione casuale di un colore random;

    - **Sparkle**: i led si accendono e spengono singolarmente in modo rapido;
- Giochi di luce reattivi al suono:
    - **Sound reactive**: la lampada si illumina in base al volume dell'audio acquisito dal microfono;
    - **Sound colors**: quando il microfono acquisisce un volume alto il colore della lampada cambia;
    - **Strobe shot**: la lampada si accende e spegne solo quando il volume acquisito dal microfono supera una certa soglia;
    - **Strobe fade**: la lampada sfuma tutti i colori lentamente, ma quando il volume acquisito dal microfono supera una certa soglia aumenta la velocità.
- Modalità random: mantiene una modalità casuale per 10 secondi e poi la cambia.

Le funzionalità aggiuntive del prodotto sono le seguenti:

- La possibilità di **automatizzare** i cambi di modalità o i cambi di colore in maniera completamente personale. Si può impostare per esempio di mantenere un colore per una durata prestabilita e poi passare ad un'altra modalità. Oppure si può impostare un cambio di colore ogni *n* secondi. Un esempio di sequenza potrebbe essere:
  1. colore rosso (10sec)
  2. colore blu (20sec)
  3. modalità stroboscopio (1min)
  4. modalità arcobaleno (1min)
  N. [...]

  Questa opportunità potrebbe essere molto utile utilizzando il prodotto in luoghi come teatri o palchi: l'illuminazione sarà automatizzata in base alle esigenze delle esibizioni, senza doverla regolare manualmente.

- La possibilità di controllare il prodotto con l'**assistente Google**, migliorandone la comodità e di conseguenza la qualità soprattutto a livello domestico. Basterà dire "Hey Google, metti le lampade rosse", per modificare il colore delle lampade. Oppure "Hey Google, imposta la luminosità bassa" per regolare la luminosità delle lampade.

3. PRESENTAZIONE TECNICA

Per la realizzazione del progetto sono state utilizzate varie tecnologie.
Alla base del sistema c'è un Database Management System (**DBMS**) che raccoglie tutti i dati delle lampade e dei loro relativi utenti. Un DMBS è proprio un sistema software in grado di gestire grandi collezioni di dati integrate, condivise e persistenti, assicurando affidabilità e privatezza. Il vantaggio principale di questo approccio è che i dati delle lampade vengono memorizzati in un unico posto in modo condiviso e diversi utenti possono accedervi attraverso diverse applicazioni anche senza dover interpellare direttamente i propri dispositivi. Inizialmente, il sistema memorizzava lo stato delle lampade in file di testo, che ogni lampada leggeva e aggiornava. Tuttavia, ampliando l'utilizzo dei dispositivi a più utenti e attraverso diverse applicazioni, è sorta la necessità di memorizzare le informazioni in un supporto condiviso, che si occupasse di gestire sia l'aspetto intensionale che l'aspetto estensionale, compresi i vincoli di integrità.

Le lampade sono comandate da delle schede Wi-Fi programmate in C/C++, attraverso l'IDE Arduino. Le lampade e il dispositivo master comunicano all'interno di una Wireless LAN, ovvero una rete informatica in cui le comunicazioni avvengono attraverso un canale senza fili. La WLAN è composta da un Access Point (il dispositivo master) e un certo numero di Station (le lampade). Un AP è un dispositivo che contiene interfacce wireless MAC e PHY conformi allo standard IEEE 802.11, che consente l'accesso a un sistema di distribuzione per le stazioni associate: in questo caso, il dispositivo master, permette alla rete privata delle lampade di raggiungere Internet, collegandosi alla rete domestica. L'autenticazione viene effettuata con il protocollo WPA2 (Wireless Protected Access), derivato dal WEP (Wired Equivalent Privacy) e ampliato utilizzando il più sicuro algoritmo di crittografia AES. La comunicazione tra le lampade e il dispositivo master avviene attraverso il UDP, un protocollo a livello trasporto non orientato alla connessione e molto veloce.

Inizialmente, si era ipotizzato che l'utente accedesse a una pagina web che modificasse il contenuto del database per scegliere la modalità delle proprie lampade. Esse, a loro volta, avrebbero fatto periodicamente delle richieste HTTP per ottenere le informazioni dal database. Tuttavia, questa soluzione è stata subito scartata: dato che il loop delle lampade che ne gestisce l'illuminazione si ripete molto velocemente non sarebbe stato opportuno fare delle richieste a una velocità troppo elevata, considerando anche possibili ritardi di trasmissione dovuti all'elevato carico. Per questo si è scelto di appoggiarsi ad un server Java che facesse da intermediario tra utenti e lampade.
Le lampade comunicano con il server Java attraverso **TCP**, un protocollo a livello trasporto orientato alla connessione, e inviano i messaggi utilizzando i socket. I socket sono canali virtuali identificati dalla coppia IP/PORT mittente e IP/PORT destinatario utilizzati per le comunicazioni nel modello client-server. Entrambi i linguaggi Java e C/C++ forniscono le API necessarie alla loro implementazione.

L'ultimo componente che appartiene al sistema è il server web PHP, che si occupa di raccogliere le richieste dell'utente e le invia al server Java. **PHP** è un linguaggio di scripting lato server, che genera pagine HTML dinamiche in base ai dati letti dal database. È stato utilizzato questo linguaggio perché possiede le librerie necessarie alla connessione al database MySQL e quelle che permettono di effettuare semplici richieste HTTP in maniera efficiente. Per gestire invece le azioni degli utenti sulle pagine web è stato utilizzato il linguaggio JavaScript, che, al contrario, è un linguaggio di scripting lato client. Permette per esempio di rilevare i click dei pulsanti per impostare una determinata modalità, oppure di gestire i form HTML attraverso cui si scelgono i colori delle lampade.

Oltre che attraverso le librerie PHP, l'accesso al database MySQL è implementato anche attraverso la libreria Java **JDBC** (Java DataBase Connectivity). L'API è sviluppata dalla SUN Microsystem e implementa un'interfaccia che si interpone tra il programma **Java** e il database. La connessione avviene tramite un apposito driver chiamato Connector/J, che si interfaccia al database. Per aprire la connessione bisogna passare l'indirizzo del database, indicandone anche il nome. Nel mio caso il database si trova sulla stessa macchina che ospita il programma Java, quindi si utilizzerà l'indirizzo localhost. Successivamente, attraverso apposite classi Java, vengono definire le query e gestiti i risultati.

Infine, dato che le lampade possono essere controllate, oltre che delle pagine web fornite dal server PHP, anche da un'applicazione **Android**, è necessario definire i protocolli di comunicazione tra quest'ultima e il server. Viene utilizzato HTTPS per garantire la sicurezza necessaria ed essere sicuri che sono gli utenti effettivamente proprietari delle lampade possano controllarle. L'applicazione è stata realizzata utilizzando Android Studio, che fornisce le librerie e gli strumenti necessari per programmare in maniera efficiente.

N.B. Tutti gli scambi di informazioni sono effettuati codificando i dati in formato **JSON**, dato che è un formato molto versatile e flessibile. Inoltre tutti i linguaggi utilizzati forniscono le API per interpretarlo in maniera efficiente.

Per utilizzare i comandi vocali attraverso l'assistente Google, è stato utilizzato il servizio **IFTTT** (IF This Than That). Permette di creare alcune applet, ossia semplici catene di condizioni che, quando verificate, generano un'azione automatica. Si possono integrare varie applicazioni, tra cui Gmail, Facebook e anche l'assistente Google. Per mio progetto, ho implementato tre applet che permettono di modificare il colore delle lampade, regolare la luminosità e impostare la modalità. Il funzionamento è molto semplice: si scrive la frase che l'assistente Google dovrà rilevare e successivamente si imposta l'URL a cui verrà effettuata una richiesta HTTP. Per esempio se si prouncia la frase "imposta il colore delle luci *rosso*", viene fatta la seguente richiesta: https://ledlampsweb.it/control/voice.php?command=rosso.

4. PROGETTAZIONE DATABASE

4.1. ANALISI DEI REQUISITI

L'intero sistema presuppone l'opportunità per vari utenti di poter controllare le proprie lampade personali, ovunque e in qualunque momento, nonché la possibilità di visualizzare il loro stato anche se non si è a casa. Inoltre, per poter automatizzare le modalità delle luci è necessario un supporto che ne memorizzi le informazioni.

A questo proposito il sistema si appoggia su un database che memorizza e gestisce i dati e le informazioni degli utenti e delle lampade.

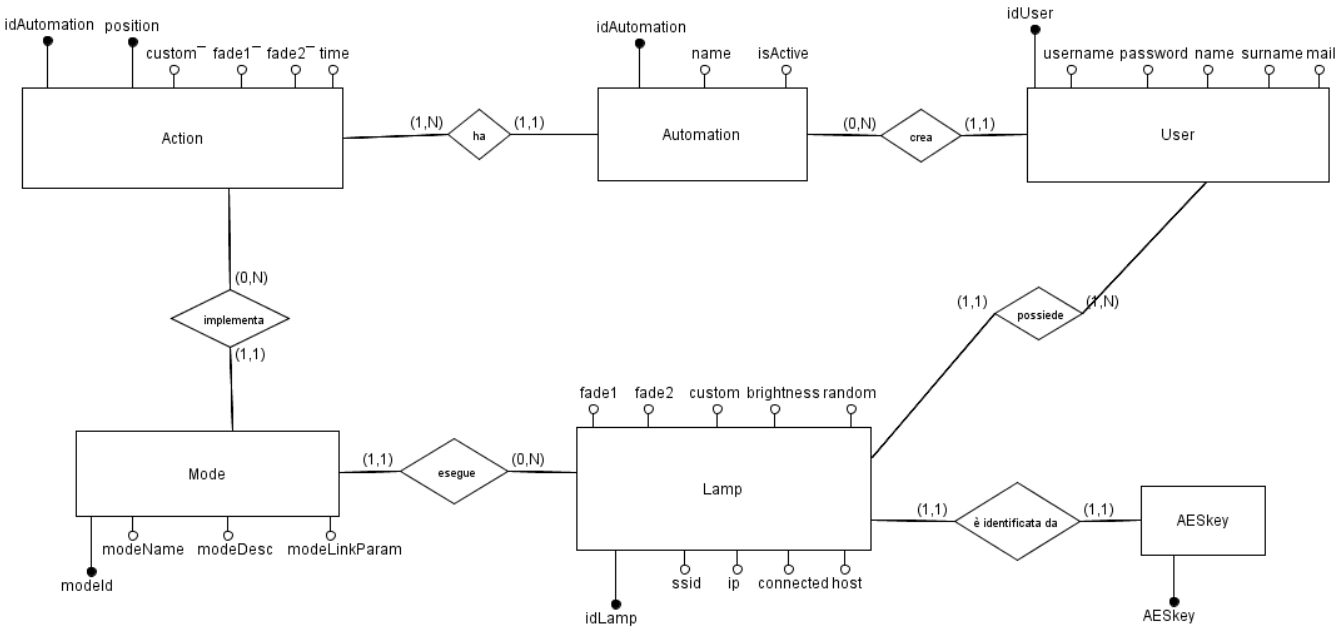
In particolare è necessario memorizzare tutte le informazioni degli utenti che utilizzano le lampade e salvare l'identificativo della lampada che possiedono. Si presuppone che un utente possegga una sola lampada ma che una lampada possa essere utilizzata da più utenti.

Per ogni lampada devono essere memorizzate alcune informazioni che ne definiscono l'attuale stato (la modalità in corso, il colore di illuminazione ecc.). Ogni lampada deve avere un identificativo univoco associato ad una chiave di crittografia univoca che verrà usata per criptare le comunicazioni con il server in maniera simmetrica. Questo concetto verrà definito in seguito.

Infine, è necessario memorizzare le automazioni che gli utenti definiscono per le proprie lampade. Le automazioni, in pratica, sono delle liste di "azioni" (cambiamenti di modalità) che la lampada deve eseguire per un certo periodo di tempo.

4.2. PROGETTAZIONE CONCETTUALE

Partendo dai requisiti sopra descritti è stato realizzato il seguente **schema ER**.

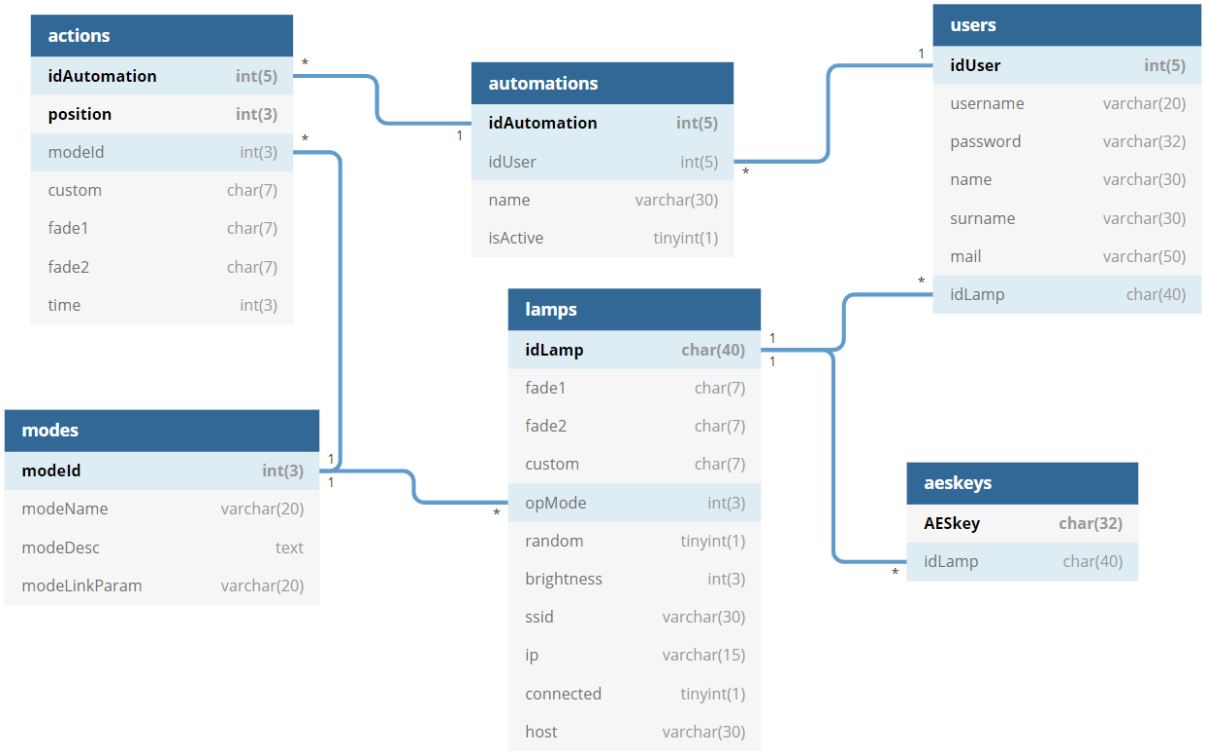


Torna all'indice

L'entità **Lamp** contiene tutte le informazioni sullo stato attuale della lampada. Si è scelto di tenere le chiavi di crittografia in un'entità separata, **AESkey**, per motivi di sicurezza: in questo modo è possibile attribuire dei vincoli di accesso e impedire a chi non ne ha diritto di accedervi. L'associazione è 1:1 perché ogni lampada ha una sola chiave di crittografia identificativa. L'idLamp è una stringa che deriva dalla chiave di crittografia e si ottiene applicando su di essa l'algoritmo di hashing SHA1 (in questo modo, conoscendo l'id della lampada non si può risalire alla sua chiave).
N.B. un'istanza dell'entità Lamp, in effetti, non identifica una singola lampada in sé, ma identifica il dispositivo gateway che la controlla, infatti può essere che uno di essi controlli più lampade che sono connesse alla sua rete, ma che saranno sempre nello stesso stato.
L'entità **Mode** contiene le informazioni delle modalità disponibili ed è collegata all'entità Lamp con un'associazione N:1 (N lampade eseguono 1 modalità), infatti una lampada può eseguire solo una modalità alla volta, ma più lampade possono eseguire la stessa modalità.
L'entità **User** rappresenta i singoli utenti che vogliono controllare le proprie lampade. L'associazione è N:1 (N utenti controllano 1 lampada) perché più utenti possono controllare la stessa la lampada, ma un utente è per forza identificato da una sola lampada. Quindi se un utente volesse controllare una lampada diversa dalla sua dovrebbe creare un altro profilo. La password degli utenti è memorizzata nel database codificata con l'algoritmo di hashing MD5, per motivi di sicurezza.
L'entità **Automation** rappresenta le singole automazioni definite dagli utenti. L'associazione è 1:N parziale (1 utente crea 0-N automazioni) perché non è obbligatorio che un utente definisca un'automazione per la propria lampada, ma può definirne anche più di una. Dato che l'utente che l'ha definita è associato a una e una sola lampada, l'automazione è associata a una e una sola lampada.
Le azioni che compongono le automazioni sono definite nell'entità **Action**, collegata con un'associazione 1:N completa (1 associazione ha N azioni) perché un'automazione può eseguire più azioni ma deve eseguirne almeno una. Questa entità è collegata all'entità **Mode** con un'associazione N:1 (N azioni implementano 1 modalità) perché ogni azioni può implementare una sola modalità.

4.3. PROGETTAZIONE LOGICA

Partendo dal precedente schema ER è stato realizzato il seguente **schema logico**.



Torna all'indice

Modes (modeId, modeName, modeDesc, modeLinkParam)
Lamps (idLamp, fade1, fade2, custom, opMode, random, brightenss, ssid, ip, connected, host)
Aeskeys (AESkey, idLamp)
Users (idUser, username, password, name, surname, mail, idLamp)
Automations (idAutomaton, idUser, name, isActive)
Actions (idAutomation, position, modeId, custom¯, fade1¯, fade2¯, time)

Modes:
modeId (int), chiave primaria
modeName (varchar), nome della modalità
modeDesc (text), breve descrizione della modalità e del suo funzionamento
modeLinkParam (varchar), il valore del parametro che va passato durante una richiesta HTTP per impostare la modalità

Lamps:
idLamp (char), chiave primaria (SHA1 della chiave di crittografia)
fade1 (char), primo colore usato nelle modalità "doppio colore"
fade2 (char), secondo colore usato nelle modalità "doppio colore"
custom (char), colore usato nelle modalità "colore personalizzato"
opMode (int), chiave esterna che identifica la modalità attualmente in esecuzione
random (boolean), flag che segnala se la modalità "random" è attiva
brightness (int), valore da 0 a 255 che definisce la luminosità della lampada
ssid (varchar), SSID della rete a cui la lampada è connessa (vuoto se disconnessa)
ip (vachar), IP privato assegnato alla lampada quando connessa alla rete domestica (vuoto se disconnessa)
connected (boolean), flag che segnala se la lampada è connessa o meno al server Java
host (varchar), IP del server Java a cui la lampada è connessa (utilizzato in fase di testing)

N.B. colori memorizzati nel formato #RRGGBB.

AESkeys:
AESkey (char), chiave primaria, la chiave di crittografia da 32 byte per l'algoritmo AES
idLamp (char), chiave esterna che collega la chiave di crittografia alla relativa lampada

Users:
idUser (int), chiave primaria
username (varchar), username dell'utente
password (varchar), password dell'utente (MD5)
name (varchar), nome dell'utente
surname (varchar), cognome dell'utente
mail (varchar), indirizzo mail dell'utente
idLamp (char), chiave esterna che identifica la lampada associata all'utente

Automations:
idAutomation (int), chiave primaria
idUser (int), chiave esterna che identifica l'utente che ha creato l'automazione
name (varchar), nome dell'automazione
isActive (boolean), flag che segnala se la modalità è in esecuzione o meno

Actions:
idAutomation (int), chiave primaria, l'automazione che contiene quell'azione
position (int), chiave primaria, la posizione di ordinamento dell'azione nell'elenco dell'automazione
modeId (int), chiave esterna che identifica la modalità da implementare in quell'azione

fade1 (char), primo colore usato nelle modalità “doppio colore” (opzionale: NULL se la modalità non implica l’utilizzo di questo colore)
fade2 (char), secondo colore usato nelle modalità “doppio colore” (opzionale: NULL se la modalità non implica l’utilizzo di questo colore)
custom (char), colore usato nelle modalità “colore personalizzato” (opzionale: NULL se la modalità non implica l’utilizzo di questo colore)
time (int), la durata in secondi dell’azione (per quanto tempo la lampada deve eseguirla prima di passare alla successiva)

4.4. PROGETTAZIONE FISICA

Partendo dal precedente schema logico sono state implementate le seguenti tabelle in **linguaggio SQL**.

```
CREATE TABLE `modes` (
  `modeId` int(3) NOT NULL,
  `modeName` varchar(20) NOT NULL,
  `modeDesc` text NOT NULL,
  `modeLinkParam` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`modeId`)
)
CREATE TABLE `lamps` (
  `idLamp` char(40) NOT NULL,
  `fade1` char(7) DEFAULT '#ff0000',
  `fade2` char(7) DEFAULT '#00ff00',
  `custom` char(7) DEFAULT '#0000ff',
  `opMode` int(3) DEFAULT '1',
  `random` tinyint(1) DEFAULT '0',
  `brightness` int(3) DEFAULT '255',
  `ssid` varchar(30) DEFAULT '',
  `ip` varchar(15) DEFAULT '',
  `connected` tinyint(1) DEFAULT '0',
  `host` varchar(30) DEFAULT '80.211.47.247',
  PRIMARY KEY (`idLamp`),
  KEY `opMode` (`opMode`),
  CONSTRAINT `lamps_ibfk_1` FOREIGN KEY (`opMode`) REFERENCES `modes` (`modeId`)
)
CREATE TABLE `aeskeys` (
  `AESkey` char(32) NOT NULL,
  `idLamp` char(40) NOT NULL,
  PRIMARY KEY (`AESkey`),
  KEY `idLamp` (`idLamp`),
  CONSTRAINT `aeskeys_ibfk_1` FOREIGN KEY (`idLamp`) REFERENCES `lamps` (`idLamp`)
)
CREATE TABLE `users` (
  `idUser` int(5) NOT NULL AUTO_INCREMENT,
  `username` varchar(20) NOT NULL,
  `password` varchar(32) NOT NULL,
  `name` varchar(30) DEFAULT NULL,
  `surname` varchar(30) DEFAULT NULL,
  `mail` varchar(50) NOT NULL,
  `idLamp` char(40) DEFAULT NULL,
  PRIMARY KEY (`idUser`),
  KEY `idLamp` (`idLamp`),
  CONSTRAINT `users_ibfk_1` FOREIGN KEY (`idLamp`) REFERENCES `lamps` (`idLamp`)
)
CREATE TABLE `automations` (
  `idAutomation` int(5) NOT NULL AUTO_INCREMENT,
  `idUser` int(5) NOT NULL,
  `name` varchar(30) NOT NULL,
  `isActive` tinyint(1) NOT NULL DEFAULT '0',
  PRIMARY KEY (`idAutomation`),
  KEY `idUser` (`idUser`),
  CONSTRAINT `automations_ibfk_1` FOREIGN KEY (`idUser`) REFERENCES `users` (`idUser`)
)

CREATE TABLE `actions` (
  `idAutomation` int(5) NOT NULL,
```

```
  `modeId` int(3) NOT NULL,
  `position` int(3) NOT NULL,
  `custom` char(7) DEFAULT NULL,
  `fade1` char(7) DEFAULT NULL,
  `fade2` char(7) DEFAULT NULL,
  `time` int(3) NOT NULL,
  PRIMARY KEY (`idAutomation`,`position`),
  KEY `modeId` (`modeId`),
  CONSTRAINT `actions_ibfk_1` FOREIGN KEY (`idAutomation`) REFERENCES `automations` (`idAutomation`) ON
DELETE CASCADE,
  CONSTRAINT `actions_ibfk_2` FOREIGN KEY (`modeId`) REFERENCES `modes` (`modeId`)
)
```

N.B. la chiave esterna della tabella Actions che identifica la relativa automazione è impostata ON DELETE CASCADE, dato che se si elimina una automazione vanno eliminate anche tutte le relative azioni.

Per soddisfare i vincoli di integrità necessari sono stati implementati alcuni trigger e alcune procedure.

4.4.1. TRIGGERS

- Quando si vuole creare una nuova lampada è necessario che venga generata una chiave di crittografia univoca che la identifichi e da questa verrà poi generato l'idLamp, con l'algoritmo SHA1. È necessario ricordare che nella tabella AESkeys è presente una chiave esterna che individua l'identificativo della lampada a cui si riferisce, ma, come sappiamo, questo id deriva dalla chiave di crittografia stessa. Per questo, prima di inserire nella tabella AESkeys un nuovo record, è necessario creare una lampada che abbia come chiave primaria l'hash della chiave di crittografia generata, in modo che sia rispettato il vincolo di integrità referenziale. Quindi, la creazione di chiave di crittografia e lampada deve avvenire ipoteticamente in modo simultaneo. Per fare questo è stato implementato il seguente trigger.

```
CREATE TRIGGER newlamp
BEFORE INSERT
ON aeskeys
FOR EACH ROW
INSERT INTO lamps (idLamp) VALUES(NEW.idlamp)
```

Praticamente, per creare una nuova lampada bisogna prima creare la relativa chiave di crittografia, attraverso la seguente query SQL. Successivamente, per rispettare il vincolo di integrità referenziale sopra citato, verrà eseguito il trigger che inserirà un nuovo record nella tabella Lamps che abbia come chiave primaria l'identificativo corretto.

```
INSERT INTO aeskeys(AESkey, idLamp) VALUES (REPLACE(UUID(), '-', ''), SHA1(UNHEX(AESkey)));
```

N.B. la funzione MySQL UUID() restituisce un valore lungo 128 bit che rappresenta un identificatore univoco universale (specificato da RFC 4122) in formato esadecimale
aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee.
La funzione è progettata in modo tale da generare un numero unico a livello globale in base allo spazio e al tempo. Se chiamiamo due UUID in successione, otterremo due valori diversi, e anche questi sono stati eseguiti su due dispositivi separati che non sono collegati. In questo caso, il risultato viene utilizzato come chiave di crittografia AES (dato che la lunghezza è la stessa, 128 bit), rimuovendo i caratteri «–».

- Quando si vuole aggiungere una nuova azione ad un'automazione già esistente è necessario che la sua posizione nell'elenco venga settata al valore successivo rispetto alle azioni già presenti. Per esempio se un'automazione contiene 3 azioni, quando se ne vuole aggiungere un'altra il suo attributo position dovrà essere impostato a 4. Per realizzare questo non basta aggiungere al campo la clausola AUTO_INCREMENT, dato che questa incrementa il valore in base a tutti i record della tabella, in pratica imposterebbe l'attributo al valore successivo rispetto al massimo presente. Di conseguenza è stato implementato il seguente trigger.

```
DELIMITER $$
CREATE TRIGGER incPosition
BEFORE INSERT
ON actions
FOR EACH ROW
BEGIN
  SET @pos = (SELECT MAX(position) FROM actions WHERE actions.idAutomation = NEW.idAutomation);
  IF (@pos IS NOT NULL) THEN
            SET NEW.position = @pos+1;
  END IF;
END $$
DELIMITER ;
```

- Il dominio del campo brightness nella tabella Lamp comprende valori da 0 a 255, è quindi necessario definire un controllo attraverso un trigger.

```
DELIMITER $$
CREATE TRIGGER checkBrightness
BEFORE UPDATE
ON lamps
FOR EACH ROW
BEGIN
IF (NEW.brightness > 255) THEN
          SET NEW.brightness = 255;
ELSEIF (NEW.brightness < 0) THEN
          SET NEW.brightness = 0;
END IF;
END $$
DELIMITER ;
```

4.4.2. PROCEDURES

- Creare una nuova automazione implica che venga creata anche una prima azione collegata. È inoltre vietato, per uno stesso utente, definire automazioni con lo stesso nome. Per fare questo è stata implementata la seguente procedura, al cui interno è definita una **transazione**. Ciò significa che se una delle query all'interno fallisce (ad esempio se è già presente una automazione con lo stesso nome, oppure se non sono stati definiti i campi della prima azione da inserire), la nuova automazione non verrà creata.

```
DELIMITER $$
CREATE PROCEDURE create_automation(IN idUserNew INT, nameNew VARCHAR(30), modeIdNew INT, positionNew
INT, customNew CHAR(7), fade1New CHAR(7), fade2New CHAR(7), timeNew INT)
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    RESIGNAL;
  END;

  START TRANSACTION;

  INSERT INTO automations (idUser, name) VALUES (idUserNew, nameNew);

  SET @id = (SELECT idAutomation FROM automations WHERE idUser = idUserNew AND name = nameNew);

  INSERT INTO actions VALUES (@id, modeIdNew, positionNew, customNew, fade1New, fade2New, timeNew);

  COMMIT;
END $$
DELIMITER ;
```

- Quando si elimina una procedura si vuole ritornare all'utente un messaggio che segnala se l'automazione eliminata era in esecuzione ed è stata creata la seguente procedura.

```
DELIMITER $$
CREATE PROCEDURE delete_automation(IN idAutomationDel INT)
```

```
BEGIN
    SET @isActive = (SELECT isActive FROM automations WHERE idAutomation = idAutomationDel);

    DELETE FROM automations WHERE idAutomation = idAutomationDel;

    SELECT @isActive result;
END $$
DELIMITER ;
```

- È stata sviluppata anche la possibilità di eliminare o muovere in un'altra posizione singole azioni di una automazione. Per farlo è necessario aggiornare le posizioni di tutte le altre azioni presenti e sono state implementate le seguenti procedure.

```
DELIMITER $$
CREATE PROCEDURE delete_action(IN idAutomationDel INT, positionDel INT)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
    END;

    START TRANSACTION;

    DELETE FROM actions
    WHERE idAutomation = idAutomationDel AND position = positionDel;

    UPDATE actions SET position = position - 1
    WHERE position > positionDel AND idAutomation = idAutomationDel;

    COMMIT;
END $$
DELIMITER ;

DELIMITER $$
CREATE PROCEDURE move_action(IN idAutomationMove INT, fromPos INT, toPos INT)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
    END;

    START TRANSACTION;

    UPDATE actions SET position = 0
    WHERE idAutomation = idAutomationMove AND position = fromPos;

    IF (fromPos < toPos) THEN
                UPDATE actions SET position = position - 1
                WHERE idAutomation = idAutomationMove AND position > fromPos AND position <= toPos;
    ELSE
                UPDATE actions SET position = position + 1
                WHERE idAutomation = idAutomationMove AND position < fromPos AND position >= toPos
        ORDER BY position DESC;
    END IF;

    UPDATE actions SET position = toPos
    WHERE idAutomation = idAutomationMove AND position = 0;

    COMMIT;
END $$
DELIMITER ;
```

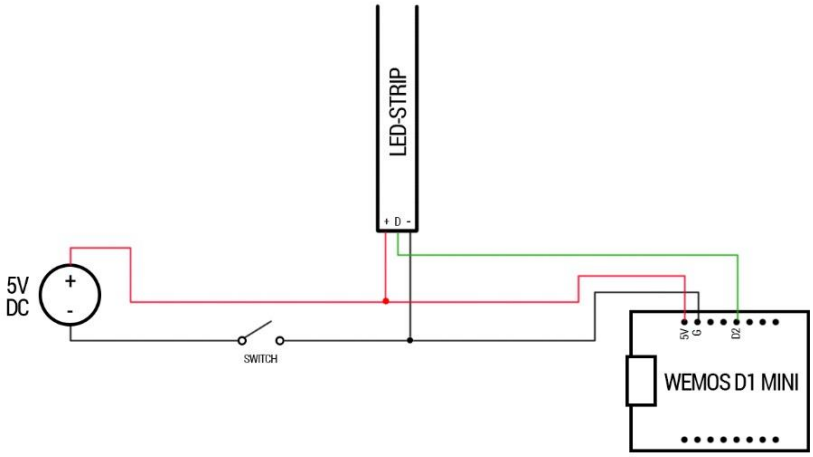
5. STRUTTURA LAMPADE

Il prototipo comprende due componenti principali:

- Una o più lampade slave che si illuminano secondo i comandi ricevuti
- Un dispositivo master che si occupa di controllare le lampade e ha la funzione di gateway

5.1. LAMP (slave)

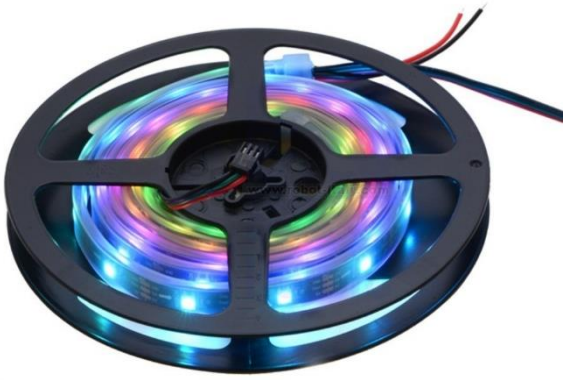
Si compone di una striscia led installata su un supporto verticale, nella cui base è presente il seguente **schema elettrico**. Un alimentatore DC 5V fornisce la corrente necessaria al circuito e un interruttore permette di gestire l'alimentazione dello stesso.



- LED STRIP

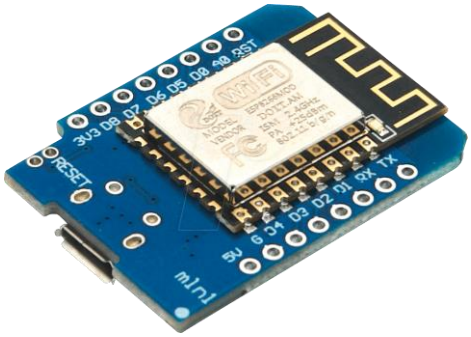
È stata utilizzata una **Adafruit NeoPixel Digital RGB LED Strip** 144leds/m, una particolare striscia led che oltre ai due normali pin per l'alimentazione ne possiede un terzo attraverso cui vengono impartiti i comandi per gestire l'illuminazione. Questa particolare striscia led è "individually addressable", ciò significa che ogni singolo led della striscia può essere controllato individualmente, ciascun pixel può avere il colore e la luminosità che si desidera: questa possibilità permette di realizzare vari effetti luminosi oltre all'illuminazione completa di tutta la lampada di uno stesso colore.

SPECIFICHE TECNICHE
Lunghezza della striscia: 1 m
Grado di protezione: IP30
Assorbimento massimo: 80 mA per LED (tutti i LED alla massima luminosità)
Requisito di alimentazione 5 V DC - nessuna protezione dalla polarità
Temperatura colore bianco neutro: ~ 4000-4500 K.

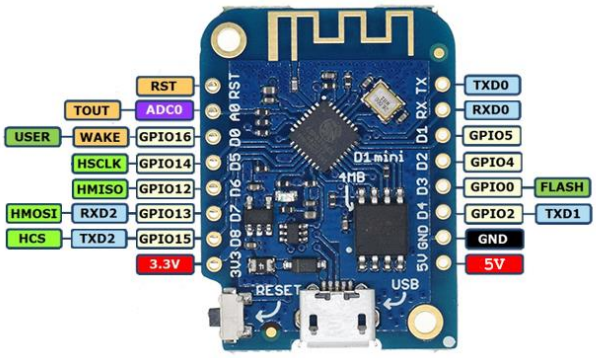


- SCHEDA WIFI

È stata utilizzata una **WEMOS D1 MINI ESP-8266**, una scheda elettronica WiFi programmabile con l'IDE Arduino, attraverso l'uso di opportune librerie. Attraverso il chip ESP8266 con Wi-Fi integrato (autenticazione WEP, WPA/WPA2 o reti aperte), la scheda supporta completamente il protocollo TCP/IP e le funzionalità da microcontrollore. La scheda, ricevendo le istruzioni via Wi-Fi attraverso il protocollo UDP, si occupa di comandare i singoli led della striscia grazie alla libreria Arduino *<FastLED.h>*.

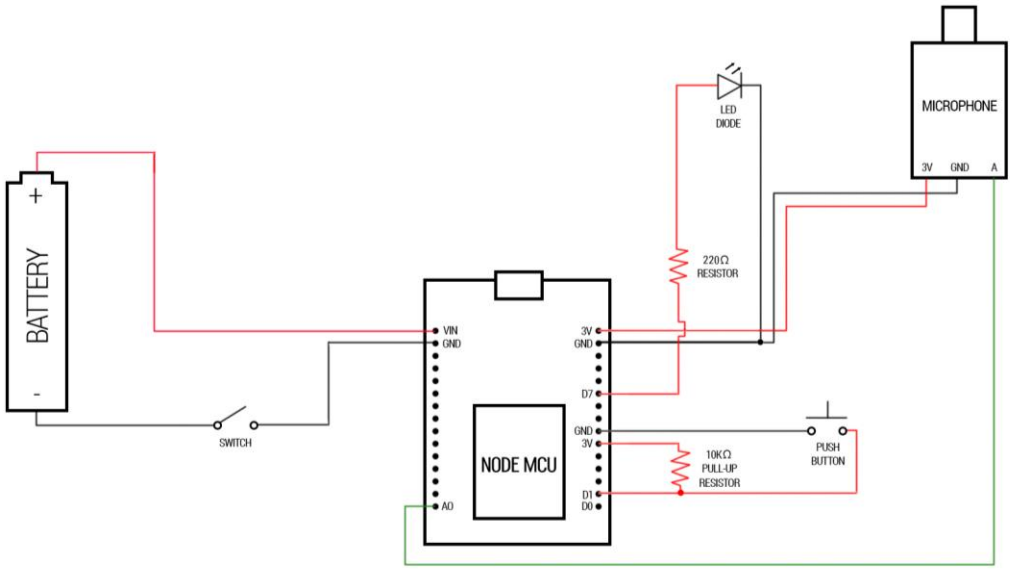


SPECIFICHE TECNICHE
Operating Voltage: 3.3V
Digital I/O Pins: 11
Analog Input Pins: 1 (3.2V Max)
Clock Speed: 80/160MHz
Flash: 4M Bytes
Dimensioni: 34.2 x 25.6 mm
Peso: 3g



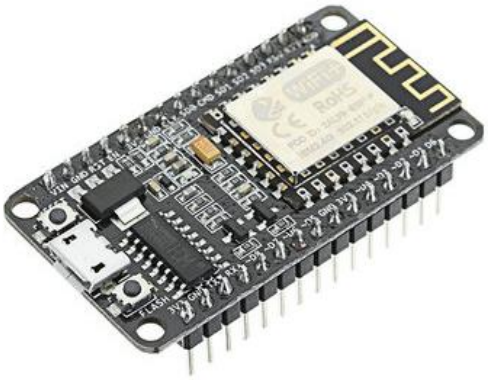
5.2. CONTROLLER (master)

Si compone di un contenitore plastico al cui interno è presente il seguente **schema elettrico**. Un set di 3 batterie alcaline AA 1.5 V fornisce la corrente necessaria al circuito e un interruttore ne gestisce l'alimentazione: in questo modo il microfono può essere posizionato ovunque nel raggio di alcuni metri dato che non necessita di una presa di corrente. Il diodo blu da 3V svolge la funzione di led segnaletico (successo: 2 flash, errore: 3 flash). Il microfono permette di gestire i giochi luminosi reattivi al suono. Il pulsante permette di cambiare la modalità delle luci in modo manuale.



Torna all'indice

• SCHEDA WIFI

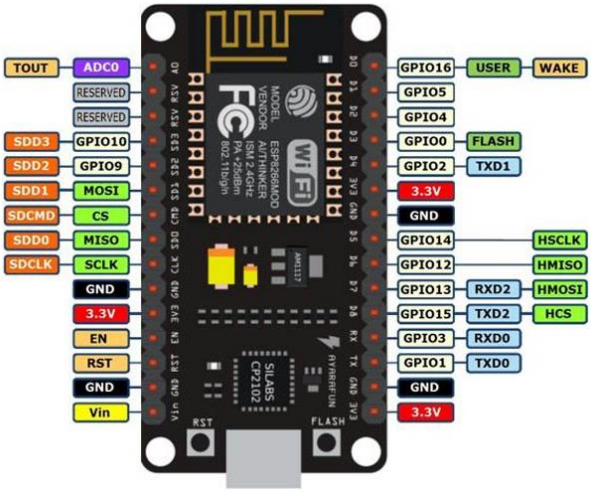


NodeMCU è una scheda di sviluppo basata su Lua open source appositamente progettata per applicazioni basate su IoT. Include il chip ESP8266 e include le stesse funzioni della Wemos D1 mini. Anche questa scheda è programmabile con l'IDE Arduino attraverso l'utilizzo di alcune librerie. Nel progetto, scheda svolge due funzioni principali:

- **Access Point**: le schede delle lampade si connettono alla rete Wi-Fi generata dalla NodeMCU (192.1868.4.0) e comunicano attraverso il protocollo UDP.
- **Gateway**: la scheda permette alla rete privata delle lampade di essere raggiunta da altre reti, per esempio quella domestica, in questo modo è possibile controllare le lampade anche senza essere connessi alla loro rete privata.

Il suo funzionamento verrà descritto in seguito.

<u>SPECIFICHE TECNICHE</u>
Wi-Fi Protocols 802.11 b/g/n (HT20)
Frequency Range 2.4 GHz ~ 2.5 Antenna PCB Trace
CPU Tensilica L106 32-bit processor
Operating Voltage 2.5 V ~ 3.6 V
Operating Current Average value: 80 mA
Operating Temperature Range –40 °C ~ 125 °C
Wi-Fi Mode Station/SoftAP/SoftAP+Station
Security WPA/WPA2
Encryption WEP/TKIP/AES



• RESISTENZA DI PULL-UP

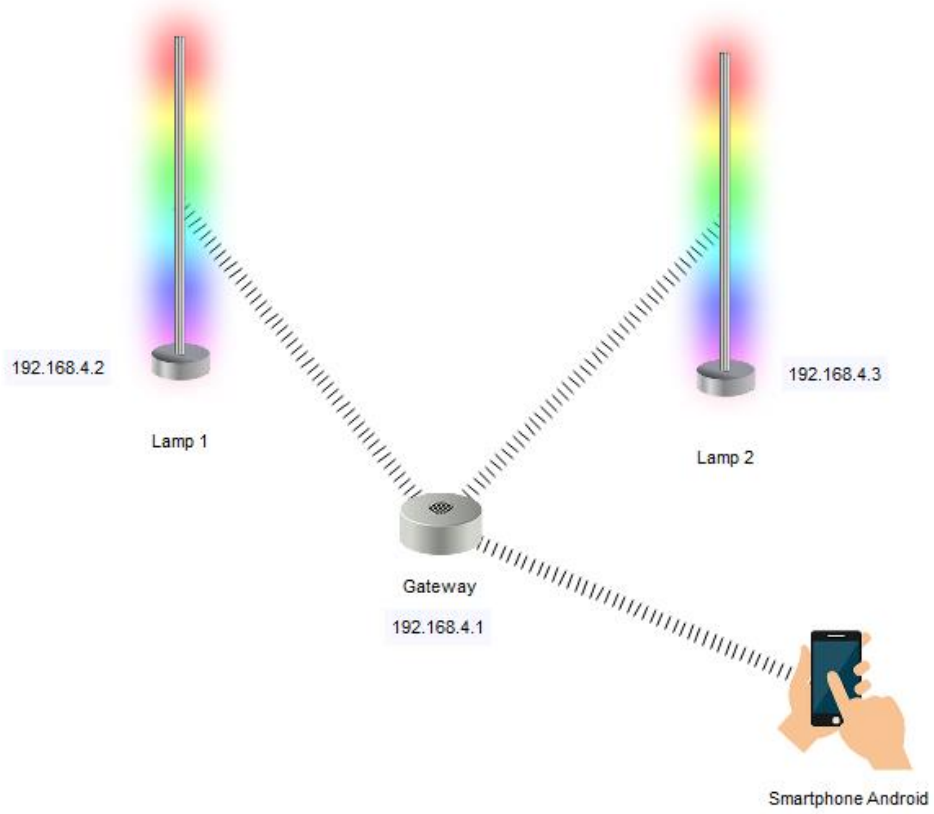
I pin di ingresso dei circuiti digitali presentano normalmente un'impedenza abbastanza alta, e questo significa che basta una corrente minima per poter cambiarne lo stato. Ma significa anche che sono soggetti a captare qualsiasi disturbo elettromagnetico o elettrostatico e quindi a commutare le loro uscite in modo del tutto imprevisto ed incontrollato. Da qui deriva la necessità di impostare il suddetto pin in uno stato perfettamente conosciuto, che non possa portare a stati di incertezza sulle letture effettuate: si inserisce una resistenza tra il pin di ingresso e la tensione di riferimento, per fare in modo che all'ingresso sia sempre presente un segnale certo. In questo caso, con la resistenza da 1KΩ, il potenziale del pin D1, è impostato alla tensione di riferimento 3V. Quando il pulsante viene chiuso l'ingresso viene portato a massa. Il codice Arduino, leggerà periodicamente il valore digitale di ingresso del pin D1: normalmente sarà a 1, quando passa a 0 vuol dire che il pulsante è stato premuto ed è il segnale che l'utente vuole cambiare modalità di illuminazione delle lampade. Di seguito la porzione di codice che gestisce la pressione del pulsante.

```
void buttonCheck() {
  int but = digitalRead(BUTTON_PIN);
  if (but == 0) {
    // pulsante premuto
  }
  else if (but == 1) {
    // pulsante rilasciato
  }
}
```

Torna all'indice

6. RETE DELLE LAMPADE

Come già anticipato, la scheda master NodeMCU svolge la funzione di access point creando una rete Wi-Fi privata a cui si connettono le lampade. Nel codice delle due schede WEMOS D1 MINI sono già presenti le credenziali per connettersi alla rete: SSID = "LedLamps", PASSWORD = [idLamp]. L'utente, per poter controllare le lampade, si connette alla rete Wi-Fi usando le stesse credenziali.
N.B. L'idLamp è la chiave primaria memorizzata nella tabella Lamps del database ed è a conoscenza solo di chi possiede le lampade, si ipotizza che venga riferito all'utente che acquista il prodotto attraverso una mail.



6.1. COMUNICAZIONE MASTER – SLAVE

Il gateway e le lampade comunicano attraverso il protocollo UDP. Questo protocollo di livello trasporto è particolarmente adatto perché la comunicazione deve avvenire molto rapidamente e UDP è molto veloce dato che non effettua il controllo di flusso (ordinamento datagrammi e controllo errori). Inoltre il protocollo UDP non è orientato alla connessione, cioè non vi è una vera e propria fase di handshake tra client e server, al contrario i pacchetti sono inviati rapidamente senza ulteriori controlli. Infatti l'affidabilità dei dati non è particolarmente necessaria perché una lieve perdita di informazione è contemplata, mentre non lo è il ritardo dei messaggi.
I messaggi che il gateway manda alle lampade sono broadcast, questo vuol dire che tutte le lampade ricevono gli stessi messaggi. Di seguito è riportata la struttura del messaggio:

```
struct led_command {
  uint8_t opmode;
  uint32_t data;
  uint32_t data2;
};
```

Il pacchetto contiene 3 informazioni principali: il codice della modalità che le lampade devono eseguire e due campi che contengono informazioni aggiuntive in base alla modalità. Per esempio, se la modalità implica la colorazione della lampada in uno o due colori, questi saranno contenuti nei due campi (codificati come un numero da 9 cifre dove ogni terzetto rappresenta il valore di rosso, verde e blu da 0 a 255, nel formato RRRGGGBBB). Oppure, se la modalità corrisponde ad un gioco di luce reattivo al suono, uno dei campi conterrà un valore da 0 a 644 che rappresenta il volume captato dal microfono. Questo è uno dei motivi per cui è necessaria una comunicazione rapida tra master e slave: se i pacchetti ritardassero il gioco di luce non sarebbe effettivamente reattivo al suono.
Il protocollo è realizzato grazie alla libreria Arduino *<WiFiUDP.h>,* implementata dal chip ESP8266. La comunicazione si svolge nelle seguenti fasi:

1. L'access point genera la rete Wi-Fi e assegna a sé stesso l'indirizzo 192.168.4.1.

```
WiFi.softAP(AP_SSID, AP_PASS);
```

2. Il master apre un socket UDP sul port 7171 e si mette in ascolto di connessioni.

```
void setup() {
  UDP.begin(7171);
  waitForConnections();
}
void waitForConnections() {
  while(true) {
      readHeartBeat();
      if (checkHeartBeats()) {
        return;
      }
      delay(checkDelay);
      resetHeartBeats();
  }
}
```

3. Le lampade si connettono alla rete Wi-Fi, aprono un socket sul port 7001, si mettono in ascolto di messaggi e iniziano a inviare messaggi di "heartbeat" (letteralmente "battito del cuore") al master sul port 7171 a intervalli regolari: significa che sono connesse e sono in attesa di istruzioni.

```
void setup() {
  connectToWifi();
  UDP.begin(7001);
  sendHeartBeat();
}
void sendHeartBeat() {
  struct heartbeat_message hbm;
  hbm.client_id = LAMP_ID;
  hbm.chk = 77777;
  Serial.println("Sending heartbeat");
  IPAddress ip(192, 168, 4, 1);
  UDP.beginPacket(ip, 7171);
  int ret = UDP.write((char*)&hbm, sizeof(hbm));
  printf("Returned: %d, also sizeof hbm: %d \n", ret, sizeof(hbm));
  UDP.endPacket();
  lastHeartBeatSent = millis();
}
```

4. Il master aspetta che tutte le lampade siano connesse (nel codice è presente una costante che indica qual è il numero di lampade che devono connettersi).

```
void readHeartBeat() {
  struct heartbeat_message hbm;
  while(true) {
    int packetSize = UDP.parsePacket();
    if (!packetSize) {
      break;
    }
    UDP.read((char *)&hbm, sizeof(struct heartbeat_message));
```

```
    if (hbm.client_id > number_of_clients) {
      Serial.println("Error: invalid client_id received");
      continue;
    }
    heartbeats[hbm.client_id - 1] = true;
  }
}

bool checkHeartBeats() {
  for (int i = 0; i < number_of_clients; i++) {
    if (!heartbeats[i]) {
      return false;
    }
  }
  resetHeartBeats();
  return true;
}
```

5. Quando tutte le lampade sono connesse il master inizia il loop infinito in cui invia i messaggi con le istruzioni per la modalità da eseguire a intervalli regolari (≈ 5 millisecondi).

```
void sendLedData(uint32_t data, uint8_t op_mode, uint32_t data2) {
  struct led_command send_data;
  send_data.opmode = op_mode;
  send_data.data = data;
  send_data.data2 = data2;
  for (int i = 0; i < number_of_clients; i++)
  {
    IPAddress ip(192,168,4,2 + i);
    UDP.beginPacket(ip, 7001);
    UDP.write((char*)&send_data,sizeof(struct led_command));
    UDP.endPacket();
  }
}
```

6. Ogni volta che una lampada riceve il messaggio con la modalità da eseguire manda in esecuzione la relativa funzione. Ogni funzione ha una durata molto breve perché la lampada deve subito rimettersi in ascolto di nuovi messaggi: se il prossimo messaggio contiene la stessa modalità allora riprede da dove era rimasta (funzioni con variabili static), altrimenti cambia modalità immediatamente.

```
void loop() {
  int packetSize = UDP.parsePacket();
  if (packetSize)
  {
    UDP.read((char *)&cmd, sizeof(struct led_command));
    lastReceived = millis();
  }
  int opMode = cmd.opmode;
  int data1 = cmd.data;
  int data2 = cmd.data2;
}
```

7. Nel frattempo il master rimane costantemente in ascolto di messaggi “**heartbeat**”: se capita che una lampada si disconnette dalla rete Wi-Fi il master se ne accorge e aspetta che la lampada in questione si riconnetti prima di continuare a inviare i messaggi.

```
void loop() {
  if (millis() - lastChecked > checkDelay) {
    if (!checkHeartBeats()) {
      waitForConnections();
    }
    lastChecked = millis();
  }
}
```

8. Di conseguenza ogni lampada deve continuare a inviare messaggi "**heartbeat**" a intervalli regolari per segnalare che è connessa è sta ricevendo le istruzioni correttamente, in caso contrario si interrompe e si riconnette.

```
void loop() {
  if (millis() - lastHeartBeatSent > heartBeatInterval) {
    sendHeartBeat();
  }
  if (millis() - lastReceived >= 5000) {
    connectToWifi();
  }
}
```

6.2. COMUNICAZIONE USER – MASTER

Il dispositivo master, dotato di scheda NodeMCU, è in grado di aprire un socket sul port 80 e accettare richieste HTTP: in questo modo verrà creato un server web all'IP 192.168.4.1 che può essere contattato da un qualsiasi utente connesso alla rete Wi-Fi.
Grazie alla libreria *<ESP8266WebServer.h>* è possibile definire un oggetto *ESP8266WebServer* che gestisce le richieste HTTP. Per esempio, può rispondere con semplici pagine HTML/JavaScript oppure può delegare una funzione specifica per analizzare la richiesta e compiere una determinata azione.
La scheda NodeMCU possiede 4MB di memoria flash, dove sono memorizzati alcuni file con le pagine HTML da inoltrare ai client che si collegano al server web. È possibile gestire questa memoria attraverso il sistema **SPIFFS** (Serial Peripheral Interface Flash File System), un semplice file system per questo tipo di microcontrollori. In questo modo si evita di dover scrivere le pagine html direttamente nel codice Arduino.
Le pagine web sono molto semplici: un elenco di opportuni elementi *<a href=[link]>* che inviano le richieste alla scheda NodeMCU per eseguire la modalità desiderata.
Per particolari richieste sono state definite funzioni specifiche all'interno del codice che le trattano, mentre per le richieste delle pagine HTML memorizzate viene inoltrato il relativo file.

```
ESP8266WebServer server(80);
server.begin();
server.on("/mode", []() {                       // link per impostare la modalità
  handleMode(server.arg(0));                    // funzione apposita che gestisce la richiesta
});
//...
server.onNotFound([]() {
  if (!handleFileRead(server.uri()))            // ricerca del file
    server.send(404, "text/plain", "404: Not Found");
});

bool handleFileRead(String path) {              // manda il file richiesto (se esiste)
  Serial.println("handleFileRead: " + path);
  if (path.endsWith("/")) path += "index.html";
  String contentType = getContentType(path);
  if (SPIFFS.exists(path)) {
    File file = SPIFFS.open(path, "r");
    size_t sent = server.streamFile(file, contentType);    // funzione di invio dati
    file.close();
    return true;
  }
  Serial.println("\tFile Not Found");
  return false;
}
```

Il dispositivo quindi, all'interno del suo ciclo infinito, dovrà ogni volta controllare se c'è qualche client che vuole interpellare il server web e, in tal caso, gestire la sua richiesta.

```
server.handleClient();
```

In questo modo, un qualsiasi utente connesso alla rete Wi-Fi delle lampade è in grado di controllarle facendo una richiesta http del tipo http://192.168.4.1/[parametri].

7. SERVER JAVA

7.1. COMUNICAZIONE MASTER – SERVER

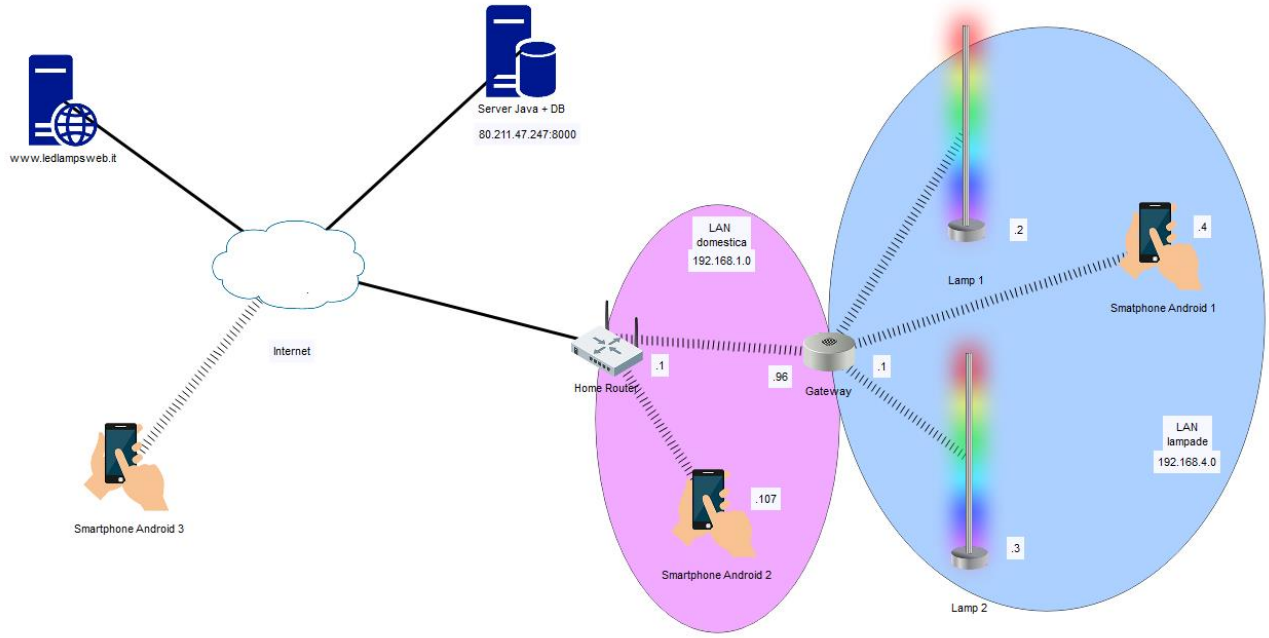
Con il sistema implementata fin ora, le lampade sono controllabili connettendosi alla loro rete privata, perché solo in questo modo è possibile accedere al server web di indirizzo 192.168.4.1 implementato dalla scheda NodeMCU. Tuttavia, ciò può risultare scomodo all'utente: ogni volta che vuole cambiare un'impostazione deve connettere il proprio smartphone alla rete locale, con una conseguente perdita di tempo. Inoltre, completata la modifica, l'utente dovrebbe disconnettersi nuovamente e riconnettersi ad un'altra rete che gli permetta di accedere a Internet per compiere operazioni di altro tipo: lo smartphone dell'utente non è dedicato solamente alla modifica delle lampade!
Questo svantaggio, può essere facilmente risolto se è presente una rete Wi-Fi domestica. La scheda NodeMCU, infatti, oltre ad avere la funzione di access point, può anche connettersi ad una LAN già esistente (modalità *station*), dato che ha due interfacce di rete. In questo modo, se l'utente si connette alla rete domestica potrà raggiungere il server web del dispositivo, connesso a sua volta alla stessa rete.
Questa soluzione però, implica comunque che l'utente si trovi presso l'abitazione e sia connesso alla rete domestica. Per poter controllare le lampade anche se ci si trova in un altro luogo è necessario avere un sistema hardware-software che consenta al dispositivo master di connettersi a Internet ed essere raggiunto tramite un indirizzo pubblico. Una semplice soluzione sarebbe configurare in modo appropriato il router domestico. Quest'ultimo, infatti, implementa automaticamente il servizio di **NAT/PAT** e permette ad un utente connesso alla LAN di raggiungere internet. Se lo si configurasse in modo da pubblicare su internet l'indirizzo privato del server web della scheda NodeMCU (NAT/PAT statico), le lampade sarebbero interpellabili da un qualsiasi luogo, connettendosi al loro indirizzo pubblico. In questo modo, il router inoltrerà i pacchetti provenienti dall'esterno e sul port 80 (HTTP) verso l'indirizzo privato del dispositivo master. Tuttavia, questa soluzione non è la migliore, dato che presenta alcuni svantaggi:

1) Il router dovrebbe essere appositamente configurato da un tecnico specializzato, dato che un normale utente che acquista questo prodotto non è in grado di farlo.
2) Implementare il NAT/PAT statico sulla porta 80 in una rete privata come quella domestica in cui non sono presenti firewall non è una buona pratica, perché va a discapito della sicurezza.

La soluzione che ho sviluppato si appoggia ad un server Java esterno, con indirizzo pubblico, a cui le lampade si connettono (raggiungendo Internet grazie alla LAN domestica) e con cui comunicano attraverso il protocollo TCP. Il server Java, dopo aver creato una connessione tramite socket con il dispositivo master, inoltra alle lampade le istruzioni per eseguire la modalità richiesta dall'utente. In particolare, l'utente invierà le sue richieste HTTP non più direttamente al server web della scheda NodeMCU, ma ad un altro **server web PHP**, che sarà raggiungibile ovunque dato che si trova su internet con un indirizzo pubblico. Successivamente, questo server web PHP si occuperà di inoltrare le richieste al server Java, che a sua volta manderà le richieste alle lampade, tramite la loro connessione socket TCP.
Il server Java implementa anche un **DBMS** dove è presente il database contenente tutti i dati delle lampade (ad es. l'ultimo stato) e le relative informazioni dei loro utenti. Grazie a questo database le applicazioni front-end saranno sempre a conoscenza dello stato delle lampade e perciò l'utente avrà la possibilità di sapere se le lampade sono accese o spente anche se non si trova a casa.
Lo schema seguente rappresenta il sistema appena descritto ed evidenzia le tre modalità con cui l'utente può controllare le proprie lampade.

- *Smartphone 1*: connesso alla rete privata delle lampade, invia le richieste http al dispositivo master sull'interfaccia 192.168.4.1;
- *Smartphone 2*: connesso alla rete domestica, invia le richieste http al dispositivo master sull'interfaccia 192.168.1.96;
- *Smartphone 3*: connesso a Internet tramite rete mobile, invia le richieste http al server web di indirizzo www.ledlampsweb.it.

N.B. Naturalmente tutto ciò è effettuato dall'applicazione Android in maniera trasparente per l'utente.



7.2. FASI DELLA COMUNICAZIONE

La comunicazione si basa sul **protocollo TCP** che è orientato alla connessione ed effettua il controllo della trasmissione dei messaggi, garantendo l'ordine di arrivo dei pacchetti e la loro integrità. In questo caso, infatti, è necessario che venga definita una connessione, ovvero un canale virtuale, attraverso cui il dispositivo master e il server Java possano comunicare. Precisamente, il server Java implementa un server multiclient ed è in grado di gestire le connessioni di diverse lampade contemporaneamente. Il server Java si basa sulla libreria *java.net* mentre il dispositivo master utilizza la libreria *<WiFiClient.h>.*

La comunicazione si svolge nelle seguenti fasi:

1. Il server Java crea un oggetto ServerSocket sul port 12345 e si mette in attesa di connessioni.

```
public void start() {
    try{
        while(true) {
            canale_ser = new ServerSocket(12345);
            Socket socket = canale_ser.accept();
            new LampClient(socket, keychain);
        }
    } catch(IOException e){ e.printStackTrace(); }
}
```

2. Il dispositivo master, all'accensione, si prova a connettere automaticamente all'ultima rete Wi-Fi conosciuta (memorizzata in un file di testo nel file system SPIFFS). Successivamente si connette al server Java all'indirizzo 80.211.47.247:12345.

```
bool connectToWifi(String ssid, String pass){
```

```
  WiFi.begin(ssid, pass);
  int i=0;
  // provo per 17 secondi a connettermi alla rete Wi-Fi
  while (WiFi.status() != WL_CONNECTED && i<17) {
    delay(1000);
    i++;
    Serial.print(".");
  }

  if(WiFi.status() == WL_CONNECTED){
    Serial.println("Connected!");
    ssid_now = ssid;
    pass_now = pass;                    // salvo le credenziali della WLAN per
    saveWifiSettings();                 // riconnettermi al prossimo avvio
    connectToServer(HOST);              // mi connetto al server Java
    return true;
  }
  else{
    Serial.print("Error, can not connect to ");
    Serial.println(ssid);
    ssid_now = "";
    pass_now = "";
    return false;
  }
}
void connectToServer(String new_host){
  if(!client.connected()){
    HOST = new_host;
    if (client.connect(HOST, PORT)){
      Serial.print("connected to the server ");
      Serial.println(HOST);
      handshake();                                  // fase di handshake
    }
    else
      Serial.println("connection failed");
    updateData();                                   // aggiorno lo stato della lampada
  }
  else{
    client.println(sha1key_enc);                    // se sono già connesso invio un messaggio di sync
  }
}
```

N.B. È possibile connettere il dispositivo al Wi-Fi e/o al server Java manualmente attraverso l'applicazione Android.

3. Alla connessione di un nuovo client, il server Java crea un oggetto Socket dedicato per gestirlo. Contemporaneamente viene fatto partire un Thread dedicato che rimane in ascolto di messaggi da parte del client, che generalmente sono informazioni riguardo lo stato attuale della lampada.

```
public LampClient(Socket socket, Keychain keychain){
    this.socket = socket;
    this.keychain = keychain;          // raccolta di chiavi in cui trovare quella relativa al client
    this.dbConnector = new DBConnector();
    try {
        socket.setSoTimeout(5000);
        System.out.print("Connected!");
        out = new PrintStream(socket.getOutputStream());
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        new ReadThread().start();
    } catch(IOException e){ e.printStackTrace(); }
}
```

4. A questo punto vi è una fase di **handshake**, in cui viene verificata l'identità del client per capire qual è la lampada che si è connessa al server (e se effettivamente è una lampada con il permesso di connettersi!). Questa fase verrà descritta in seguito in modo dettagliato.

5. Una volta che il dispositivo master e il server Java hanno stabilito la connessione possono iniziare a inviarsi messaggi. In particolare, il dispositivo master invierà al server messaggi di "**sync**" (simili a quelli di "heartbeat" inviati tra slave e master nella comunicazione UDP precedentemente descritta), che contengono l'idLamp criptato. Questo perché, durante le fasi di testing, ho notato che la libreria Arduino che implementa il client TCP non chiude automaticamente il socket quando la scheda si spegne. Perciò, è necessario che il server controlli che il client sia effettivamente connesso leggendo i suoi messaggi di "sync": se dopo un certo periodo di tempo questi messaggi non vengono più ricevuti la connessione viene interrotta.

```
if(millis() - lastSync > 7000){
  client.println(sha1key_enc);
  lastSync = millis();
}
```

6. A questo punto client e server possono comunicare:

- Ad ogni richiesta dall'utente il server Java invia il comando al dispositivo master.

```
public void send(String data) {                 // invio del server
    dbConnector.lock(AESkey.getSha1key());
    try {
        String len = data.length()+"$";  // lunghezza del messaggio originale utilizzata per decriptare
        out.print(len+AES.encrypt(data, AESkey.getKey(), AESkey.getIvParameterSpec()));
    } catch (Exception e) { e.printStackTrace(); }
}
```

- Il dispositivo master legge il messaggio, lo interpreta ed esegue il comando

```
if(client.connected()){                         // ricezione del client
  String received="";
  while(client.available()) {
    received += (char)client.read();
  }
  if(!received.equals("")){
    received = AES_decrypt(received);
    handleMessageFromServer(received);          // interpreto il messaggio
  }
}
```

- Ad ogni cambio di stato della lampada il dispositivo master invia al server Java i nuovi dati.

```
void sendDataToServer(){                        // invio del client
  if(client.connected()) {
    Serial.println("sending data to server");
    client.println(AES_encrypt(getJsonData()));
    lastSync = millis();
  }
  else
    Serial.println("not connected to server");
}

String getJsonData(){        // ritorna le informazioni sullo stato attuale codificati in JSON
  String data = "{ \"fade1\": \"" + color1 + "\"" +
                ", \"fade2\": \"" + color2 + "\"" +
                ", \"custom\": \"" + color3 + "\"" +
                ", \"opMode\": \"" + opMode + "\"" +
                ", \"random\": \"" + randomEnabled + "\"" +
                ", \"brightness\": \"" + brightness_now_int + "\"" +
                ", \"ssid\": \"" + ssid_now + "\"" +
                ", \"connected\": \"" + client.connected() + "\"" +
                ", \"ip\": \"" + WiFi.localIP().toString() + "\"" +
                ", \"host\": \"" + HOST + "\" }";
  return data;
}
```

- Il server Java riceve il messaggio, effettua i controlli necessari e lo salva nel database

```
private void handle(){                          // ricezione del server all'interno del Thread
    String frase = "";
    while(!frase.equals("#")) {
        try {
            frase=in.readLine();
            if(restart<5) { restart = 0; }      // sync
            if(frase!=null && !frase.equals(AESkey.getSha1key_enc())){
                frase = AES.decrypt(frase, AESkey.getKey(), AESkey.getIvParameterSpec());
                if(!frase.equals("#"))
                    saveJSONdata(frase);        // se ho ricevuto un aggiornamento salvo nel DB
            }
        } catch(SocketTimeoutException e) {     // se dopo un certo periodo di tempo
            restart++;                          // non ricevo più messaggi ...
            if(restart>=4)
                 break;                         // ... interrompo
         } catch(Exception e) {
              break;
         }
    }
}
```

N.B. il salvataggio avviene attraverso il driver JDBC che si occupa della connessione al database MySQL.

```
private boolean doQuery(String jsonString, String idlamp) throws Exception{
    JSONObject jobj = new JSONObject(jsonString);
    Statement stmt = connection.createStatement();

    String querySQL = "UPDATE lamps SET " +
                      "fade1 = '" + jobj.getString("fade1") + "', " +
                      "fade2 = '" + jobj.getString("fade2") + "', " +
                      "custom = '" + jobj.getString("custom") + "', " +
                      "opMode = " + jobj.getInt("opMode") + ", " +
                      "random = " + jobj.getInt("random") + ", " +
                      "brightness = " + jobj.getInt("brightness") + ", " +
                      "ssid = '" + jobj.getString("ssid") + "', " +
                      "connected = " + jobj.getInt("connected") + ", " +
                      "ip = '" + jobj.getString("ip") + "', " +
                      "host = '" + jobj.getString("host") + "' " +
                      "WHERE idlamp = '" + idlamp + "';";

    System.out.print("["+idlamp+"] ");
    System.out.println("SQLQuery: "+querySQL);
    if(stmt.executeUpdate(querySQL)!=1){
        connection.close();
        return false;
    }
    stmt.execute("COMMIT;");
    connection.close();
    return true;
}
```

7.3. WEB SERVER

Il programma Java in esecuzione, oltre a gestire la connessione con le lampade, deve anche restare in ascolto di eventuali richieste da parte di un utente. Come già anticipato, l'utente non invia le richieste HTTP direttamente al server Java, ma le inoltra ad un server web PHP che a sua volta le inoltra al server Java. Tutto ciò per motivi di sicurezza. Quindi, sul server Java sarà presente un altro socket in ascolto dei messaggi provenienti dal server web PHP su un'altra porta, diversa da quella a cui si connettono le lampade.
Dato che le richieste provenienti dal server web PHP si basano sul protocollo http, il socket è in ascolto sulla porta 8000 (simile ma non uguale alla porta 80 del HTTP). In pratica, è come se il server Java implementasse un server web in ascolto sulla porta 8000 invece che sulla porta 80 (sempre per ragioni di sicurezza).

Tutto ciò è implementato grazie alla libreria Java *com.sun.net.httpserver* che permette di definire un oggetto *HttpServer* che si occupa di gestire il protocollo HTTP.
All'arrivo di una richiesta sul port 8000, il programma controlla prima di tutto che l'IP del mittente sia quello del server web PHP (solo lui è autorizzato ad interpellarlo) e successivamente analizza i parametri della richiesta. Dopo aver identificato a quale lampada si riferisce la richiesta, il programma richiama il socket del relativo client e inoltra la richiesta sotto forma di stringa. Il dispositivo master, connesso tramite TCP, si occuperà di analizzare la stringa ricevuta e compiere la corretta operazione.
N.B. Le richieste http che vengono inviate in questa fase sono nel formato
http://80.211.47.247:8000/[idLamp]/[parametri].

```
private void handleResponse(HttpExchange httpExchange, String requestParamValue, String key)
throws IOException {
    OutputStream outputStream = httpExchange.getResponseBody();
    String htmlResponse = "";
    int code = 200;
    LampClient client = Clients.get(key);
    if(httpExchange.getRemoteAddress().getAddress().toString().equals("/80.211.73.225")){
        if(client!=null) {
            if(requestParamValue.contains("mode") || ...) {
                client.send(requestParamValue);
                htmlResponse = "success";
            }
            else if(requestParamValue.contains("automations")){
                String params = httpExchange.getRequestURI().getQuery();
                if(params.contains("start")){
                    client.stopAutomator();
                    String idAutomation = params.split("=")[2];
                    client.startAutomator(Integer.parseInt(idAutomation));
                }
                else if(params.contains("stop")){
                    client.stopAutomator();
                }
                else if(params.contains("update")){
                    String idAutomation = params.split("=")[2];
                    client.updateAutomator(Integer.parseInt(idAutomation));
                }
                htmlResponse = "success";
            }
            else if(requestParamValue.contains("restart")) {
                client.restart();
            }
        }
        else {
            code = 500;
            htmlResponse = "client not connected!";
        }
    }
    else {
        code = 403;
        htmlResponse = "403: forbidden";
    }
}
httpExchange.sendResponseHeaders(code, htmlResponse.length());
outputStream.write(htmlResponse.getBytes());
outputStream.flush();
outputStream.close();
```

7.4. AUTOMAZIONI

Come già anticipato, le lampade offrono la possibilità di automatizzare i cambiamenti di modalità. Questo è interamente implementato dal server Java e le lampade non si accorgono che stanno eseguendo un'automazione.
Infatti, il server non manda nessun messaggio al dispositivo master per informarlo che l'utente ha attivato un'automazione. Al contrario, un Thread si occupa di inviare, a intervalli definiti, il messaggio con la nuova modalità che le lampade devono eseguire. Come già detto, le automazioni sono delle liste di azioni, perciò, quando l'utente attiva una modalità attraverso una richiesta HTTP, il server Java si occupa di ottenere dal database le azioni relative. Successivamente fa partire il Thread (che dura fino a che l'utente non interrompe l'automazione o seleziona manualmente una nuova modalità) che legge ciclicamente la lista di azioni: per ognuna di esse invia al dispositivo il messaggio contenente le informazioni della stessa (modalità, eventuali colori) e in seguito effettua uno sleep di tanti secondi quanti deve durare l'azione; subito dopo legge la prossima azione e ripete la procedura.
Le lampade, a loro volta, ricevono periodicamente messaggi con le istruzioni per eseguire una nuova modalità e non sanno che l'invio di questi messaggi da parte del server avviene in modo automatico, perché non è l'utente che richiede ogni volta il cambio di modalità manualmente.

```
public void run() {
    try{
        while(true){
            for (int i=0; i<actions.size(); i++) {
                Action action = actions.get(i);
                if(action.getModeId() == 44 || action.getModeId() == 45){
lampClient.send("/set_color_hash?color="+action.getCustom()+"&colorMode="+action.getModeId());
                }
                else if(action.getModeId() >= 46 && action.getModeId() <= 49){
lampClient.send("/set_color_hash?first="+action.getFade1()+"&second="+action.getFade2()+"&colorMode="+a
ction.getModeId());
                }
                else {
                    lampClient.send("/mode?opMode="+action.getModeLinkParam());
                }
                Thread.sleep((long)action.getTime());
            }
        }
    } catch(InterruptedException e){}
}
```

7.5. CLASSI

LedLamps.java: contiene il metodo main().
JavaServer.java: gestisce le connessioni di nuovi client.
WebServer.java: gestisce le richieste http.
LampClient.java: rappresenta la connessione di ogni client.
    ReadThread.java: Thread che rimane in ascolto di messaggi dal client.
Clients.java: classe statica che contiene la lista di client connessi.
Keychain.java: contiene le chiavi di crittografia dei client.
    AESKey.java: contiene le informazioni della chiave di crittografia.
AES.java: si occupa della crittografia dei messaggi.
DBConnector.java: gestisce la connessione al database tramite JDBC.
Automator.java: Thread che si occupa delle automazioni.
Action.java: singola azione che compone una automazione.

8. SERVER WEB PHP

Come già accennato, le richieste HTTP dell'utente non vengono indirizzate direttamente al server Java per motivi di sicurezza. Al contrario viene eseguito un passaggio in più.
Il controllo delle lampade può essere effettuato attraverso l'applicazione Android oppure attraverso una pagina web. Per uniformare le due opzioni, il server web PHP si occupa di inoltrare le richieste che provengono sia dall'applicazione sia dalle pagine web. Le pagine web, naturalmente, risiedono già sul server web PHP.

Grazie al servizio di hosting offerto da www.hostingvirtuale.com, ho realizzato un server web PHP che contiene le pagine dinamiche con i dati letti dal database. Attraverso queste pagine si possono controllare le lampade anche senza aver scaricato l'applicazione Android.
All'indirizzo https://ledlampsweb.it è presente il sito web da dove è possibile accedere alla pagina di controllo. Una volta effettuato il login alla pagina https://ledlampsweb.it/control/login.php, l'utente accederà alla pagina di controllo, dove potrà vedere l'attuale stato delle proprie lampade (o se sono disconnesse, il loro ultimo stato). Le informazioni delle lampade sono ottenute facendo una semplice query **SELECT** verso il database. La connessione avveiene attraverso le funzioni native di PHP:

```php
<?php
    $host = "80.211.47.247";
    $dbUser = "dellamateral";
    $dbPwd = "D_ellamateral_21";
    $dbName = "ledlamps";
    $con = mysqli_connect($host, $dbUser, $dbPwd, $dbName);
?>
```

Il login viene effettuato completando un form HTML (la password verrà criptata con MD5); poi la pagina PHP effettuerà il controllo delle credenziali, chiedendo al database presente all'indirizzo 80.211.47.247 se esiste quell'utente: in caso positivo l'utente verrà reindirizzato alla pagina che permette di controllare la lampada associata all'account https://ledlampsweb.it/control/index.php.

La pagina principale contiene una serie di pulsanti per attivare le varie modalità e una sfumatura bianca evidenzia quella attualmente in esecuzione. Al click di uno di essi viene invocata una funzione JavaScript che, attraverso la tecnologia **AJAX** e l'oggetto JavaScript *XMLHttpRequest*, effettua una richiesta HTTP a un'altra pagina PHP sul server web.

```javascript
function requestHTTP(param){
    var url = param;
    xmlHttp = new XMLHttpRequest();
    xmlHttp.open("GET", url, true);
    xmlHttp.onreadystatechange = function() {
        if (xmlHttp.readyState == 4 && xmlHttp.status == 200) {
            var text = xmlHttp.responseText;
            if(text.includes("error"))
                alert(xmlHttp.responseText);
            else
                load(xmlHttp.responseText);
        }
    }
    xmlHttp.send();
}
```

La pagina richiamata si occupa di comunicare al server Java la modalità desiderata, inviando a sua volta una richiesta del tipo http://80.211.47.247:8000/[idLamp]/[parametri] attraverso la libreria **cURL**.

```php
<?php
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, "http://80.211.47.247:8000/".$idLamp.$url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
    $output = curl_exec($ch);
    curl_close($ch);
?>
```

Se la richiesta va a buon fine, la stessa pagina PHP legge i dati della lampada dal database e li ritorna sotto forma di oggetto **JSON**. Così, la pagina *index.php* che aveva effettuato la richiesta AJAX, può leggere i dati JSON, interpretarli e ricaricarsi con i dati aggiornati.

```javascript
function load(data) {
    var mydata = JSON.parse(data);

    var n = document.querySelectorAll(".button");
    for (i = 0; i < n.length; i++) {
        n[i].className = (n[i].id == mydata["opMode"] ? "button current" : "button default");
    }

    var y = document.getElementById("double");
    var x = y.querySelectorAll(".button");
    var i;
    for (i = 0; i < x.length; i++) {
        x[i].style.background = "linear-gradient(to right,"+mydata["fade1"]+","+mydata["fade2"]+")";
    }

    y = document.getElementById("44");
    x = document.getElementById("45");
    y.style.background = mydata["custom"];
    x.style.background = "linear-gradient(to right," + mydata["custom"] + ", white)";

    var int = mydata["brightness"];
    var hex = (+int).toString(16);
    var text;
    if(int<=80)
        text = "white";
    else
        text = "black";
    x = document.getElementById("brightness");
    x.style.background = "#"+hex+hex+hex;
    x.style.color = text;
    x.value = int;
}
```

Sempre grazie alla tecnologia AJAX, la pagina è in grado di ricaricarsi in modo automatico quando la modalità delle lampade viene cambiata da un altro utente, per esempio. Infatti, la pagina HTML esegue ogni 3 secondi la richiesta di "sync", chiedendo alla pagina *sync.php* di leggere i dati dal database e ritornarli in formato JSON.

```javascript
window.setInterval(function() {
    requestHTTP('sync.php');
}, 3000);
```

• SESSIONI

Attraverso il meccanismo delle sessioni fornito da PHP, è possibile mantenere in memoria alcuni valori mentre si naviga tra le pagine web. In questo caso è molto utile memorizzare l'id della lampada relativa all'utente loggato dato che tutte le richieste verso il server Java lo contengono. Inoltre, solo se effettivamente l'utente ha inserito le corrette credenziali può visualizzare lo stato delle proprie lampade. Per questo, la pagina *login.php*, dopo aver controllato le credenziali e aver verificato che siano corrette, si deve occupare di salvare l'idLamp relativo nell'array super globale *$_SESSION*, in modo che ogni pagina che verrà richiamata in seguito lo possa ottenere.

```php
<?php
    $q = "SELECT * FROM users WHERE username = '$username' AND password = '$password';";
    $risultato = mysqli_query($con, $q);
    $data = mysqli_fetch_array($risultato, MYSQLI_ASSOC);
    $idLamp = $data["idLamp"];
    if($idLamp!=null){
        $_SESSION["idLamp"] = $idLamp;
?>
```

Quindi, ogni pagina PHP deve come prima cosa invocare la funzione *session_start(),* per creare una nuova sessione o per ripristinarla nel caso sia stata creata in precedenza. In seguito, controlla che sia definito il valore idLamp nell'array super globale *$_SESSION*. Se questo non fosse, l'accesso alle pagine web verrà proibito.

```php
<?php
    session_start();
    if (!isset($_SESSION['idLamp'])) {
        if(!isset($_POST['idLamp'])) {
            echo "error: forbidden";
            die;
        }
        else {
            $idLamp = $_POST['idLamp'];
        }
    }
    else {
        $idLamp = $_SESSION['idLamp'];
    }
?>
```

• COMPONENTI SERVER WEB

In pratica sono stati definiti tre tipi elementi che compongono il server web:

1) Pagine HTML dinamiche che permettono di selezionare la modalità, scegliere i colori… :
   - *index.php*: selezione modalità;
   - *colorPicker.php*: selezione colore per modalità "single color";
   - *doubleColorPicler.php*: selezione colori per modalità "double color";
   - *brightnesspicker.php*: selezione luminosità;
   - *login.php*: accedere al proprio account.
2) Pagine interamente PHP che effettuano le richieste http verso il server Java:
   - *db_connect.php*: per la connessione al database;
   - *mode.php*: invio di una modalità;
   - *set_color_hash.php*: invio colore/i;
   - *set_brightness_hash.php*: invio luminosità;
   - *random.php*: attivare/disattivare la modalità random;
   - *voice.php*: utilizzo dei comandi vocali;
   - *sync.php*: richiesta dell'attuale stato della lampada al database;
   - *signup.php*: registrare un nuovo utente;
   - *automations/automator.php*: attiva/disattiva un'automazione;
   - *automations/create.php*: crea una nuova automazione;
   - *automations/delete.php*: elimina un'automazione;
   - *automations/select.php*: ottiene le informazioni delle automazioni e le converte in JSON;
   - *automations/addAction.php*: aggiunge un'azione a un'automazione esistente;
   - *automations/deleteAction.php*: elimina un'azione da un'automazione esistente;
   - *autoamtions/moveAction.php*: muove un'azione in una posizione diversa;
   - *automations/getModes.php*: ottiene le informazioni di tutte le modalità disponibili.

3) Elementi di AJAX/JavaScript per aggiornare le pagine HTML automaticamente:
   - *ajax.js*.

8.1. FASI DELLA COMUNICAZIONE

Di seguito è riportato un esempio che illustra i passi che avvengono quando un utente vuole modificare la modalità di una sua lampada collegandosi al server web.

1. L'utente accede alla pagina *login.php* e inserisce le proprie credenziali;
2. Viene reindirizzato alla pagina *index.php* dove controlla l'attuale stato delle proprie lampade;
3. Vuole selezionare la modalità "sound reactive" e preme il relativo pulsante;
4. Viene richiamata la funzione AJAX *requestHTTP()* che effettua una richiesta HTTP all'URL https://ledlampsweb.it/control/mode.php?opMode=sound_reactive;
5. La pagina *mode.php* invia un'altra richiesta HTTP al server Java con la libreria cURL: il nuovo indirizzo è http://80.211.47.247:8000/[idLamp]/mode?opMode=sound_reactive;
   N.B. Oltre alla modifica dell'IP e l'aggiunta dell'idLamp, è stato anche rimosso ".php" dall'indirizzo, dato che non viene richiamata una pagina PHP sul server web del server Java.
6. Il server Java riceve la richiesta, controlla l'id della lampada e invia al socket del client la stringa *"/mode?opMode=sound_reactive";*
7. Il dispositivo master delle lampade riceve il messaggio, lo interpreta e imposta la nuova modalità. Successivamente invia un messaggio JSON al server Java comunicando il suo nuovo stato.
8. Il server Java riceve il messaggio e, attraverso una query SQL eseguita dalla libreria JDBC, aggiorna il record della tabella Lamps con i nuovi dati, dove idLamp è quello del client.
9. Nel frattempo, il server WEB si collega al database e legge i dati della lampada (la cui modalità in esecuzione sarà cambiata).
10. La pagina *mode.php* ritorna come risposta alla richiesta AJAX i dati codificati in JSON.
11. Viene mandata in esecuzione la funzione JavaScript *load()* che interpreta i dati JSON e aggiorna la pagina *index.php*.

8.2. ACCESSO CONCORRENTE AL DATABASE

Dalla sequenza di fasi appena descritta si può notare che possono accadere alcune anomalie dovuti all'accesso concorrente al database. Infatti, può accadere che il server web PHP voglia accedere al database per leggere i dati della lampada aggiornati quando ancora il dispositivo master non ha risposto: in questo caso il server web PHP leggerà i dati vecchi. Per impedire ciò ho usufruito del meccanismo dei **lock** fornito dal DMBS.
Il server Java, prima di inviare sul socket del client il messaggio con la nuova modalità, effettua un lock in lettura all'interno di una transazione:

```
SELECT * FROM lamps WHERE idLamp=[idLamp] FOR UPDATE;
```

In questo modo impedisce ad altri utenti di leggere le righe coinvolte fino al termine della transazione. In seguito, quando il client risponderà con i nuovi aggiornamenti e verrà eseguito l'**UPDATE** della tabella Lamps, viene completata la transazione con l'istruzione **COMMIT** e il lock viene rilasciato automaticamente.
Se il server web PHP prova a leggere la riga su cui è stato effettuato il lock prima che questo sia rilasciato, la query si interrompe e rimane in attesa di poter accedere alla risorsa. In questo modo si evitano anomalie dovute alla lettura di dati non ancora aggiornati. Per fare ciò è necessario eseguire la seguente query:

```
SELECT * FROM lamps WHERE idLamp=[idLamp] LOCK IN SHARE MODE;
```

9. SICUREZZA

Per quanto riguarda la **sicurezza** dell'intero sistema, sono stati adottati alcuni accorgimenti per garantire integrità, privatezza e autenticazione.
Innanzi tutto, è necessario autenticare i client che si connettono al server Java. Infatti, la connessione viene effettuata tramite socket TCP e chiunque che conosca IP (80.211.47.247) e port (12345) del server può accedervi. Per questo motivo, è necessario crittografare i messaggi che vengono scambiati tra dispositivo master e server Java in modo da garantire una certa dose di sicurezza.

Per garantire l'autenticazione, una soluzione potrebbe essere utilizzare un algoritmo di crittografia a chiave asimmetrica tipo **RSA**. In questo modo, criptando con la chiave privata (conosciuta solo dal client) e decriptando con la chiave pubblica (conosciuta dal server) è possibile garantire la paternità del messaggio. Infatti, se il server ottiene qualcosa di sensato decriptando, vuol dire che il pacchetto è stato effettivamente mandato dal client, dato che solo lui è in possesso della chiave privata che lo ha generato. Questo algoritmo tuttavia presenta uno svantaggio da non sottovalutare: le operazioni di crittografia sono relativamente lunghe e rallentano il processo di comunicazione. La soluzione migliore sarebbe utilizzare la crittografia ibrida, scambiando la chiave simmetrica criptandola con un algoritmo asimmetrico. In seguito i messaggi saranno criptati in modo simmetrico, con conseguente miglioramento in velocità. In questo modo si può garantire l'autenticazione del client e anche la privatezza della comunicazione. Tuttavia, questa soluzione è anche piuttosto complicata da implementare.
Durante lo sviluppo del progetto ho incontrato alcune difficoltà per sviluppare questa soluzione. Il problema principale nasce dal fatto che client e server sono programmati in due linguaggi diversi: C/C++ e Java. Naturalmente, i socket si trovano a livello trasporto e tutti i linguaggi di programmazione implementano API che li gestiscono. Infatti, la comunicazione TCP è realizzabile in modo semplice utilizzando le librerie apposite. Nonostante ciò, non sono riuscito a trovare API completamente funzionanti ed efficienti che implementassero l'algoritmo RSA per l'applicazione della crittografia asimmetrica. Java offre varie librerie che gestiscono la crittografia e includono anche l'algoritmo RSA. Al contrario, non ne ho trovate altrettante per lo sviluppo del codice della scheda NodeMCU. Questa, in effetti, ha molta meno capacità di elaborazione di un computer, e un algoritmo così complesso è piuttosto difficile da implementare in maniera efficace.

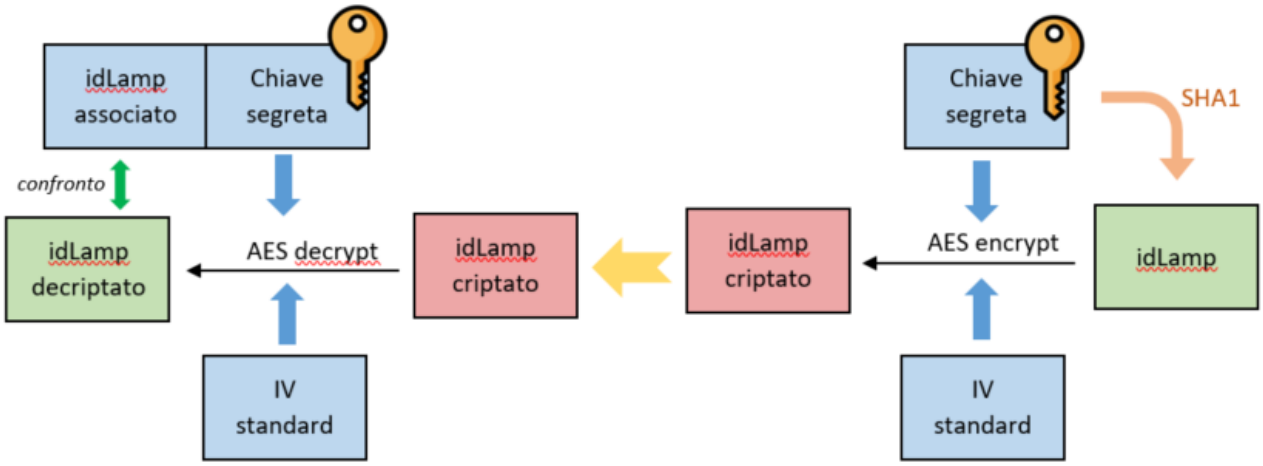
Per questo motivo, ho ideato un protocollo alternativo per garantire lo stesso grado di sicurezza. Questo si basa sull'algoritmo di crittografia simmetrica **AES**, che utilizza la stessa chiave per criptare e decriptare. Il limite degli algoritmi simmetrici è che, per comunicare in modo segreto, bisogna inizialmente accordarsi riguardo la chiave da utilizzare. In realtà, nel mio caso, questa fase non è necessaria, dato che la chiave segreta è univoca per ogni lampada, uguale per ogni comunicazione e già memorizzata sia sul client che sul server (il quale la legge dal database).
L'Advanced Encryption Standard (AES) è stato sviluppato nel 1997 per sostituire l'ormai obsoleto DES. L'algoritmo Rijndael, sviluppato da due crittologi belgi, vinse il concorso organizzato dal NIST (National Istitute of Standards and Technology) e fu adottato come nuovo standard crittografico. L'algoritmo si basa su una chiave di dimensioni minime di 128 bit, ha un'elevata efficienza computazionale, è semplice da codificare, molto flessibile e molto versatile. È un cifrario a blocchi di 128 bit che vengono disposti in una griglia di 4x4. La codifica è composta da 10 round suddivisi in 4 fasi:

1) *Substitute Bytes*: ogni byte viene trasformato mediante una permutazione non lineari di byte che vengono mappati tramite una tabella particolare.
2) *Shift Rows*: le righe della matrice subiscono uno scorrimento circolare di bytes nell'array.

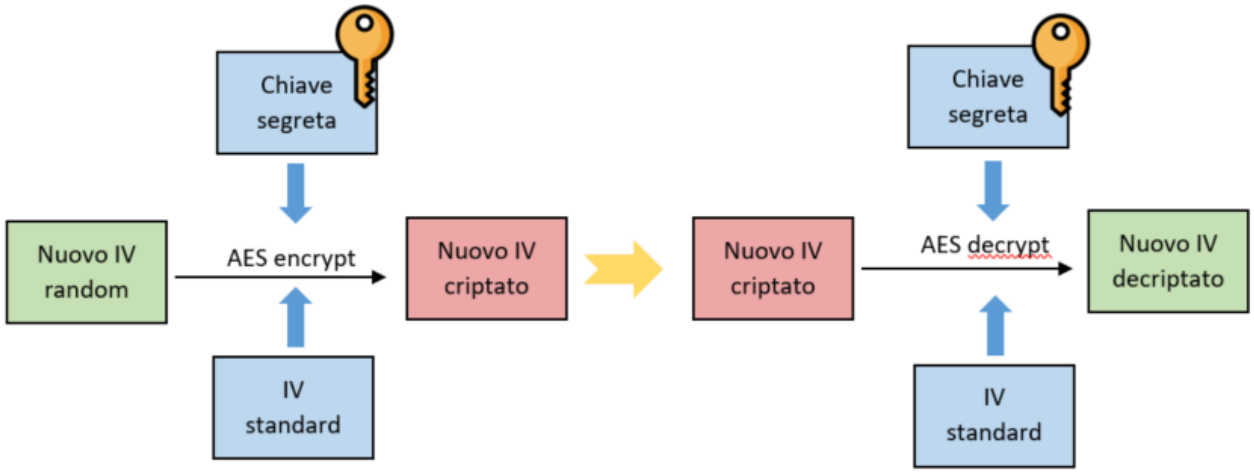
3) *Mix Columns*: ogni colonna viene trasformata mediante una operazione che può essere vista come una moltiplicazione matriciale con una particolare matrice generata da un polinomio prefissato.
4) *Add Round Key*: viene inserita la chiave segreta che rende il cifrario sicuro: ogni byte viene combinato in XOR con la chiave da 128 bit (16 bytes).

Ad oggi non sono conosciuti attacchi in grado di violarlo dato che impiegherebbero tempi inaccessibili. Come tutti i cifrari a blocchi, viene utilizzato un vettore di inizializzazione, dall'inglese **initialization vector** (IV): è un blocco di bit di lunghezza predefinita che viene utilizzato per inizializzare lo stato del cifrario, in modo che a chiavi identiche il keystream risultante sia differente. Questa funzione verrà utilizzata, come spiegato in seguito, per modificare il risultato della codifica a ogni connessione.

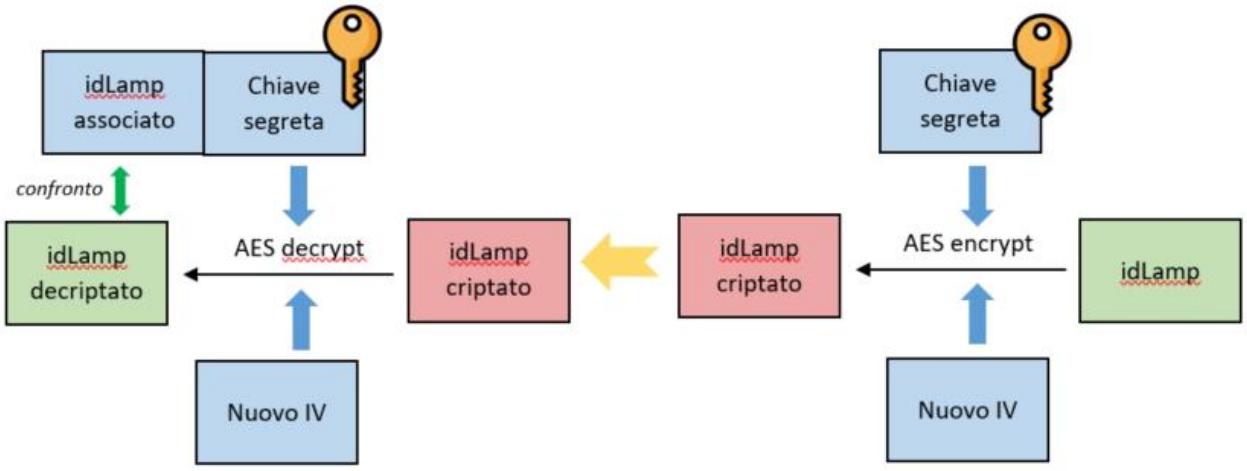
Nel protocollo che ho sviluppato, questo algoritmo utilizza una chiave da 16 bytes per criptare i messaggi tra dispositivo master e server Java, garantendo la privatezza dei dati inviati. Oltre a ciò, si può anche garantire l'autenticazione del client, dato che la chiave simmetrica è univoca per ogni lampada. Quando il dispositivo master si connette al server Java, invia il proprio idLamp (che è la chiave di crittografia codificata in SHA1) al server criptandolo con la sua chiave segreta. Il server, una volta aperta la connessione socket, prova a decriptare il messaggio con tutte le chiavi a lui conosciute, che sono lette dal database, verificando ogni volta il risultato ottenuto. Se questo corrisponde all'idLamp della lampada, la cui relativa chiave segreta è stata usata per decodificarlo, allora il mittente è effettivamente chi ha inviato il messaggio, dato che si presuppone la chiave sia conosciuta solo da lui e dal server. Al contrario, se la decodifica non verifica la condizione con nessuna delle chiavi, la connessione viene chiusa dal server e il client non potrà più comunicare, siccome non è stato identificato. Tutti i messaggi vengono codificati in **base 64** dopo essere stati criptati.



In seguito il server genera un IV random che verrà usato per la corrente sessione e lo invia al client criptato.



Il client risponde poi con il suo idLamp criptato con il nuovo IV.



9.1. FASI DELL'HANDSHAKE

Di seguito sono riportate in dettaglio tutte le fasi dell'handshake.

1. Quando si vuole connettere al server, il dispositivo master cripta il suo idLamp con la sua chiave segreta e un initialization vector default; poi lo invia al server per autenticarsi.

```
void handshake(){
  memcpy(my_iv, default_iv, 16);                // my_iv è l'array con l'IV usato per criptare:
                                                // ci copio il vettore di inizializzazione default
  client.println(AES_encrypt(sha1key));
  // ...
}
```

2. Il server Java legge il messaggio di richiesta connessione e prova a decriptarlo con le chiavi che conosce, confrontando il risultato con l'idLamp. Se trova una corrispondenza, memorizza le informazioni della chiave.

```
public AESKey handshake_findKey(String enc){
    for (AESKey aesKey : keys) {
        try {
            String dec = AES.decrypt(enc, aesKey.getKey(), aesKey.getIvParameterSpec());
            if(dec.equals(aesKey.getSha1key()))
                return aesKey;
        } catch (Exception e) {
            continue;
        }
    }
    return null;
}
```

3. A questo punto genera un nuovo vettore di inizializzazione random, in modo che ogni sessione dello stesso client sia diversa dalla precedente, dato che a ogni richiesta di connessione il vettore cambia. Poi lo cripta utilizzando la chiave del client e il vecchio initialization vector default e lo invia.

```
private boolean handshake(){
    String input = in.readLine();
    AESkey = keychain.handshake_findKey(input);
    byte[] newiv = AES.getIV();
    String response = Base64.getEncoder().encodeToString(newiv);
    String cipherText = AES.encrypt(response, AESkey.getKey(), AESkey.getIvParameterSpec());
    String len = response.length() + "$";
    out.print(len+cipherText);
    // ...
}
```

4. Il client riceve il messaggio, lo decripta e salva il nuovo vettore di inizializzazione: d'ora in poi, fino a che non si disconnette, utilizzerà questo per inizializzare i blocchi di codifica.

```
void handleIV(String enc){
    String dec = AES_decrypt(enc);              // decripto il messaggio
    byte plainiv[17];
    base64_decode((char *)plainiv, (char *)dec.c_str(), dec.length());
                                                // salvo nel vettore my_iv il nuovo IV ricevuto
    memcpy(my_iv, plainiv, 16);                 // che d'ora in poi userò per la crittografia
}
```

5. Poi, il dispositivo master cripta di nuovo il suo idLamp, utilizzando il nuovo initialization vector e la sua chiave segreta. Lo invia al server dicendogli che ha ricevuto il nuovo vettore ed è pronto a comunicare.

```
void handshake() {
  // ...
  sha1key_enc = AES_encrypt(sha1key);  // memorizzo il mio idLamp criptato per i prossimi messaggi
  client.println(sha1key_enc);
}
```

6. Il server, infine, controlla la risposta decriptandola con i nuovi parametri e, se il risultato è corretto e gli idLamp corrispondono, mantiene la connessione, altrimenti la chiude.

```
private boolean handshake(){
    // ...
    input = in.readLine();
    AESkey.setIvParameterSpec(new IvParameterSpec(newiv));      //new iv
    String plainText = AES.decrypt(input, AESkey.getKey(), AESkey.getIvParameterSpec());
    if(!plainText.equals(AESkey.getSha1key())){
        System.out.println("!! err2: new key not matching"+plainText);
        return false;
    }
 // memorizzo l'idLamp criptato per i prossimi messaggi
 AESkey.setSha1key_enc(AES.encrypt(AESkey.getSha1key(), AESkey.getKey(), AESkey.getIvParameterSpec()));
}
```

N.B. Dove nel codice si fa riferimento a "sha1key" si intende l'idLamp.

La sicurezza di questo processo sta nei seguenti punti:

1. Solo il vero dispositivo master che controlla le lampade che hanno un determinato idLamp conosce la chiave segreta con cui criptare e decriptare i messaggi: nessuno può leggere i suoi messaggi tranne il server.
2. Se un malintenzionato volesse sostituirsi al dispositivo master e connettersi al server potrebbe intercettare il pacchetto iniziale che esso invia al server per autenticarsi: anche se non sa il suo significato sa che il server lo interpreterà correttamente. A questo punto però, quando il server comunicherà il nuovo initialization vector da utilizzare, l'hacker non sarà mai in grado di decriptare il messaggio e interpretarlo, quindi non potrà rispondere con l'idLamp correttamente criptato. Anche perché il server non rimane in ascolto molto tempo ad aspettare la conferma da parte del client: allo scadere del timeout chiude la connessione perché il client impiega troppo per rispondere.
3. Cambiando il vettore di inizializzazione ad ogni sessione, i malintenzionati non potranno utilizzare vecchi pacchetti intercettati per connettersi al server e, inoltre, avranno più difficoltà a forzare i messaggi.

L'unico svantaggio di questa soluzione è che la chiave segreta per criptare i dati è sempre uguale per lo stesso client. Quindi se qualcuno la scopre (per esempio rubandola dal database o forzando l'idLamp codificato in SHA1), si può sostituire ad esso. A questo scopo è necessario difendere la tabella AESkey del database da chi non ha l'autorizzazione a leggerne il contenuto, gestendo correttamente i privilegi delle utenze del DBMS.

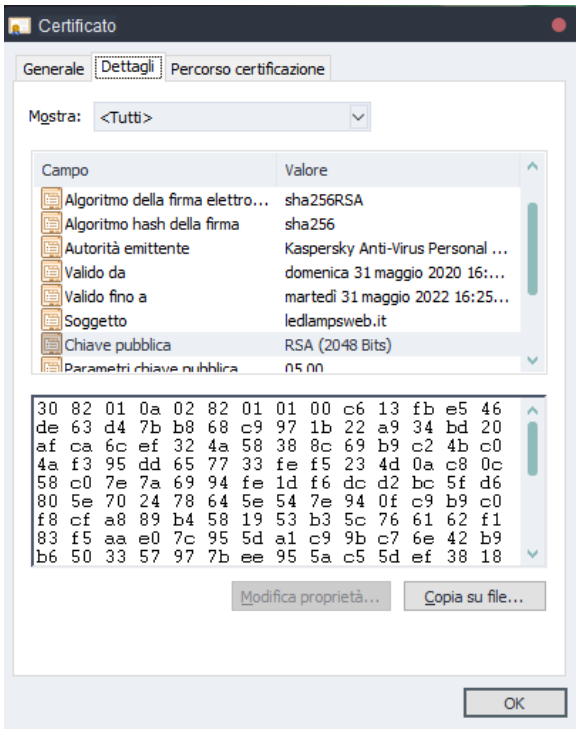
9.2. PROTOCOLLO HTTPS

Un'altra accortezza importante per il progetto è l'utilizzo di un protocollo per le richieste al server web che cripti le connessioni ed eviti che qualcun altro si spacci per esso e rubi i dati di utenti e lampade. Invece di effettuare normali richieste HTTP, infatti, si è scelto di implementare il protocollo **HTTPS**. Questo si basa sul protocollo SSL/TLS che garantisce la sicurezza del collegamento mediante tre funzionalità principali:

1) *Privatezza del collegamento*: crittografia simmetrica (DES/RC4)
2) *Autenticazione*: crittografia a chiave pubblica (RSA/DSS)
3) *Affidabilità*: controlli su integrità dei messagi (SHA/MD5)

HTTPS combina HTTP e SSL ed usa la porta 443. I siti web che utilizzano questo protocollo possiedono un certificato (*stardard X.509*) contente le autorizzazioni e le caratteristiche della connessione. Il certificato contiene il nome e indirizzo del server, la relativa chiave pubblica e il nome dell'autorità certificante. Quest'ultimo significa che il certificato è firmato digitalmente da un'autorità che si fa garante dell'autenticità delle informazioni contenute nel certificato.

Il protocollo **TSL** è composto da due livelli: *Record Protocol* e *Handshake Protocol*. Quest'ultimo si occupa della fase di negoziazione in cui si autentica l'interlocutore e si stabiliscono le chiavi segrete condivise. In particolare, il server invia al client un certificato che attesta la sua identità e contiene la chiave pubblica. Il client verifica l'identità, genera una chiave di sessione e la invia al server cifrandola con la chiave pubblica. Il server decifra il messaggio con la propria chiave privata e ottiene la chiave di sessione che verrà utilizzata per criptare la comunicazione.



Certificato HTTPS (standard X.509) di www.ledlampsweb.it

Il sito web www.ledlampsweb.it è certificato dalla certification autority *Let's Encrypt* che permette di generare un certificato valido in maniera gratuita.

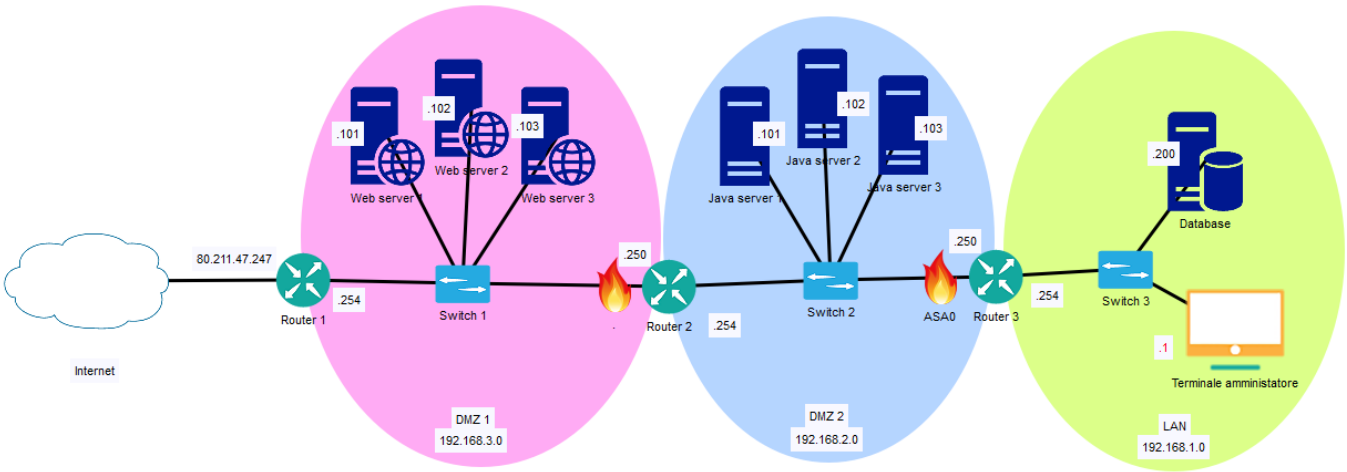
Garantendo questa sicurezza si è sicuri che nessun malintenzionato possa visualizzare le credenziali che gli utenti usano per accedere alla propria lampada, compreso l'idLamp, e nemmeno i comandi che l'utente esegue.

Tuttavia, solo i messaggi provenienti dall'utente e dirette al server web utilizzano questo protocollo. Le richieste che il server PHP invia al server Java utilizzano semplicemente il protocollo HTTP, senza nessuna criptazione. Per questo motivo, è stata ipotizzata la realizzazione di una rete aziendale, descritta in seguito, dove server PHP e Java sono all'interno di DMZ protette da attacchi di eventuali hacker.

10. RETE AZIENDALE

Tutto il processo descritto è stato realizzato in maniera esemplificativa e in modo che tutto il sistema funzioni correttamente. Naturalmente, possiede alcuni svantaggi. Basti pensare al fatto che il messaggio che l'utente invia dall'applicazione Android per controllare i propri dispositivi deve "rimbalzare" su Internet per due server (prima quello web, poi quello Java) prima di raggiungere il dispositivo master delle lampade. Si potrebbe obiettare che questo percorso sia inutile, dato che le lampade che devo controllare, generalmente, si trovano vicino a me, quando mi trovo a casa. Ovviamente, se ci si trova a casa, è possibile controllarle localmente connettendosi alla loro rete Wi-Fi privata, come spiegato. Tuttavia, i vantaggi di avere a disposizione un database dove viene memorizzato lo stato delle lampade sono immediati. Infatti con il sistema realizzato è possibile controllare i propri dispositivi anche a distanza.
Ciononostante, il sistema potrebbe essere ottimizzato. Avendo la possibilità di costruire una rete aziendale dedicata, si riuscirebbero ad evitare i "rimbalzi" dei messaggi su Internet. Inoltre, utilizzando un sistema distribuito si ottimizzerebbero anche le connessioni dei client al server Java. Dato che un singolo server faticherebbe ad accogliere le richieste di molti client, si potrebbe ipotizzare di costruire una **server farm** per gestire tutti gli utenti che necessitano di utilizzare questo prodotto: il sistema di load balancing gestirà il numero di connessioni. La rete seguente realizza un sistema distribuito che, ipoteticamente, sostituirebbe i due server separati utilizzati nel progetto. È stata sviluppata un'archietettura a 4 tier:

- applicazione Android = *presentation layer*;
- server web PHP = *application logic layer 1*;
- server Java = *application logic layer 2*;
- database = *resource management layer*.



Nella rete sono presenti 3 LAN:

- **192.168.3.0**: primo livello del sistema dove sono presenti i server web raggiungibili dall'esterno.
- **192.168.2.0**: secondo livello dove sono presenti i server Java che gestiscono le connessioni delle lampade.
- **192.168.1.0**: rete interna privata dove è presente il database e gli eventuali terminali degli amministratori.

Le reti rosa e blu sono due **DMZ** (Delimitarized Zone). Infatti, il loro accesso è protetto da firewall.
Il **router 1** implementa il NAT/PAT statico (PORT forwarding) e inoltra i pacchetti provenienti da Internet e diretti al port 443 (HTTPS) verso uno dei server web. Inoltre, fornisce a tutto il sistema un indirizzo pubblico attraverso cui può essere raggiunto su Internet (IP masquerading).

Il **router 2** implementa un firewall, perché la rete dei server Java non può essere raggiunta di chiunque. Al contrario, i pacchetti dei server web della *DMZ 1* sono autorizzati a passare perché devono riferire ai server Java i comandi che l'utente vuole inviare ai propri dispositivi. Anche i pacchetti provenienti dalle lampade poi dovranno poter passare attraverso il router 2, perché devono stabilire una connessione TCP con i server Java.
Il **router 3** implementa un secondo firewall, perché il database è ospitato sul server che necessita più protezione in quanto contiene i dati di lampade e utenti. Di qui non può passare nessuno eccetto i server web della *DMZ 1* e i server Java della *DMZ 2*, i quali sono autorizzati a leggere e scrivere nel database.

• TABELLE DI ROUTING

I dispositivi della rete 192.168.3.0 avranno come default gateway il router 1 (192.168.3.254).
Il **router 1** avrà la seguente tabella routing:

RETE	SUBNET MASK	NEXT HOP
192.168.2.0	255.255.255.0	192.168.3.250
192.168.1.0	255.255.255.0	192.168.3.250

I dispositivi della rete 192.168.2.0 avranno come default gateway il router 2 (192.168.2.254).
Il **router 2** avrà la seguente tabella di routing:

RETE	SUBNET MASK	NEXT HOP
0.0.0.0	0.0.0.0	192.168.3.254
192.168.1.0	255.255.255.0	192.168.2.250

I dispositivi della rete 192.168.1.0 avranno come default gateway il router 3 (192.168.1.254).
Il **router 1** avrà la seguente tabella di routing:

RETE	SUBNET MASK	NEXT HOP
0.0.0.0	0.0.0.0	192.168.2.254

• ACCESS CONTROL LISTS

Ipotizzando di utilizzare router Cisco, i firewall verranno implementati attraverso le seguenti **ACL**.

- Sul **router 2** è impostata la seguente ACL, in ingresso all'interfaccia 192.168.3.250:

- permette il passaggio di connessioni TCP provenienti da qualsiasi host verso un dispositivo della rete 192.168.2.0 con il port 12345 (connessioni delle lampade verso i server Java)
- permette il passaggio di connessioni http (port 80) provenienti da un dispositivo della rete 192.168.3.0 verso un server web della rete 192.168.2.0
- permette il passaggio di connessioni MySQL (port 3306) provenienti da un dispositivo della rete 192.168.3.0 verso il database della rete 192.168.1.0
- nega tutto il resto

```
Extended IP access list 100
    10 permit tcp any 192.168.2.0 0.0.0.255 eq 12345
    20 permit tcp 192.168.3.0 0.0.0.255 192.168.2.0 0.0.0.255 eq 80
    30 permit tcp 192.168.3.0 0.0.0.255 192.168.1.0 0.0.0.255 eq 3306
```

- Sul **router 1** è impostata la seguente ACL, in ingresso all'interfaccia 192.168.2.250:

- permette il passaggio di connessioni MySQL (port 3306) provenienti da un dispositivo delle reti 192.168.3.0 o 192.168.2.0 verso il database della rete 192.168.1.0
- nega tutto il resto

```
Extended IP access list 101
    10 permit tcp 192.168.0.0 0.0.3.255 192.168.1.0 0.0.0.255 eq 3306
```

11. APPLICAZIONE ANDROID

Come già anticipato, le lampade possono essere controllate anche attraverso un'applicazione Android dedicata. L'app mostra lo stato attuale delle lampade o, se sono disconnesse, il loro ultimo stato che è stato memorizzato nel database. Per di più, alcune funzionalità sono utilizzabili sono attraverso l'applicazione, come per esempio l'utilizzo delle automazioni. Il vantaggio di utilizzare l'applicazione invece che le pagine web, sta nel fatto che questa si occupa in automatico di individuare a chi indirizzare le proprie richieste HTTP in base alla rete Wi-Fi a cui è connesso il telefono. Infatti, se il cellulare è connesso a Internet, la richiesta sarà HTTPS e diretta verso il server PHP, mentre se è connesso alla rete delle lampade, la richiesta sarà HTTP e diretta verso il server web del dispositivo master. L'applicazione effettua tutto ciò in maniera trasparente per l'utente.

Le richieste HTTP sono effettuate attraverso la libreria Java *java.net.HttpURLConnection*. Mentre le richieste HTTPS sono rese possibili dalla libreria *javax.net.ssl.HttpsURLConnection*. L'esecuzione delle stesse viene effettuata in modo asincrono attraverso la classe Android *android.os.AsyncTask* che implementa un Thread eseguito separatamente dalle procedure che gestiscono la grafica. Questo perché Android non permette di effettuare richieste HTTP in modo sincrono, dato che andrebbero ad intralciare l'esecuzione dell'intera applicazione. In questo modo, le richieste vengono effettuate in background e, quando la risposta è pronta, viene richiamato un opportuno listener all'interno di un'activity, il quale interpreta il risultato e aggiorna l'app.

```
protected String doInBackground(String... strings) {
    try {
        strurl = strings[0];
        if(!strurl.contains("ledlampsweb")){     // se mi sto connettendo al dispositivo master elimino
            strurl = strurl.replace(".php","");  // .php dall'URL perché non sto chiedendo pagine PHP
            URL url = new URL(strurl);
            HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
            urlConnection.setConnectTimeout(timeout);
            urlConnection.setReadTimeout(timeout);
            return request(urlConnection);
        }
        else {
            URL url = new URL(strurl);
            HttpsURLConnection urlConnection = (HttpsURLConnection) url.openConnection();
            urlConnection.setConnectTimeout(timeout);
            urlConnection.setReadTimeout(timeout);
            return request(urlConnection, strings);
        }
    } catch (IOException e) {
        e.printStackTrace();
        return "exception";
    }
}

private String request(HttpURLConnection urlConnection) throws IOException{
    int responseCode = urlConnection.getResponseCode();
    if(responseCode==HttpURLConnection.HTTP_ACCEPTED)  // questo codice è inviato quando si vogliono
        return "sync";                                 // modificare le impostazioni delle lampade e il
                                                       // dispositivo master sta eseguendo il comando
    if(responseCode!=HttpURLConnection.HTTP_OK)
        return "error: server responded with code "+responseCode;

    InputStream response = urlConnection.getInputStream();
    Scanner scanner = new Scanner(response);
    String responseBody = scanner.useDelimiter("\\A").next();
    return responseBody;
}
```

```
protected void onPostExecute(String s) {
    super.onPostExecute(s);
    if(s!=null && !s.equals("exception")) {
        if(s.contains("error: ")){
                onSyncReadyListener.onError(s);
                return;
        }
        try {
            JSONObject jobj = new JSONObject(s);
            onSyncReadyListener.onSyncReady(jobj);
        } catch (JSONException e) {
            onSyncReadyListener.onError(s);
            e.printStackTrace();
        }
    }
    else {
        Toast.makeText(context, "An exception has occurred", Toast.LENGTH_SHORT).show();
        onSyncReadyListener.onError(s);
    }
}
```

Una nota importante da evidenziare è che il server PHP basa le sue pagine web sull'utilizzo delle sessioni. Infatti, all'accesso di un utente, viene memorizzato il relativo idLamp nell'array super globale *$_SESSION*, in modo che si possano autenticare le connessioni al server Java tramite esso, qualsiasi sia la pagina che le effettua. Purtroppo, le sessioni necessitano di un browser che acceda alle pagine web. Al contrario, l'applicazione Android non è un browser ed effettua occasionalmente specifiche richieste HTTP. Per autenticare la lampada dell'utente che effettua le richieste, quindi, non si possono utilizzare i valori memorizzati dalle sessioni. La soluzione è molto semplice: basta allegare ad ogni richiesta un parametro **POST** contenente l'idLamp. Il server PHP controllerà inizialmente che non sia già salvato il valore dalle sessioni e, in caso contrario, leggerà il parametro POST inviato. Se né *$_SESSION* né *$_POST* contengono il parametro idLamp, allora la richiesta sarà scartata.

L'invio di parametri POST in Android è implementato in questo modo:

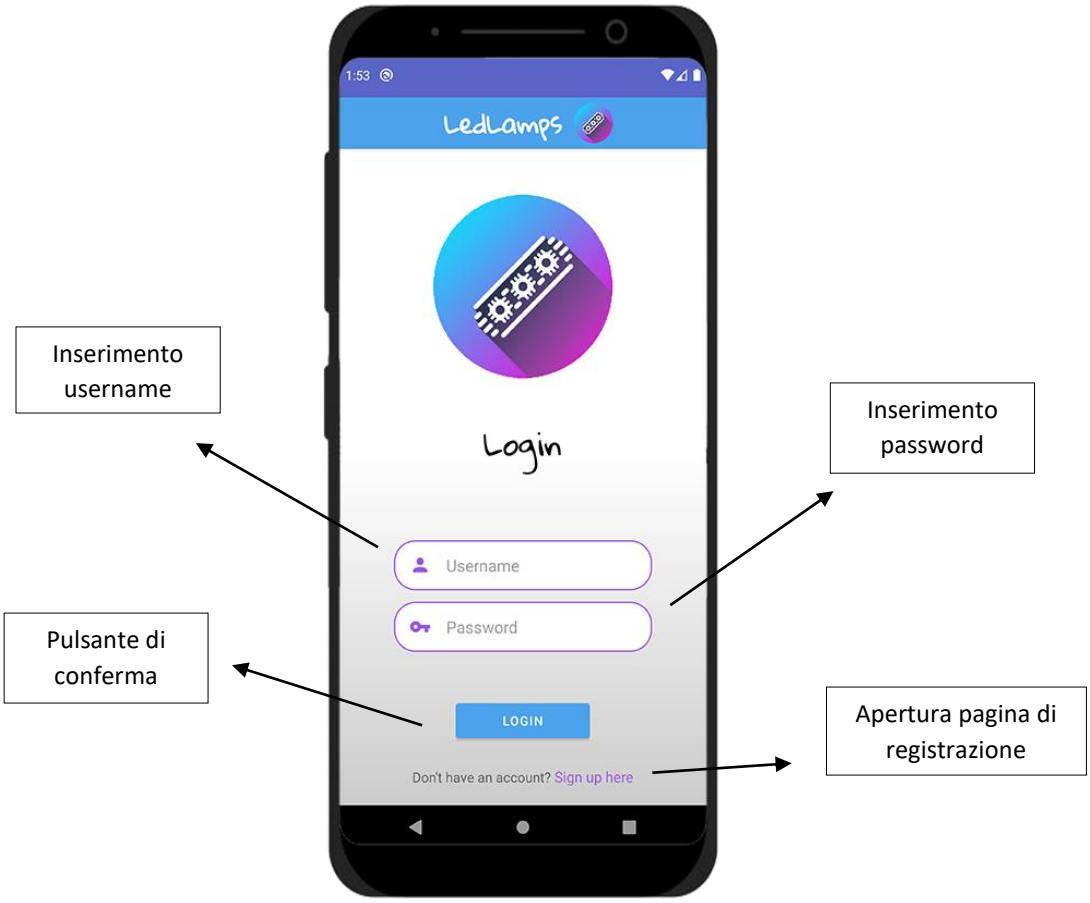
```
private String request(HttpsURLConnection urlConnection, String... strings) throws IOException{
    urlConnection.setDoOutput(true);
    String content = "idLamp=" + URLEncoder.encode(User.getIdLamp(), "utf-8");
    urlConnection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
    urlConnection.setRequestProperty("Content-Length", "" + content.getBytes().length);
    OutputStream output = urlConnection.getOutputStream();
    output.write(content.getBytes());
    output.close();

    // ...
}
```

C'è da sottolineare che, se il telefono è connesso alla rete delle lampade, alcune funzionalità saranno limitate. In particolare, non sarà possibile accedere alle informazioni del proprio account (rimangono memorizzate dopo l'ultimo login, ma potrebbero non essere aggiornate). Inoltre, non è possibile effettuare un nuovo login (perché l'autenticazione delle credenziali avviene sul server web PHP) e nemmeno creare un nuovo account. Infine, non è possibile utilizzare le automazioni. Naturalmente, l'applicazione si preoccuperà di informare l'utente di tutto ciò.

Di seguito sono riportate le varie sezioni.

- LOGIN

Permette di inserire le credenziali dell'utente che vuole utilizzare l'applicazione. Questo avviene solo al primo avvio, dato che all'accesso i dati vengono memorizzati. Le credenziali per l'accesso sono solamente username e password dato che si presuppone che l'utente abbia già inserito in fase di registrazione l'id della sua lampada: in questo modo non c'è bisogno di ricordarsi il lungo id ogni volta che si vuole accedere. Premendo il pulsante di login si verrà reindirizzati alla pagina principale.



All'avvio dell'applicazione viene effettuata una richiesta HTTPS del tipo: https://www.ledlampsweb.it/contol/login.php?details=true, passando tramite POST username e password (se eventualmente salvati). Il server web, se le credenziali sono corrette, risponde con tutti i dati dell'utente (letti dal database), compreso l'idLamp. Altrimenti l'utente viene reindirizzato alla pagina di *login*. Qui, inserirà le proprie credenziali e verrà eseguito di nuovo il controllo tramite la richiesta HTTP precedente. Se il server PHP risponde correttamente verranno salvati i dati in un'apposita classe statica e poi verrà aperta la pagina principale.

```
public void onSyncReady(JSONObject jobj) {
    User.setUser(
            jobj.getInt("idUser"),
            jobj.getString("username"),
            jobj.getString("password"),
            jobj.getString("name"),
            jobj.getString("surname"),
            jobj.getString("mail"),
            jobj.getString("idLamp")
    );
    Intent i = new Intent(LoginActivity.this, MainActivity.class);
    startActivity(i);
}
```

- REGISTRAZIONE UTENTE

Permette a nuovi utenti di inserire i propri dati per creare un account e controllare le proprie lampade. Tra le varie informazioni richieste, è necessario inserire anche l'id della propria lampada, che verrà associato all'account creato.

Tutti i campi sono obbligatori, l'applicazione si occupa di verificare la loro validità. A quel punto, al click del pulsante, l'applicazione effettuerà una richiesta http del tipo https://www.ledlampsweb.it/contol/login.php, inviando tramite POST tutti i dati (la password criptata con MD5). Il server web si preoccupa di controllare che l'username non sia già in uso ed effettua una query **INSERT** per aggiungere il nuovo utente nel database. Se la query fallisce, può essere che l'idLamp non è valido e la chiave esterna non rispetta il vincolo di integrità referenziale. In caso di errore, quindi, viene inviato un messaggio all'applicazione che mostrerà un Toast per informare l'utente. Altrimenti, se nessun errore viene ricevuto, l'applicazione aprirà la pagina di login per permettere all'utente di accedere con le nuove credenziali.

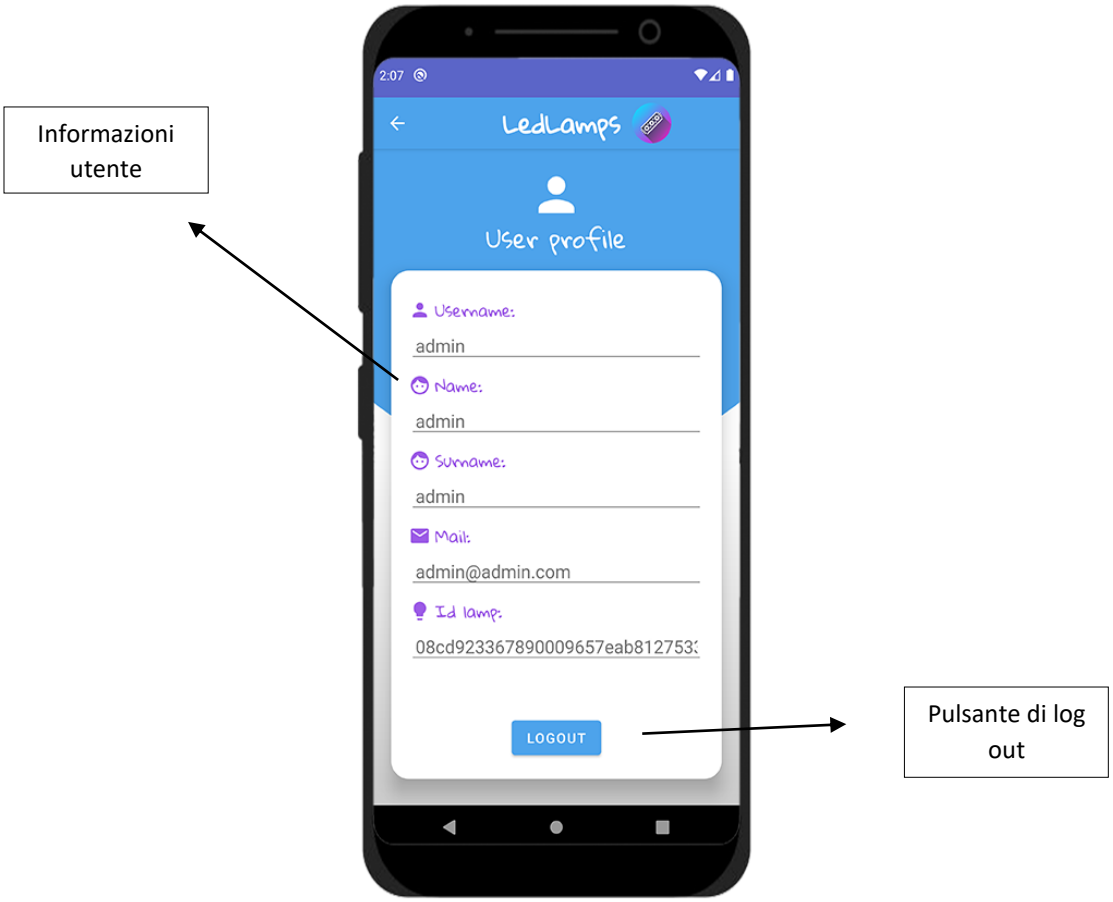
```
public void onClick(View v) {
    // controlli sui dati inseriti ...

    JsonHttpRequest jsonHttpRequest = new JsonHttpRequest(this, true);
    jsonHttpRequest.setOnSyncReadyListener(SignUpActivity.this);
    jsonHttpRequest.execute(LedLampsUtils.getHost()+"/signup.php",
            strNome,
            strCognome,
            strMail,
            strUsername,
            User.md5(strPassword),
            strIdLamp
    );
}
```

Torna all'indice

```php
<?php
    $q = "SELECT username
          FROM users
          WHERE username = '$username'";
    $ris = mysqli_fetch_array(mysqli_query($con, $q), MYSQLI_ASSOC);
    if($ris != NULL){
        if($ris["username"] == $username)
        echo "error: username already taken";
    }
    else {
        $q = "INSERT INTO users (username, password, name, surname, mail, idLamp)
              VALUES ('$username', '$password', '$name', '$surname', '$mail', '$idLamp');";
        $ris = mysqli_query($con, $q);
        if(!$ris){ echo "error: idLamp does not exist"; }
    }
?>
```

- ACCOUNT UTENTE

In questa sezione sono visibili le informazioni dell'utente loggato ed è possibile effettuare un log out per utilizzare un altro account. Premendo il pulsante si verrà reindirizzati alla pagina di login.

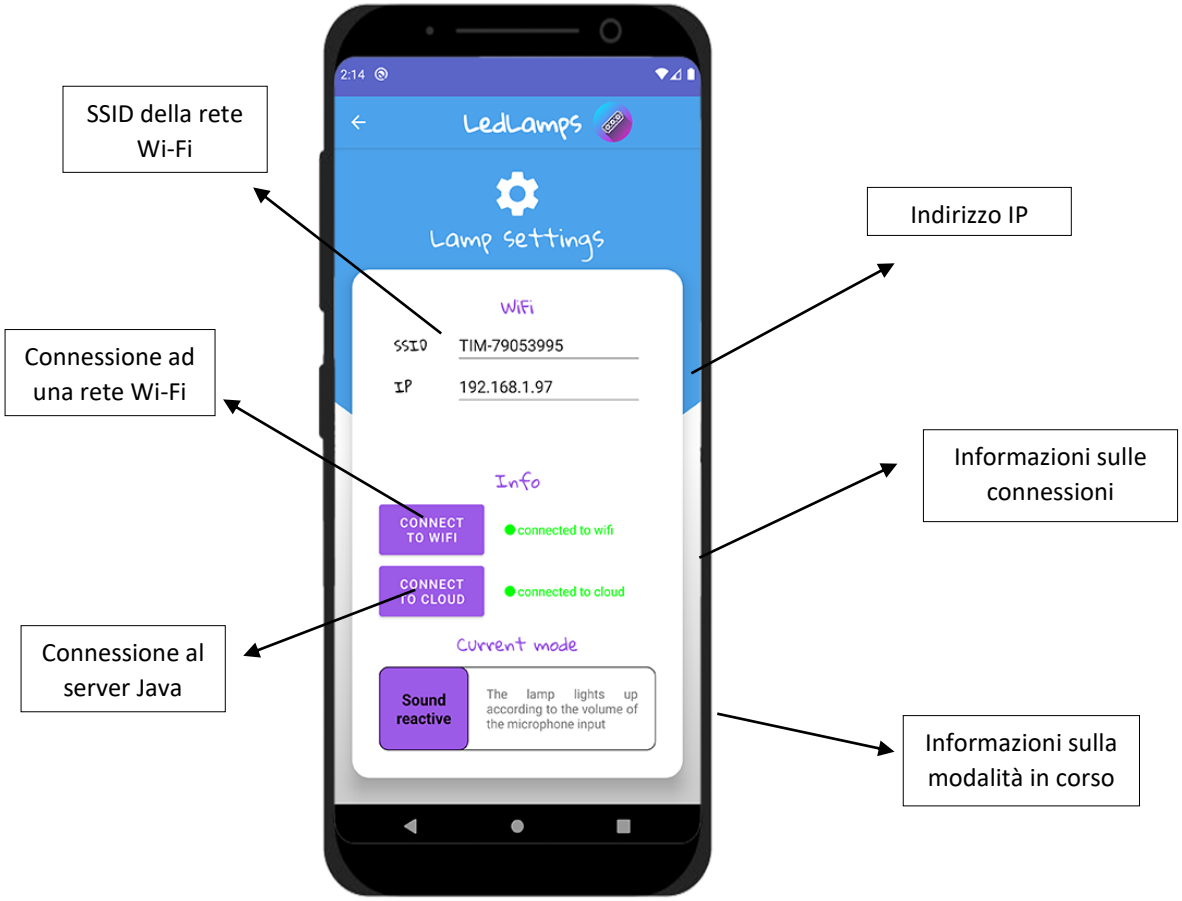


I dati sono memorizzati in una classe statica apposita (*User.java*) e sono richiesti al database (passando per il server WEB) ogni volta all'apertura dell'applicazione.

- SETTINGS

Questa sezione contiene le informazioni sullo stato delle lampade: la rete Wi-Fi a cui sono connesse, l'indirizzo IP del dispositivo master nella LAN domestica, la segnalazione di connessione alla rete Wi-Fi e al server e l'attuale modalità in corso.

Attraverso il primo pulsante è possibile connettere la lampada a una rete Wi-Fi, o se lo è già, è possibile cambiarla. Con il secondo pulsante invece si connette la lampada al server Java (se non lo è già).



Aprendo la pagina, l'applicazione effettua una richiesta HTTPS di *sync*: https://www.ledlampsweb.it/contol/sync.php?details=true. Il server PHP leggerà le informazioni sullo stato della lampada dal database, attraverso un'opportuna query, e risponderà con il risultato. L'applicazione si preoccupa di interpretare i dati JSON e costruire la schermata. Facendo uno swipe dall'alto verso il basso la richiesta viene rifatta e i dati aggiornati.

```
<?php
    $q = "SELECT lamps.*, modes.modeName, modes.modeDesc FROM lamps INNER JOIN modes ON opMode=modeId
          WHERE idLamp='$idLamp' LOCK IN SHARE MODE";
    $risultato = mysqli_query($con, $q);
    $data = mysqli_fetch_array($risultato, MYSQLI_ASSOC);
    echo json_encode($data);
?>
```

Queste informazioni (a parte quelle riguardanti la modalità in corso che necessitano una richiesta al database) sono visualizzabili anche se si è connessi alla rete delle lampade. In quel caso, la richiesta HTTP sarà del tipo http://192.168.4.1/sync. Il dispositivo master inoltrerà le informazioni riguardanti il suo stato attuale (in pratica risponde con la stessa stringa JSON che invia al server Java per aggiornarlo dei suoi cambi di stato).

È possibile modificare la connessione alla rete Wi-Fi o al server Java solo se il telefono è connesso alla rete delle lampade, dato che la richiesta va inviata direttamente al dispositivo master che possiede le funzioni dedicate a queste operazioni. In questo modo si evitano anomalie dei i messaggi di risposta dovute alla connessione-disconnessione del dispositivo. Per connettere il dispositivo a una rete Wi-Fi, si preme il relativo pulsante e si inseriscono le credenziali per connettersi alla rete nel popup che compare. Se SSID è diverso da quello della rete a cui il dispositivo è attualmente connesso verrà effettuata una richiesta del tipo:

http://192.168.4.1/wifi?ssid=[ssid]&password=[password]. Il dispositivo master risponderà con un codice 202 (HTTP_ACCEPTED) che significa che ha ricevuto la richiesta e sta provando a connettersi. L'applicazione allora effettua una richiesta di *sync* verso il dispositivo master e si mette in attesa della risposta (mostrando una progress bar). Questo ci metterà un po' a rispondere, perché sta effettuando un'altra operazione. Quando ha completato la connessione, risponderà all'applicazione con il suo nuovo stato. Da questo messaggio si può capire se il tutto è andato a buon fine o meno: se i parametri riguardanti la connessione Wi-Fi sono vuoti (vedi tabella Lamps), la connessione è fallita.

```
public void onClick(View v) {
    if(LedLampsUtils.getSystemSsid().equals("\"LedLamps\"")) {
        WifiDialog wifiDialog = new WifiDialog();
        wifiDialog.show(getSupportFragmentManager(), "wifi dialog");
    }
}
```

```
void handleWifi(String ssid, String pass){
  server.send ( 202, "text/plain", "connecting...");

  if(client.connected()) {                      // prima di tutto mi disconnetto dal server Java
    Serial.println("client disconnected");
    client.println(AES_encrypt("#"));
  }
  client.stop();
  bool res = connectToWifi(ssid, pass);         // poi mi connetto alla rete Wi-Fi con le credenziali
  Serial.println(res);
}
```

Per connettere il dispositivo al server Java, invece, viene inviata una richiesta del tipo http://192.168.4.1/connect?host=[javaServer_ip]. Come nel caso precedente, il dispositivo master risponderà con il codice 202 e la verifica della corretta connessione al server Java viene effettuata in modo analogo.

- ## MAIN ACTIVITY

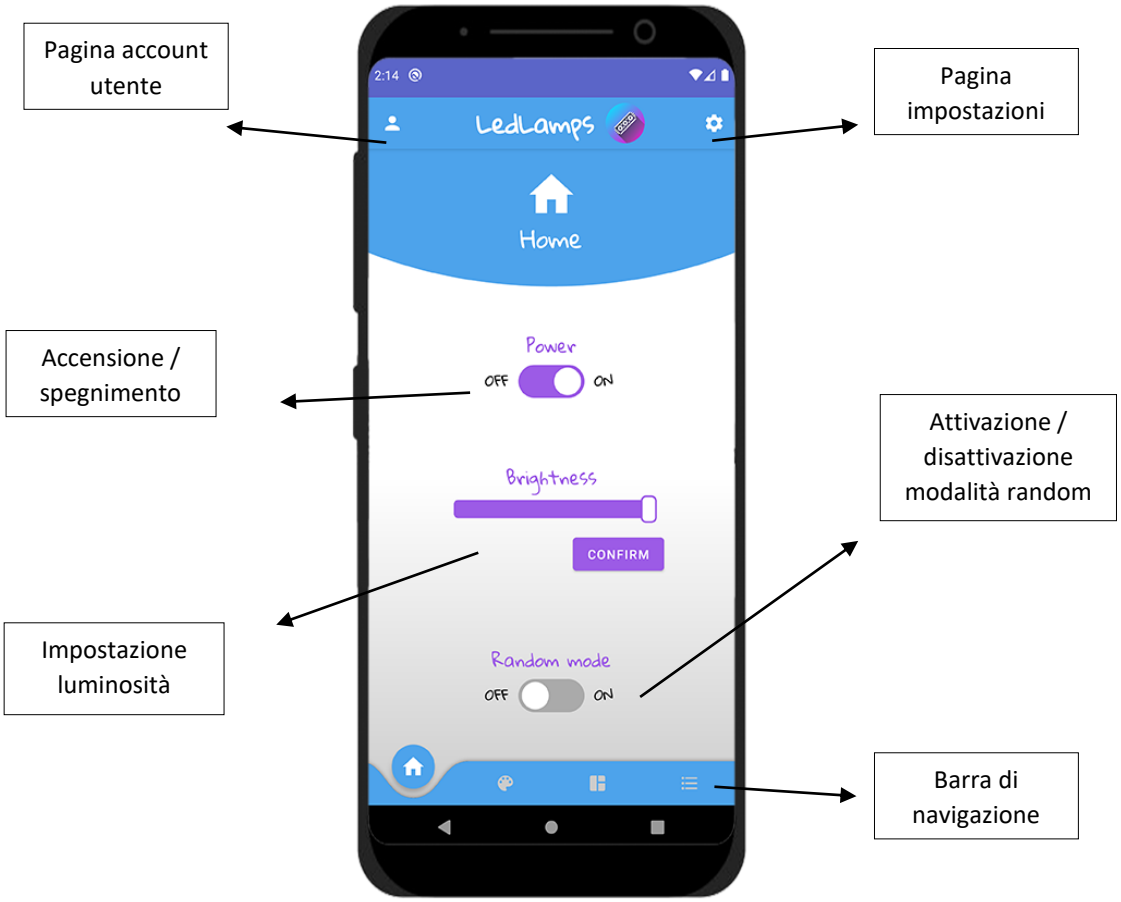
L'activity principale è un insieme di fragment che divide l'applicazione nelle 4 sezioni principali, che permettono di controllare le lampade. È possibile navigare tra le schermate attraverso la barra inferiore o facendo swipe laterali. Inoltre, attraverso le icone superiori, è possibile aprire le pagine precedentemente descritte. Con uno swipe dall'alto verso il basso è possibile aggiornare le informazioni riguardanti lo stato della lampada. Infatti, verrà effettuata una richiesta di *sync* verso il server PHP (che leggerà dal database) o verso il dispositivo master (che invierà il suo stato). Ognuno di essi estende la classe astratta *SyncronizableFragment.java* che definisce il metodo *sync()* per effettuare la richiesta HTTP.

```
public void sync(){
    JsonHttpRequest jsonHttpRequest = new JsonHttpRequest(getContext(), showToast);
    jsonHttpRequest.setOnSyncReadyListener(HomeFragment.this);
    jsonHttpRequest.execute(LedLampsUtils.getHost()+"/sync.php");
}
```

```
swipeRefreshLayout = view.findViewById(R.id.swipe_refresh);
swipeRefreshLayout.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {
    @Override
    public void onRefresh() { sync(); }
});
public void onSyncReady(JSONObject jobj) {
    // aggionramenti ...
}
```

- HOME

Permette di accendere/spegnere le lampade, regolare la luminosità e attivare/disattivare la modalità random.



Per accendere o spegnere le lampade viene effettuata una richiesta HTTP:
- al server PHP https://www.ledlampsweb.it/control/mode.php?opMode=[on|off]
- o al dispositivo master http://192.168.4.1/mode?opMode=[on|off].

```
public void onClick(View v) {
    JsonHttpRequest jsonHttpRequest = new JsonHttpRequest(getContext(), showToast);
    jsonHttpRequest.setOnSyncReadyListener(HomeFragment.this);
    if(power.isChecked())
        jsonHttpRequest.execute(LedLampsUtils.getHost()+"/mode.php?opMode=on");
    else
        jsonHttpRequest.execute(LedLampsUtils.getHost()+"/mode.php?opMode=off");
}
```

Per attivare o disattivare la modalità random viene effettuata una richiesta HTTP:
- al server PHP https://www.ledlampsweb.it/control/random.php?enabled=[true|false]
- o al dispositivo master http://192.168.4.1/random.php?enabled=[true|false].

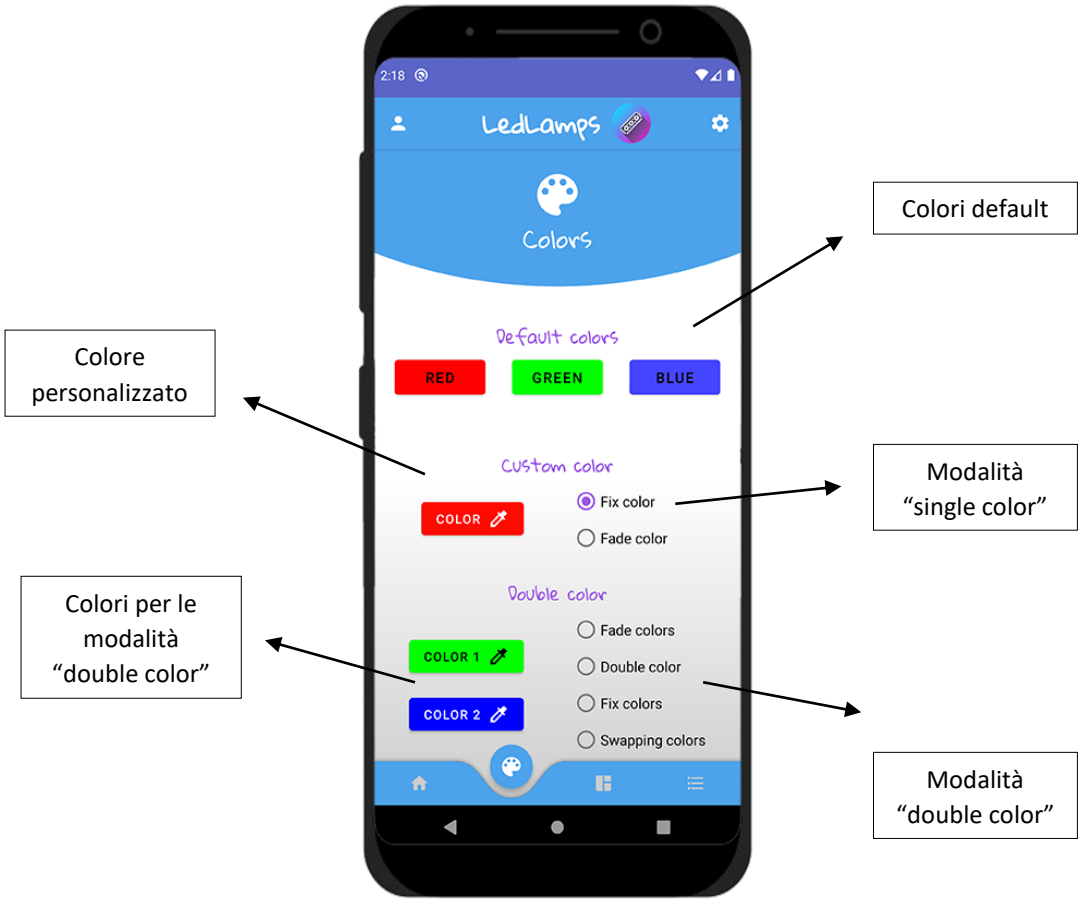
Per modificare la luminosità basta trascinare la seek bar e premere il tasto conferma. Verrà inviata una richiesta HTTP:
- al server PHP https://www.ledlampsweb.it/control/set_brightness_hash.php?brightness=[0-255]
- o al dispositivo master http://192.168.4.1/set_brightness_hash?brightness=[0-255].

In ogni caso, la risposta sarà un messaggio di *sync* (JSON) con il nuovo stato assunto dalla lampada. Nel caso in cui le lampade non siano connesse al server Java e si prova a controllarle attraverso il server PHP, il cambio di modalità fallirà, non sarà applicato e verrà mostrato un Toast con il messaggio di errore.

Torna all'indice

- COLORS

Permette di impostare il colore delle lampade e di attivare le modalità “single color” o “double color”. Attraverso i pulsanti superiori si imposta un colore default (rosso, verde o blu). Con il pulsante centrale, invece, si può scegliere un colore personalizzato. I due pulsanti inferiori, invece, permettono di impostare due colori per le modalità “double color”. Cliccando i pulsanti, si apre un popup con un color picker per la scelta del colore. I radio button settano la modalità desiderata.



Quando si clicca uno dei pulsanti “default colors” viene effettuata una richiesta HTTP:
- al server PHP https://www.ledlampsweb.it/control/mode.php?opMode=[red|green|blue]
- o al dispositivo master http://192.168.4.1/mode?opMode=[red|green|blue].
Quando si clicca uno dei radio button viene effettuata una richiesta HTTP:
- al server PHP https://www.ledlampsweb.it/control/mode.php?opMode=[opMode]
- o al dispositivo master http://192.168.4.1/mode?opMode=[opMode].
N.B. Il parametro *opMode* è l’attributo *modeLinkParam* della relativa modalità nella tabella Modes.

Quando si clicca uno dei pulsanti per scegliere il colore si apre un popup con un color picker. Una volta scelto il colore viene effettuata una richiesta HTTP tra le seguenti, al server PHP
https://www.ledlampsweb.it/control/set_color_hash.php?color=[#RRGGBB],
https://www.ledlampsweb.it/control/set_color_hash.php?first=[#RRGGBB]&second=[#RRGGBB]
o al dispositivo master
http://192.168.4.1/set_color_hash?color=[#RRGGBB],
http://192.168.4.1/set_color_hash?first=[#RRGGBB]&second=[#RRGGBB].
N.B. Quando si sceglie un colore non viene in automatico impostata la modalità corrispondente. A meno che non sia già in esecuzione, per attivarla bisogna cliccare il radio button relativo.

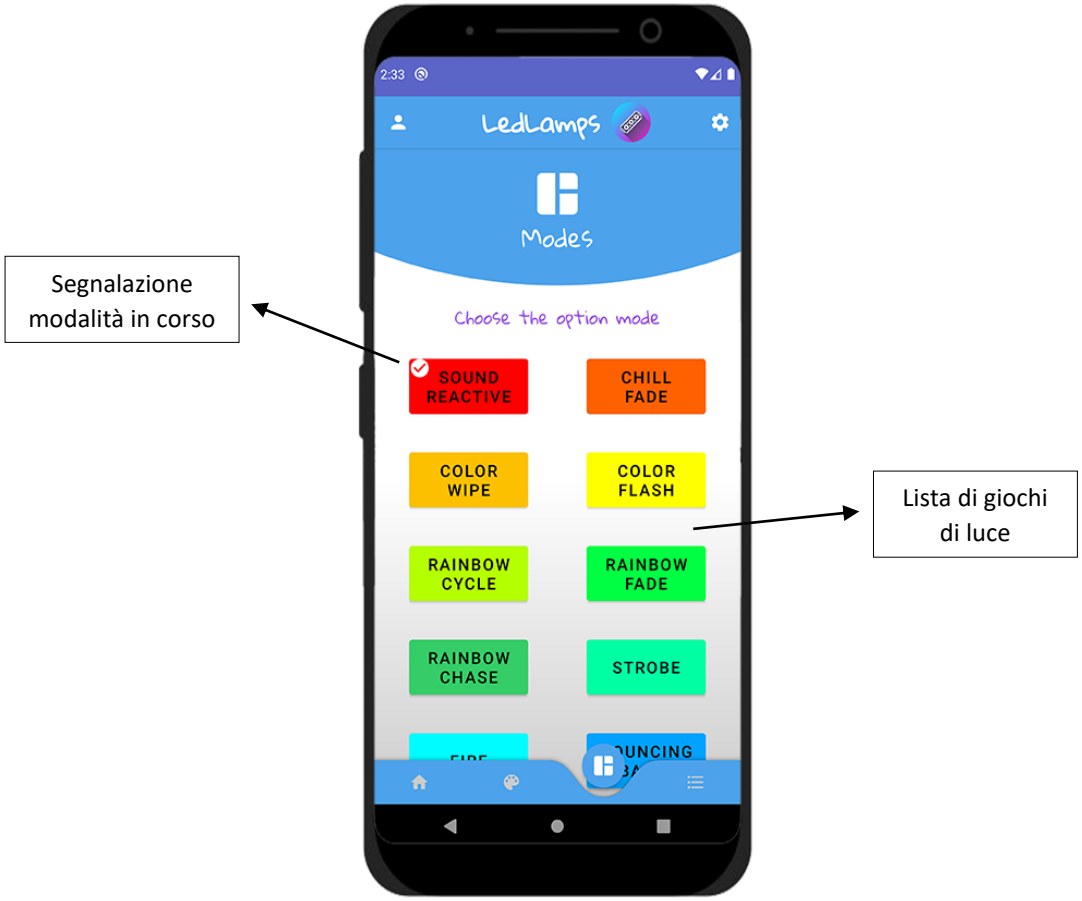
Torna all’indice

```
public void onColorSelected(ColorEnvelope envelope, boolean fromUser) {
    JsonHttpRequest jsonHttpRequest = new JsonHttpRequest(getContext(), showToast);
    jsonHttpRequest.setOnSyncReadyListener(ColorsFragment.this);
    if(button.getId() == R.id.custom){           // se sto impostando il colore "custom"
jsonHttpRequest.execute(LedLampsUtils.getHost()+"/set_color_hash.php?color=%23"+envelope.getHexCode().s
ubstring(2));
    }
    else if(button.getId() == R.id.fade1){       // se sto impostando il colore "fade1"
        String hexColor1 = "first=%23" + envelope.getHexCode().substring(2);
        String hexColor2 = "second=%23" +
Integer.toHexString(fade2.getBackgroundTintList().getDefaultColor()).substring(2);
        jsonHttpRequest.execute(LedLampsUtils.getHost()+"/set_color_hash?"+hexColor1+"&"+hexColor2);
    }
    else if(button.getId() == R.id.fade2){       // se sto impostando il colore "fade2"
        String hexColor1 = "first=%23" +
Integer.toHexString(fade1.getBackgroundTintList().getDefaultColor()).substring(2);
        String hexColor2 = "second=%23" + envelope.getHexCode().substring(2);
        jsonHttpRequest.execute(LedLampsUtils.getHost()+"/set_color_hash?"+hexColor1+"&"+hexColor2);
    }
}
```

In ogni caso, la risposta sarà un messaggio di *sync* (JSON) con il nuovo stato assunto dalla lampada. Nel caso in cui le lampade non siano connesse al server Java e si prova a controllarle attraverso il server PHP, il cambio di modalità fallirà, non sarà applicato e verrà mostrato un Toast con il messaggio di errore.

- MODES

Permette di impostare un gioco di luce cliccando sul relativo pulsante. Una spunta sopra uno dei pulsanti indica che quella modalità è attualmente in esecuzione.



Quando si clicca su un pulsante viene effettuata una richiesta HTTP:
- al server PHP https://www.ledlampsweb.it/control/mode.php?opMode=[opMode]

- o al dispositivo master http://192.168.4.1/mode?opMode=[opMode].

N.B. Il parametro *opMode* è l'attributo *modeLinkParam* della relativa modalità nella tabella Modes.

```
public void onClick(View v) {
    JsonHttpRequest jsonHttpRequest = new JsonHttpRequest(getContext(), showToast);
    jsonHttpRequest.setOnSyncReadyListener(ModesFragment.this);
    switch(v.getId()){
        case R.id.mode1:
            jsonHttpRequest.execute(LedLampsUtils.getHost()+"/mode.php?opMode=sound_reactive");
            break;
        // ...
    }
}
```

In ogni caso, la risposta sarà un messaggio di *sync* (JSON) con il nuovo stato assunto dalla lampada. Nel caso in cui le lampade non siano connesse al server Java e si prova a controllarle attraverso il server PHP, il cambio di modalità fallirà, non sarà applicato e verrà mostrato un Toast con il messaggio di errore.

• AUTOMATIONS

Permette di gestire le automazioni delle lampade. Cliccando su un elemento della lista si attiva o disattiva la relativa automazione (una sfumatura evidenzia l'automazione attualmente in esecuzione). Il pulsante + in alto fornisce la possibilità di aggiungere una nuova automazione: va scelto un nome identificativo (diverso da quelli già presenti) e almeno una modalità. Per eliminare un'automazione, invece, basta tenere premuto il relativo elemento della lista. La freccia a sinistra di ogni automazione permette di visualizzare le azioni che la compongono: è possibile eliminarle (swipe verso sinistra), trascinarle in una nuova posizione o aggiungerne.

Come già spiegato, le automazioni possono essere utilizzate solo attraverso l'applicazione Android dato che, effettivamente, queste sono implementate dal server Java quando riceve un opportuno messaggio dal server PHP. All'apertura della pagina, viene fatta una richiesta HTTPS attraverso cui si richiedono al server PHP tutte le automazioni https://www.ledlampsweb.it/control/automations/select.php. Attraverso opportune query vengono ottenuti i dati delle automazioni collegate alla propria lampada e viene ritornato un array JSON. L'applicazione lo interpreta e costruisce la lista di automazioni.

```php
<?php
    $q = "SELECT automations.*, users.username FROM automations INNER JOIN users USING (idUser)
          WHERE idLamp = '$idLamp'";
    $risultato = mysqli_query($con, $q);
    $data = mysqli_fetch_all($risultato, MYSQLI_ASSOC);
    foreach($data as $automation){
        $q = "SELECT actions.*, modes.modeName FROM actions INNER JOIN modes USING (modeId)
              WHERE idAutomation = ".$automation["idAutomation"];
        $risultato = mysqli_query($con, $q);
        $autom_arr = array();
        $autom_arr += ["idAutomation"=>$automation["idAutomation"], "idUser"=>$automation["idUser"],
"name"=>$automation["name"], "username"=>$automation["username"], "isActive"=>$automation["isActive"]];
        $autom_arr += ["actions"=>mysqli_fetch_all($risultato, MYSQLI_ASSOC)];
        array_push($obj["automations"], $autom_arr);
    }
    echo json_encode($obj);
?>
```

```json
//esemio di risposta JSON
{
   "automations":[
      {
         "idAutomation":"1",
         "idUser":"1",
         "name":"Automation 1",
         "username":"admin",
         "isActive":"0",
         "actions":[
            {
               "idAutomation":"1",
               "modeId":"41",
               "position":"1",
               "custom":null,
               "fade1":null,
               "fade2":null,
               "time":"5",
               "modeName":"On"
            },
            {
               "idAutomation":"1",
               "modeId":"44",
               "position":"2",
               "custom":"#ff0000",
               "fade1":null,
               "fade2":null,
               "time":"10",
               "modeName":"Custom color"
            },
            {
               "idAutomation":"1",
               "modeId":"42",
               "position":"3",
               "custom":null,
               "fade1":null,
               "fade2":null,
               "time":"10",
               "modeName":"Off"
            },
            {
               "idAutomation":"1",
               "modeId":"44",
               "position":"4",
               "custom":"#00ffa9",
               "fade1":"null",
               "fade2":"null",
               "time":"5",
               "modeName":"Custom color"
            }
         ]
      },
      {
         "idAutomation":"2",
         "idUser":"1",
         "name":"Automation 2",
         "username":"admin",
         "isActive":"0",
         "actions":[
            {
               "idAutomation":"2",
               "modeId":"41",
               "position":"1",
               "custom":null,
               "fade1":null,
               "fade2":null,
               "time":"5",
               "modeName":"On"
            },
            {
               "idAutomation":"2",
               "modeId":"42",
               "position":"2",
               "custom":null,
               "fade1":null,
               "fade2":null,
               "time":"5",
               "modeName":"Off"
            },
            {
               "idAutomation":"2",
               "modeId":"49",
               "position":"3",
               "custom":null,
               "fade1":"#00ff00",
               "fade2":"#0000ff",
               "time":"5",
               "modeName":"Swapping colors"
            }
         ]
      }
   ]
}
```

Per attivare/fermare un'automazione, viene inviata una richiesta HTTPS del tipo https://www.ledlampsweb.it/control/automations/automator.php?operation=[start|stop|update]&idAutomation=[idAutomation].

Il server PHP comunicherà poi la richiesta al server Java (analogamente a quello che fa per inviare un cambio modalità), il quale effettuerà l'operazione richiesta:

- **start** = fa partire l'automazione che ha id inviato
- **stop** = ferma l'automazione attualmente in corso
- **update** = aggiorna la lista di azioni dell'automazione in corso (quando si elimina/aggiunge/sposta un'azione dall'elenco)

Come risposta, il server ritorna il risultato di un'opportuna query che contiene le stesse informazioni di una richiesta di *sync*, ma in più viene aggiunto l'id dell'automazione in corso (se l'attivazione è andata a buon fine ed è stata memorizzata sul database). In questo modo viene aggiornata la lista e viene colorato lo sfondo dell'elemento automazione attivo.

```php
<?php
    $q = "SELECT lamps.*, if(isActive,idAutomation,0) as activeAutomation ".
         "FROM lamps INNER JOIN users USING(idLamp) INNER JOIN automations USING(idUser) ".
         "WHERE idLamp='$idLamp' ORDER BY activeAutomation DESC LIMIT 1 LOCK IN SHARE MODE;";

    $risultato = mysqli_query($con, $q);
    $data = mysqli_fetch_array($risultato, MYSQLI_ASSOC);
    echo json_encode($data);
?>
```

Per aggiungere un'automazione, bisogna cliccare il tasto **+** viola. Una volta inserito il nome dell'automazione e scelta la prima azione, viene fatta una richiesta HTTPS del tipo:
https://www.ledlampsweb.it/control/automations/create.php?idUser=[idUser]&name=[name]&modeId=[modeId]&custom=[#RRGGBB]&fade1=[#RRGGBB]&fade2=[#RRGGBB]&time=[time],
dove, oltre ai dati della nuova automazione (utente che l'ha creata e nome dell'automazione), vengono passati anche gli attributi dell'azione (id modalità, colori, durata). Il server PHP richiama la procedura MySQL *create_automation()* e, se va a buon fine, ritorna un array JSON analogo a quello di una richiesta alla pagina *select.php*.

```java
private void createAutomation(String automation_name, Action action){
    try {
        System.out.println(action);
        String custom = action.getCustom() != null ? action.getCustom() : "null";
        String fade1 = action.getFade1() != null ? action.getFade1() : "null";
        String fade2 = action.getFade2() != null ? action.getFade2() : "null";
        JsonHttpRequest jsonHttpRequest = new JsonHttpRequest(getContext(), showToast);
        jsonHttpRequest.setOnSyncReadyListener(AutomationsFragment.this);
        jsonHttpRequest.execute(LedLampsUtils.getHost()+"/automations/create.php?" +
                "idUser=" + User.getId() +
                "&name=" + automation_name +
                "&modeId=" + action.getModeId() +
                "&custom=" + URLEncoder.encode(custom, "utf-8") +
                "&fade1=" + URLEncoder.encode(fade1, "utf-8") +
                "&fade2=" + URLEncoder.encode(fade2, "utf-8") +
                "&time=" + action.getTime());

    } catch (UnsupportedEncodingException e){}
}
```

```php
<?php
    $q = "CALL create_automation($idUser, '$name', $modeId, 1, '$custom', '$fade1', '$fade2', $time);";
         $ris = mysqli_query($con, $q);
    if(!$ris)
        echo "error: creation failed (maybe you already have an automation called \"$name\")";
?>
```

Per eliminare un'automazione, viene fatta una richiesta HTTPS del tipo:
https://www.ledlampsweb.it/control/automations/delete.php?idAutomation=[idAutomation].
Analogamente a quanto succede per creare un'automazione, anche in questo caso il server PHP richiama la procedura MySQL *delete_automation()* e ritorna l'array JSON.

Per aggiungere un'azione ad un'automazione esistente, bisogna cliccare il tasto **+** blu. Una volta impostata l'azione, viene fatta una richiesta HTTPS del tipo
https://www.ledlampsweb.it/control/automations/addAction.php?idAutomation=[idAutomation]&position=[position]&modeId=[modeId]&custom=[#RRGGBB]&fade1=[#RRGGBB]&fade2=[#RRGGBB]&time=[time]. Il server PHP effettua una query **INSERT** e inserisce la nuova azione nel database. Se ciò va a buon fine ritorna un array JSON con la nuova lista di azioni relative a quell'automazione. L'applicazione lo interpreta e aggiorna i dati.

```
public void onActionCreated(Action action) {     // richiamata alla chiusura della finestra di dialogo
    try {                                         // che permette di definire la nuova azione
        action.setIdAutomation(automation.getIdAutomation());
        System.out.println(action);
        String custom = action.getCustom() != null ? action.getCustom() : "null";
        String fade1 = action.getFade1() != null ? action.getFade1() : "null";
        String fade2 = action.getFade2() != null ? action.getFade2() : "null";
        JsonHttpRequest jsonHttpRequest = new JsonHttpRequest(getContext(), showToast);
        jsonHttpRequest.setOnSyncReadyListener(AutomationsFragment.this);
        jsonHttpRequest.execute(LedLampsUtils.getHost()+"/automations/addAction.php?" +
                "idAutomation=" + action.getIdAutomation() +
                "&modeId=" + action.getModeId() +
                "&position=" + action.getPosition() +
                "&custom=" + URLEncoder.encode(custom, "utf-8") +
                "&fade1=" + URLEncoder.encode(fade1, "utf-8") +
                "&fade2=" + URLEncoder.encode(fade2, "utf-8") +
                "&time=" + action.getTime());

    } catch (UnsupportedEncodingException e){}
}
```

```
<?php
    $q = "INSERT INTO actions VALUES ($idAutomation, $modeId, $position, '$custom', '$fade1', '$fade2',
$time)";
    $ris = mysqli_query($con, $q);
    if(!$ris) echo "error: not inserted";
    else{
        $q = "SELECT actions.*, modes.modeName FROM actions INNER JOIN modes USING (modeId)
              WHERE idAutomation = ".$idAutomation;
        $risultato = mysqli_query($con, $q);
        $data = mysqli_fetch_all($risultato, MYSQLI_ASSOC);
        echo json_encode(array("actions"=>$data));
    }
?>
```

Per eliminare un'azione si effettua uno swipe laterale verso sinistra. Viene eseguita una richiesta HTTPS del tipo:
https://www.ledlampsweb.it/control/automations/deleteAction.php?idAutomation=[idAutomation]&position=[position]. Il server PHP richiama la procedura MySQL *delete_action()* e, se va a buon fine, ritorna un array JSON con la nuova lista di azioni relative a quell'automazione. L'applicazione lo interpreta e aggiorna i dati.

Per spostare un'azione in un'altra posizione bisogna tenere premuto e trascinarla. Viene eseguita una richiesta HTTPS del tipo:
https://www.ledlampsweb.it/control/automations/deleteAction.php?idAutomation=[idAutomation]&from=[vecchia_posizione]&to=[nuova_posizione]. Il server PHP richiama la procedura MySQL *move_action()* e, se va a buon fine, ritorna un array JSON con la nuova lista di azioni relative a quell'automazione. L'applicazione lo interpreta e aggiorna i dati.

12. CONCLUSIONE

L'idea per la realizzazione di questo progetto mi è venuta dopo aver visto un video sul web, dove un ragazzo costruiva delle lampade che si illuminavano a tempo di musica. Ho preso spunto da quel tutorial e ho realizzato anche io la mia versione. La struttura base del codice, infatti, è identica a quella utilizzata nel video e gestisce la comunicazione tra le lampade e il dispositivo con il microfono.
Successivamente, continuando i miei studi nell'ambito dell'informatica, ho voluto aggiungere nuove funzionalità al progetto originale. Ho implementato la possibilità di controllare le lampade non sono manualmente, attraverso il pulsante, ma anche attraverso uno smartphone che si connettesse al server web ospitato sulla scheda Wi-Fi del dispositivo. Poi, piano piano, ho aumentato le modalità disponibili, prendendo spunto da librerie online. In seguito, dopo aver approfondito la programmazione Android, ho voluto creare un'applicazione dedicata per controllare le lampade. Infine, ho aggiunto la possibilità di controllarle da remoto, appoggiandomi ad un server Java che facesse da intermediario. Tutto ciò è stato realizzato negli ultimi due anni, mentre la mia passione per l'informatica si stava sviluppando in contemporanea al percorso di studi.
Quando poi ho dovuto presentare un progetto per l'esame di stato, dopo aver analizzato l'ipotesi che un'azienda producesse un dispositivo del genere, elaborando un Business Plan, ho valutato la possibilità di estendere l'utilizzo delle mie lampade da parte di più utenti, ovviamente con un occhio di riguardo alla sicurezza. Così ho programmato il database che contenesse tutte le informazioni necessarie e ho aggiunto la possibilità di definire automazioni per le proprie lampade, considerati i molti vantaggi di archiviazione offerti dal DBMS. Infine ho cercato un servizio di hosting che mi permettesse di pubblicare online il mio sito web come se si trattasse di una vera start-up.

Sondrio, 31 maggio 2021 *Lorenzo Della Matera*