



UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

SCUOLA DI INGEGNERIA

Corso di Laurea in Ingegneria Informatica
Classe n. L-8

LEDPLATE: LED STRIPS MODULAR LEDWALL

Controllo remoto di un microprocessore:
da linea di comando a Progressive Web
App.

Relatore: Chiar.mo Prof. Stefano Paraboschi
Prova finale di Andrea Galliani
Matricola n. 1046750

ANNO
ACADEMICO
2018/2019



Abstract

In questo progetto si prende in carico il lavoro svolto per una vecchia tesi di laurea realizzata da due studenti di Ingegneria Informatica dell'Università degli Studi di Bergamo. Ne vengono velocemente illustrati la struttura e il funzionamento. Vengono analizzate le sue criticità, tra cui la totale assenza di un'interfaccia macchina-utente. L'obiettivo di questo progetto è quello di analizzare il vecchio progetto e creare un nuovo sistema che sia più strutturato ed efficiente. Questo elaborato va preso in considerazione insieme all'elaborato di altri due studenti del corso di Ingegneria Informatica: Simone Ciocca e Andrea Patelli. Essi hanno collaborato attivamente al progetto e nel loro elaborato illustrano ognuno una particolare sezione dello sviluppo del progetto. In questo elaborato si illustra l'unità fondamentale del progetto: il LED. Se ne analizzano velocemente le caratteristiche fisiche ed elettroniche. Si analizza il suo legame con i colori attraverso il modello RGB. Si vedono i LED RGB. Spiego, poi, la struttura ed il funzionamento dei LED WS2811 e il loro utilizzo per la realizzazione degli schermi LED da comandare. Illustro la problematica e la necessità di codificare un algoritmo di ri-ordinamento dei pixel che costituiscono le immagini da riprodurre da parte dei LED, in quanto gli schermi vengono realizzati con dei pannelli costruiti a partire da led-strips. Dobbiamo controllare i LED attraverso un dispositivo programmabile. Usiamo un Raspberry e costruiamo un sistema client(cmd line)-server per leggere, convertire e riprodurre immagini, GIF e video su schermi led realizzati con i ws2811. Ci si concentra, poi, sull'interfaccia grafica, che in futuro dovrà sostituire il client da linea di comando, ottenuta realizzando una Progressive Web App, standard di Google in uso dal 2019. Si illustra il funzionamento delle PWA, le tecnologie e gli strumenti utili per la progettazione e l'analisi dei suoi comportamenti. Si affronta, infine la codifica dell'applicazione vera e propria attraverso l'uso del framework Flutter, tecnologia rilasciata da Google nel 2018 studio. Il metodo di lavoro principalmente usato è la consultazione della documentazione ufficiale fornita da Google. Vengono mostrati i principali punti chiave del funzionamento dell'app e le schermate che ne derivano. Infine, vengono proposti alcuni aggiornamenti che è possibile dare al sistema per ottimizzarlo e renderlo più completo.

Sommario

Abstract	1
1 LedPlate	4
1.1 Introduzione.....	4
1.2 La tesi precedente.....	4
1.3 Criticità della vecchia tesi	6
1.4 Cosa abbiamo tenuto?.....	6
1.5 Perché LedPlate? da dove nasce il progetto?.....	8
1.6 Requisiti	8
2 LED	9
2.1 Emissione della luce.....	9
2.3 Alimentazione	11
2.4 Corrispondenza LED-Pixel e modello dei colori RGB	12
2.5 Modelli utilizzati: WS2811	15
3 Algoritmo di ri-ordinamento	18
4 Controllo dei LED	21
4.1 Arduino vs Raspberry.....	21
4.2 Progettazione del sistema	23
4.3 Il Server	24
4.4 Il Client (linea di comando).....	24
5 Interfaccia Grafica.....	27
5.1 Progressive Web App.....	27
5.1.1 Velocità e affidabilità	29
5.1.2 Installabilità	30
5.1.3 PWA Optimized	36

5.2 Service workers.....	39
5.2.1 Promise.....	40
5.2.3 Service worker life cycle	41
5.2.4 Gestione e uso di un service worker per il controllo di una pagina web.....	44
5.3 Flutter.....	47
5.3.1 Widgets	48
5.3.2 Panoramica del sistema	50
5.3.3 Costruire un widget.....	51
5.3.4 Gestione delle interazioni con l'utente.....	51
5.4 Prima Bozza UI	53
5.5 L'Applicazione	55
6 Funzionamento del sistema e schermate dell'app.....	64
7 Architettura di sistema con GUI in Flutter.....	72
Conclusioni	73
Sitografia.....	75

Nel documento sono presenti note indicate con una lettera alfabetica e note con un carattere numerico. Le prime indicano un collegamento ad una risorsa online, il cui link è presente nella sitografia in coda all'elaborato. Le seconde sono note a pie' di pagina.

1 LedPlate

1.1 Introduzione

LedPlate è una semplice piattaforma software che ha il compito di controllare la riproduzione di contenuti multimediali su dei pannelli LED. È un progetto nato da tre studenti di Ingegneria Informatica dell'Università degli Studi di Bergamo. L'obiettivo principale è stato quello di ampliare e migliorare un lavoro di tesi precedentemente svolto da altri due studenti del dipartimento di Ingegneria Informatica: il progetto Matelight^a.

Il progetto Matelight è un sistema informatico a scopo decorativo inserito in alcuni dei più moderni hackerspace europei: un muro fatto di bottiglie di vetro vuote che contengono LED programmabili. Il nome Matelight deriva dalle bottiglie di Club Mate^{1b} usate per costruirlo.

1.2 La tesi precedente

I colleghi della tesi precedente hanno utilizzato una scheda Arduino Uno^c, due addressable led-strips da 50 LED ciascuna e del codice in Java aperto ed eseguito in ambiente di sviluppo Eclipse.

Arduino è una piattaforma di prototipazione elettronica open-source che consente agli utenti di creare oggetti elettronici interattivi. Il modello Arduino Uno è una scheda a microcontrollore basata sul processore ATmega320P. Essa possiede 14 pin input/output digitali (di cui 6 possono essere usati come uscite PWM^{2d}), un cristallo di quarzo a 16MHz, una porta USB, power jack, un header ICSP e un bottone di reset.

ICSP sta per *In Circuit Serial Programming*, che rappresenta uno dei numerosi metodi disponibili per la programmazione delle schede Arduino. Permette di programmare una

¹ **Club-Mate** è una bibita a base di mate (*Ilex paraguariensis*, pianta della famiglia delle Aquifoliaceae, nativa del Sud America) che contiene caffeina e viene prodotta dall'azienda tedesca Loscher KG a Münchsteinach in Franconia.

² **Pulse-Width Modulation**: è un tipo di modulazione digitale che permette di ottenere una tensione media variabile dipendente dal rapporto tra la durata dell'impulso positivo e di quello negativo (duty-cycle).

scheda Arduino senza passare dal bootloader³, il che è utile se si vuole caricare un programma già sviluppato in un altro ambiente. In sostanza è possibile collegare la scheda Arduino alla porta USB del PC direttamente via cavo seriale e installarne l'eseguibile in pochi semplici passi.

Nel progetto Matelight, Arduino è stato utilizzato come controllore per indicare ai singoli led se e con che colore illuminarsi.

La parte sviluppata in Java, invece, riceve in input il percorso file di un'immagine o di una GIF⁴ e restituisce come output un codice compatibile con l'IDE⁵ di Arduino. Questo codice contiene delle istruzioni di comando per i LED e un vettore di interi che indicano i colori. Copiato l'output fornito dal programma in Java e incollato nell'IDE di Arduino, è possibile caricare l'eseguibile attraverso cavo seriale sulla scheda Arduino che, poi, permette il corretto funzionamento dei LED e la conseguente visualizzazione di immagini e GIF.



Le strisce LED sono state collegate alla scheda Arduino e montate a serpentina su un pannello di cartone. In questo modo è stato realizzato un pannello da 10x10LED. I LED sono stati montati a circa 3cm di distanza l'uno dall'altro, con conseguente dimensione del pannello di circa 1m².

³ Il **bootloader** è il programma che, nella fase di avvio del computer (boot), carica il kernel del sistema operativo dalla memoria secondaria alla memoria primaria, permettendone l'esecuzione da parte del processore e il conseguente avvio del sistema.

⁴ **Graphics Interchange Format:** formato di file per immagini digitali di tipo bitmap utilizzato nella grafica digitale. L'estensione .gif viene associata tipicamente ad immagini in movimento.

⁵ **Integrated Development Environment:** software che, in fase di programmazione, supporta i programmatore nello sviluppo del codice sorgente di un programma.

1.3 Criticità della vecchia tesi

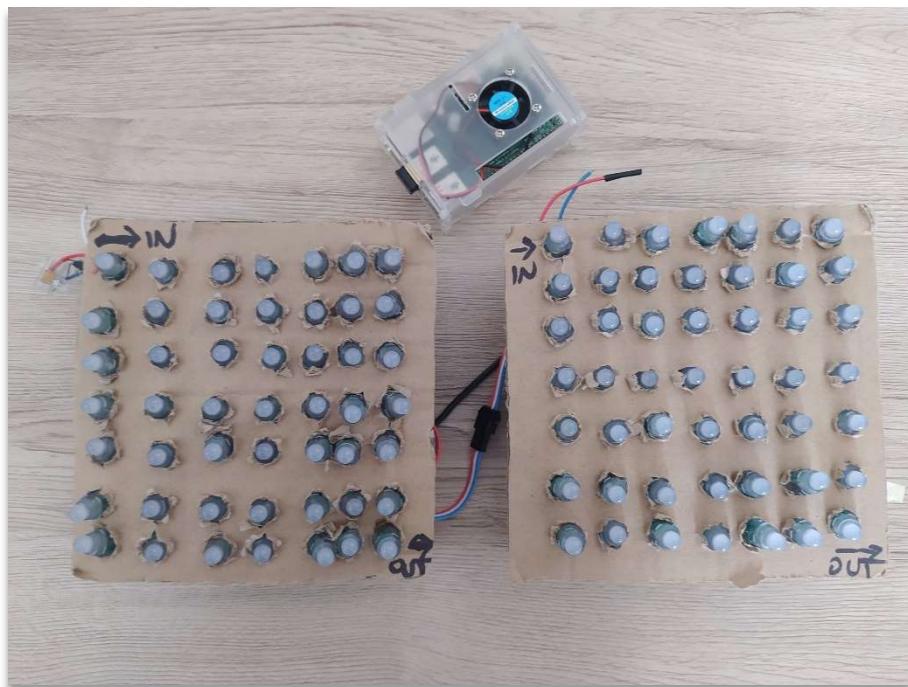
1. Processo di funzionamento macchinoso: la criticità più grande riguarda l'usabilità del sistema. Innanzitutto, bisogna avere sempre a disposizione un PC che abbia installato un IDE che supporti Java. Inoltre, è sempre necessario avere i file e le cartelle del progetto contenenti il codice sorgente del software. Per di più, si deve eseguire il programma e interagire con esso direttamente dalla console di esecuzione di Eclipse. Tutto questo per svolgere solo metà del processo. Dopo questi passaggi è poi necessario aprire l'IDE di Arduino per poter installare il programma che controlla i LED sulla scheda, che nel frattempo deve essere collegata al PC attraverso il cavo seriale.
Insomma, manca una vero e proprio approccio alla progettazione del sistema per renderlo standalone. Inoltre, è assente una interfaccia macchina-utente che renda il sistema *user-friendly* e che nasconda il codice all'utente.
2. Pannello LED: Uno schermo di tali dimensioni con così pochi LED comporta una risoluzione molto bassa a causa della scarsa densità di LED.
3. Non vengono riprodotti video.
4. Non è possibile controllare i LED da remoto.

1.4 Cosa abbiamo tenuto?

L'impostazione base delle strisce LED è la stessa della prima tesi Matelight: vincolo stringente è la configurazione a serpentina. Questa scelta ha portato molte difficoltà nella programmazione dell'algoritmo di gestione dei LED, tuttavia la compatibilità del sistema con delle strisce LED economiche controllabili conferisce all'intero sistema maggiore agilità.

Inoltre, un algoritmo simile può essere bypassato in modo semplice, rimuovendolo dalla catena logica di elaborazione dell'immagine. Di conseguenza, con LedPlate è possibile comandare sia pannelli costruiti con led-strips economiche (serpentina → algoritmo ON), sia con matrici di LED prefabbricate (sistema cartesiano xy → algoritmo OFF).

Come prima cosa è stata modificata la struttura del pannello LED a disposizione. Le due strisce erano lunghe 50 LED ciascuna. Ho rimosso un LED da ogni striscia così da ottenere due led-strips lunghe 49 elementi. In questo modo è stato possibile realizzare due pannelli LED di dimensione 7x7. Così facendo abbiamo avuto gli strumenti necessari per strutturare il progetto in modo modulare, con dimensioni ridotte (il lato di ogni pannello è lungo circa 20cm) e migliore comodità di test.



Ovviamente l'uso di questi pannelli artigianali è vincolato alla fase sperimentale del progetto. Il sistema finale è funzionante con qualunque pannello realizzato industrialmente che possieda LED compatibili.

Per continuità di obiettivo con la tesi Matelight, sarebbe sufficiente inserire i singoli LED all'interno di bottiglie di vetro per ottenere il muro di bottiglie illuminate degli hackerspace europei.

1.5 Perché LedPlate? da dove nasce il progetto?

L'obiettivo del nostro progetto è quello di riprodurre immagini, GIF e video su uno schermo LED realizzato con strisce LED economiche.

Il nome del progetto, LedPlate, è l'accostamento dei due elementi protagonisti del progetto: i LED e i pannelli, o altrimenti detti piastra/piatto di LED, che in inglese si dice Plate. Il concetto di pannello ha un ruolo decisivo per il progetto, in quanto si tratta dell'unità fondamentale che compone uno schermo.

Uno schermo è composto da pannelli (Plates), i quali sono a loro volta composti da LED. Date queste informazioni, per inizializzare il sistema, sarà innanzitutto necessario conoscere quanti pannelli compongono uno schermo e quanti LED compongono un pannello. Tutto questo è necessario per poter configurare il sistema e renderlo funzionante.

Prima di iniziare a lavorare al nostro progetto, però, è stato fondamentale decidere a priori quali fossero le nostre idee. Ci siamo trovati e abbiamo steso un testo che riassumesse in via generica i requisiti della nostra piattaforma.

1.6 Requisiti

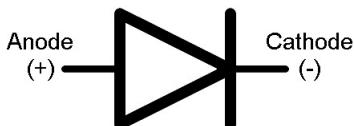
Vogliamo realizzare un dispositivo che sia in grado di controllare più pannelli LED.

Il dispositivo, che chiamo controllore, deve poter leggere delle memorie di massa o poter ricevere dei file via rete. Il controllore, scelto un file, deve elaborarlo in modo che gli schermi LED connessi mostrino il/i file scelti.

Il controllore deve inviare i segnali che fanno capire agli schermi quali LED accendere e con quale colore. L'utente deve poter accedere al pannello di controllo via web, inizializzare la dimensione e la quantità di pannelli presenti.

La pagina web deve essere accessibile anche da smartphone e da qui posso fare alcune delle operazioni che posso fare anche da pc. All'avvio accendo lo schermo collegato al primo pin del dispositivo e accedo alla pagina web di amministrazione, la quale come prima cosa mi chiede quanti schermi, composti da quanti pannelli e proporzioni ho a disposizione.

2 LED



Un LED, ovvero *Light Emitting Diode*, è un particolare tipo di diodo in grado di emettere radiazioni luminose.

Un diodo è un dispositivo semiconduttore a doppio strato (ovvero composto da due materiali conduttori diversi) che può funzionare in **polarizzazione diretta**, quando la differenza di tensione tra anodo e catodo è positiva e quindi la corrente è in grado di scorrere attraverso il dispositivo, oppure in **polarizzazione inversa**, situazione nella quale la differenza di tensione tra anodo e catodo è negativa e, di conseguenza, il diodo si oppone al passaggio di corrente.

Un diodo attraversato da corrente continua è riconducibile ad una resistenza o a un generatore di tensione, a seconda della polarizzazione. I valori di tale resistenza o di tale tensione del generatore sono parametri di costruzione.

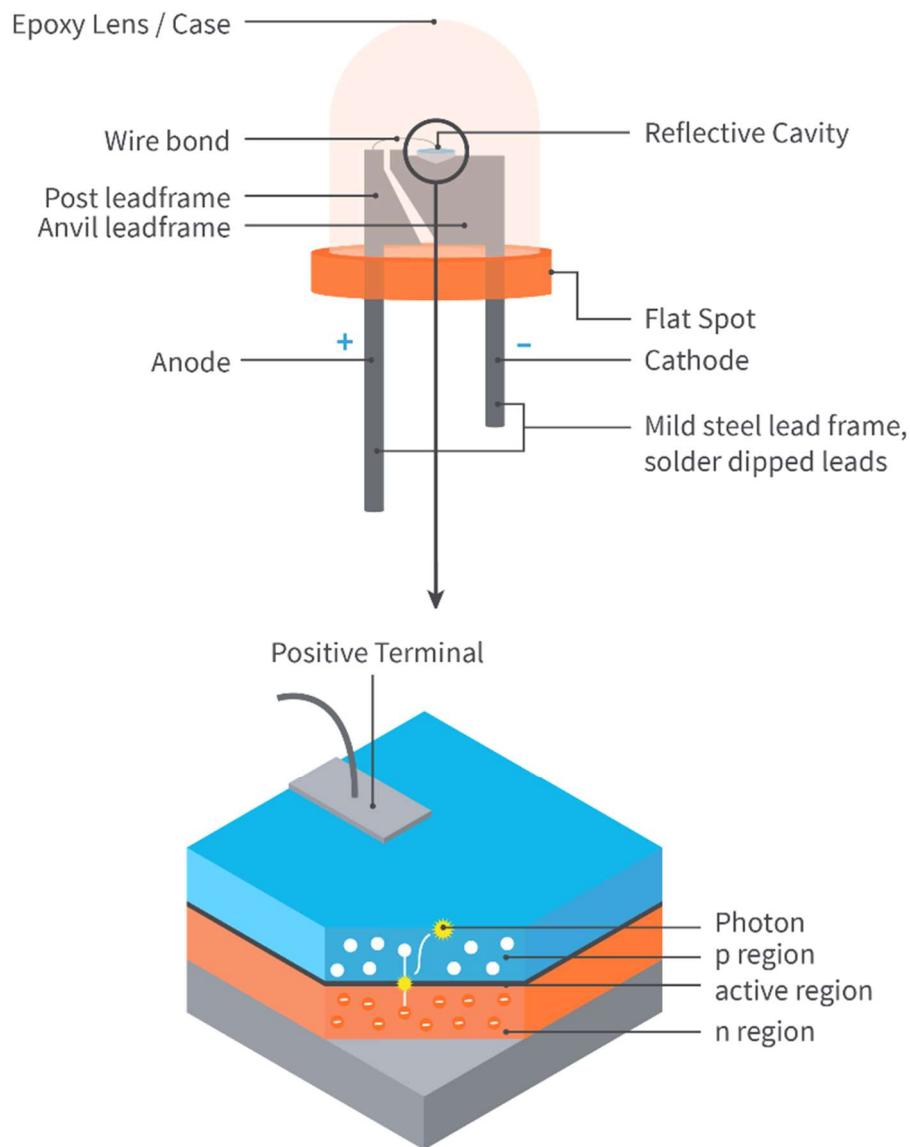
Un diodo attraversato da corrente alternata fa passare solo la semi-onda positiva della sinusoida che compone un segnale elettrico e blocca la semi-onda negativa (polarizzazione diretta).

2.1 Emissione della luce

I LED sono dei particolari diodi a giunzione P-N⁶. In quanto tali, essi prevedono il movimento di elettroni attraverso la giunzione dalla zona P alla zona N e viceversa per le lacune. Durante questo movimento le cariche transitano da un livello energetico alto ad un livello energetico più basso, ciò implica la presenza di energia liberata nella regione attiva del diodo. A livello atomico l'energia è una grandezza quantizzata, dunque abbiamo a che fare con "pacchetti di energia". Se l'energia liberata è sufficientemente elevata i pacchetti energetici sono **fotoni** e la ricombinazione elettroni-lacune viene definita *radiativa*, altrimenti l'energia viene liberata sotto forma di calore (**fononi**) e la ricombinazione è definita *non-radiativa*.

⁶ Con giunzione P-N si intende un dispositivo realizzato giustapponendo due strati di metallo semiconduttore (tipicamente Silicio) con drogaggio differente.

Se il chip che compone il diodo ha uno spessore sufficientemente ridotto, un ragionevole numero di fotoni può essere liberato (quindi vi è emissione di luce visibile) e il diodo può essere catalogato come trasduttore elettro-ottico.



Come si può ben vedere dal disegno, la giunzione P-N è di dimensioni molto ridotte rispetto alla dimensione complessiva del LED. L'80% del LED è costituito dal rivestimento di plastica colorata o trasparente che incapsula la giunzione P-N con lo scopo di proteggere la giunzione, fornire un sostegno fisico ai fili elettrici e favorire l'uscita della luce dal semiconduttore aumentando l'angolo di direzione⁷ dei fotoni. La plastica,

⁷ Riflessione, rifrazione e Leggi di Fresnel.

infatti, agisce come mezzo fisico intermedio tra il semiconduttore, che possiede un indice di riflessione relativamente alto, e l'aria con un indice di riflessione basso.

La frequenza della radiazione emessa è definita dalla quantità di energia liberata dal diodo. Maggiore energia liberata, maggiore frequenza. La scelta dei semiconduttori determina dunque la lunghezza d'onda (inverso della frequenza) dell'emissione dei fotoni e, quindi, il colore della luce.

L'efficienza della conversione elettrico-ottica determina l'intensità luminosa dei fotoni in uscita dal diodo.

Pertanto, **un LED è un particolare tipo di diodo a giunzione P-N che, percorso da corrente in polarizzazione diretta, è in grado di emettere luce.**

Negli schemi elettronici i LED sono indicati con il seguente simbolo:



2.3 Alimentazione

Affinché un LED sia in grado di emettere luce, dunque, è necessario che in esso scorra corrente in polarizzazione diretta con intensità sufficientemente elevata. Tale intensità di corrente, però, deve anche essere inferiore al valore massimo sopportabile dal LED in quanto, se superata la tensione di breakdown, il LED viene danneggiato.

I costruttori forniscono le specifiche di tensione o intensità di corrente necessarie ad alimentare (polarizzare) un LED. Ovvero, forniscono la tensione da applicare tra anodo e catodo in modo che all'interno del LED scorra uno specifico valore di corrente. Tale valore deve essere regolato in modo stabile, in quanto la luminosità è anche funzione dell'intensità di corrente.

È quindi importante che il LED mantenga costanti i parametri che caratterizzano la zona di lavoro prescelta.

Il metodo più semplice per polarizzare correttamente un LED è quello di porlo in serie ad una resistenza ricordando che la posizione di anodo e catodo deve rispettare il verso della corrente. Il valore della resistenza viene calcolato in funzione della misura di intensità di corrente che si vuole che scorra nel LED:

$$R = \frac{V_{CC} - V_D}{I_D}$$

Con **R** = resistenza, **V_{CC}** = tensione di alimentazione, **V_D** = caduta di tensione sul diodo, **I_D** = corrente che deve scorrere nel diodo.

Se si dovesse rivelare necessario illuminare più LED contemporaneamente, è sufficiente porli in serie e considerare la somma delle cadute di tensione ai capi dei LED. Per tensioni di alimentazione molto basse, ossia dello stesso ordine di grandezza della caduta di tensione ai capi dei LED, è consigliabile mettere i LED in parallelo, per evitare dissipazioni di potenza troppo elevate.

Pertanto, **il ruolo della resistenza in serie ad un LED, è quello di limitare e mantenere al valore ottimale la corrente al fine di garantire il funzionamento del LED anche in presenza di inevitabili variazioni della tensione di alimentazione.** Nel caso di disturbi sulla alimentazione, la resistenza dissiperà la potenza in eccesso sotto forma di calore, salvaguardando la salute del LED.

2.4 Corrispondenza LED-Pixel e modello dei colori RGB

Non bisogna scordare che l'obiettivo finale del progetto LedPlate è quello di riprodurre delle immagini su uno schermo LED.

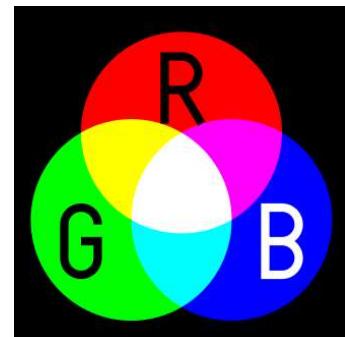
Un'immagine è composta da numerosi pixel, che possiamo considerare come l'unità minima convenzionale della superficie di un'immagine digitale. I pixel, disposti in modo da comporre una griglia fissa rettangolare, per la loro piccolezza e densità, appaiono all'occhio umano fusi in un'unica immagine. Dunque, ogni singolo pixel possiede un singolo colore.

Dalla Matematica sappiamo che esistono diversi modelli di colori^e che permettono di codificarli in forma numerica usando tre o quattro valori (o componenti cromatiche). Un modello di colore si serve di una applicazione matematica che associa ad un vettore numerico (composto da tre o quattro interi) un elemento appartenente allo spazio dei colori.

Esempio codifica RBG: [199, 21, 247] →

All'interno dello spazio dei colori di riferimento, il sottoinsieme dei colori rappresentabili con un dato modello di colore costituisce a sua volta uno spazio di colori più limitato. Questo sotto insieme è detto *gamma* e dipende dalla funzione utilizzata per il modello di colore. **Un modello dei colori, quindi, non è in grado di garantire una corrispondenza con tutti i colori esistenti nella realtà.**

Con i LED, solitamente, si utilizza il modello RGB⁸. Il modello RGB è di tipo **additivo** in quanto il colore è ottenuto sommando il rosso (Red), il verde (Green) e il blu (Blue). La somma totale dei tre colori costituisce il bianco, mentre la loro totale assenza costituisce il nero, come mostrato dall'immagine a fianco.

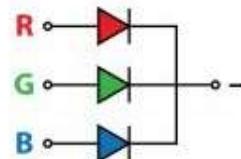


Tipicamente, in informatica, viene predisposto un intervallo numerico che va da 0 a 255 per ogni colore. Quindi, ad esempio, per indicare la totale assenza di colori e ottenere il nero, si comunicherà al computer di riprodurre il colore dato dalla somma di 0 per il rosso, 0 per il verde e 0 per il blu: [0, 0, 0] → . Per indicare il bianco, si considereranno tutti e tre gli elementi del vettore (o canali) con il valore massimo di 255.

Per riprodurre il colore di un pixel, dunque, è necessaria la presenza di almeno tre LED: uno rosso, uno verde e uno blu. In commercio esistono dei LED RBG che uniscono in

⁸ Le specifiche di questo modello sono state descritte nel 1936 dalla Commission Internationale de l'Eclairage.

modo compatto i tre LED. Questi LED possiedono quattro terminali: tre anodi (+) dei tre colori e un catodo (-).



Tuttavia, rimane scoperto il problema del controllo. È necessario, infatti, che i LED siano collegati ad un driver, affinché sia possibile comandare la riproduzione dei colori attraverso l'uso di un calcolatore.

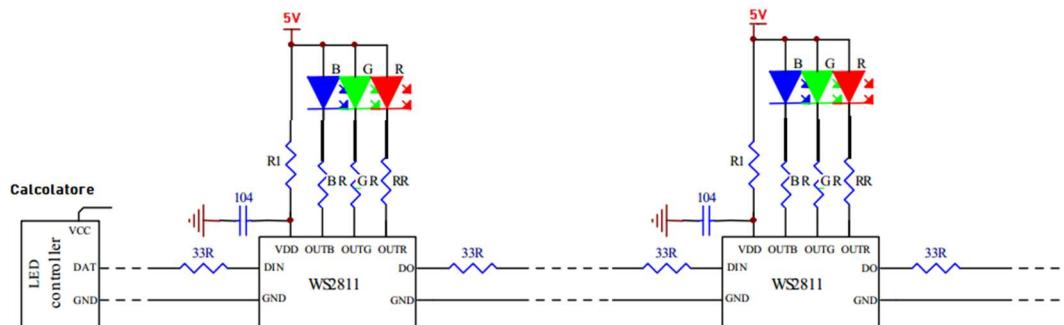
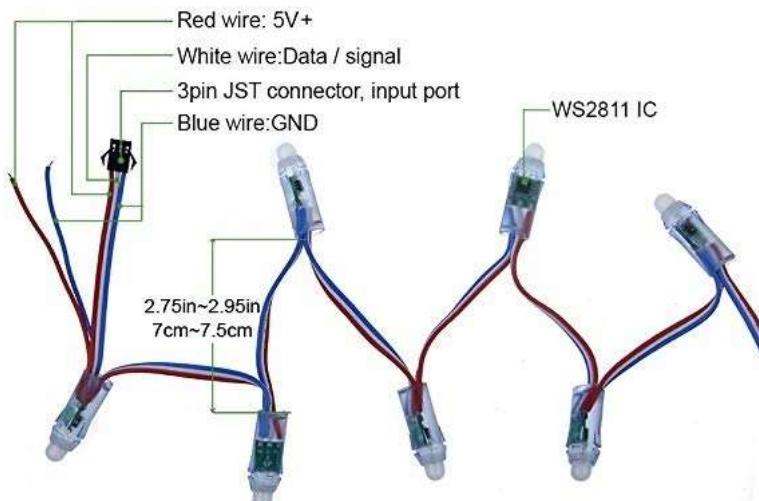
Esistono altri modelli di LED RGB già muniti di un IC⁹ driver in grado di associare ai valori digitali ricevuti in ingresso dal controller degli specifici valori di corrente per ogni LED, ottenendo, così, il colore desiderato.

Pertanto, è possibile dire che **ad ogni singolo pixel facciamo corrispondere un singolo LED RGB.**

⁹ Integrated Circuit

2.5 Modelli utilizzati: WS2811

Dalla tesi precedente abbiamo ereditato le due led-strips, le quali sono composte da LED RGB WS2811.



I LED RGB WS2811 sono impropriamente chiamati così: in realtà WS2811 è il nome del solo Circuito Integrato che opera da driver.

Il singolo dispositivo è costituito da un **driver** WS2811 e un **LED RGB** ad esso collegato.

Il driver riceve da un calcolatore il segnale dei dati e comanda l'intensità della corrente che deve scorrere negli anodi R, G e B attraverso le uscite PWM OUTB, OUTG e OUTR.

Le resistenze BR, GR e RR sono necessarie per questioni di alimentazione dei LED¹⁰.

Le due resistenze 33R e la resistenza R1, insieme al condensatore

104, hanno il compito di filtrare le alimentazioni riducendo i disturbi dati dalle interferenze in alta frequenza. In particolare, il condensatore non solo ha il compito di filtrare, ma anche di disaccoppiare il circuito del LED rendendo il segnale indipendente dalle tensioni di alimentazione dei circuiti.

Bisogna ricordare, infatti, che i LED, così come il driver, necessitano di una alimentazione in continua e che, inoltre, i segnali dei dati hanno tipicamente dimensioni dell'ordine del millivolt, mentre le alimentazioni sono dell'ordine del volt (mille volte più grandi). L'eventuale presenza di interferenze in alta frequenza sulle alimentazioni andrebbe ad impattare drasticamente sul segnale dei dati.



Tutti i suddetti componenti sono contenuti nel dispositivo illustrato in foto. Si può ben vedere il circuito integrato (il rettangolo nero) e il LED RGB (la parte sferica in plastica nell'estremità alta del dispositivo). Si intravedono anche le resistenze 33R poste in corrispondenza delle saldature dei cavi blu e dei cavi rossi (sono dei piccoli rettangoli di colore marrone/giallo).

Dal punto di vista logico, il WS2811 si comporta come un registro a scorrimento (shift register¹¹), ovvero come una catena di celle di memoria ad 1 bit interconnesse tra loro. Molto simili a dei flip-flop i quali, ad ogni impulso di clock, consentono lo scorrimento dei bit da una cella a quella immediatamente adiacente.

Nel nostro caso il WS2811 è concatenato agli altri chip appartenenti alla led-strip in modo che l'uscita dati (DO) di un WS2811 sia collegata con l'ingresso (DIN) del chip successivo.

Avendo a che fare con una catena di LED RGB, il segnale dei dati sarà costituito da un array di colori RGB. Ad ogni LED sono destinati 3 bytes di informazione (1 byte per colore).

¹⁰ Cfr. Led/Alimentazione

¹¹ https://en.wikipedia.org/wiki/Shift_register

Il primo WS2811 della catena legge i primi 3 bytes del vettore, setta i suoi output (PWM) e invia la parte rimanente del data stream al resto della catena, attraverso l'uscita DO, attraverso cui i dati possono raggiungere il WS2811 successivo.

Pertanto, il WS2811 salva lo stato attuale del proprio LED, che è stato ricevuto per mezzo del flusso di dati in uscita dal calcolatore.

I chip, però, non hanno un vero e proprio indirizzo, ma possono essere identificati sulla base della loro posizione nella catena. Di conseguenza, non è possibile comandare singolarmente un LED senza scrivere dei valori sui LED precedenti.

Per esempio, se si volesse scrivere un dato sul quinto LED della catena, sarebbe obbligatoriamente necessario scrivere qualche valore anche sui LED uno, due, tre e quattro.

È invece possibile comandare una porzione ridotta della catena, non è quindi obbligatorio accendere tutta l'intera striscia. Tramite il calcolatore è possibile inviare un array di dati di lunghezza minore rispetto alla lunghezza totale della catena, ignorando così l'esistenza dei LED fino alla fine della striscia che non coinvolti dai dati contenuti nell'array.

Questi LED RGB + driver WS2811 hanno impostato l'approccio necessario per l'intero progetto. Essendo strisce LED collegabili in serie, è difficile poter trasformare una strip in una matrice su cui riprodurre delle immagini senza dover smontare e dover modificare fisicamente il circuito dei LED.

Per poter riprodurre delle immagini, bisogna inviare alla striscia LED un array di colori. Ogni elemento dell'array deve indicare al LED che sta in posizione corrispondente all'indice di posizione di tale array con che colore accendersi.

Giunti a questo punto abbiamo compreso:

- Il funzionamento di un generico LED
- Quali specifici LED utilizzare per il nostro progetto
- Come impostare la struttura dei pannelli

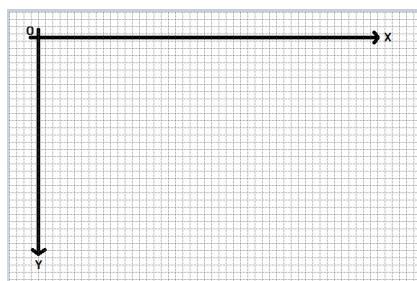
Sorge da risolvere il seguente problema: *creare una corrispondenza tra LED da accendere e pixel dell'immagine.*

3 Algoritmo di ri-ordinamento

Per risolvere il problema sopracitato, abbiamo speso la maggior parte del tempo ragionando su una possibile soluzione. Il risultato è stato un algoritmo di riordinamento. Attualmente permette di risolvere il problema dell'ordinamento in modo ottimale, grazie al lavoro di codifica svolto dal collega Andrea Patelli.

Sarebbe interessante, in futuro, riprendere in carico tale algoritmo e ottimizzarlo in ottica di risparmio di spazio di allocazione e di ottimizzazione delle performance.

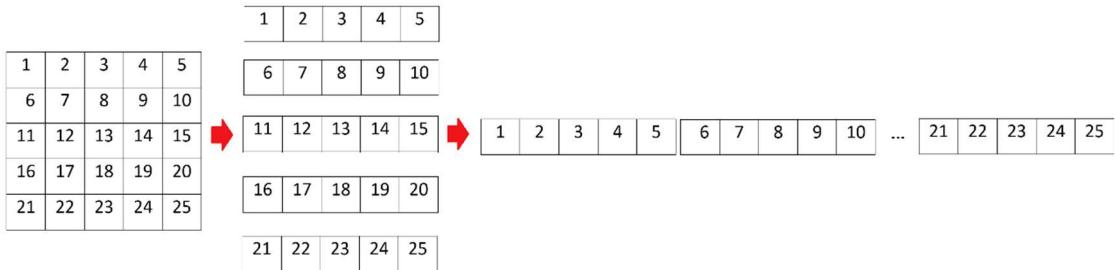
Innanzitutto, è necessario esplicitare il problema: un'immagine digitale che segue la codifica bitmap è composta da una matrice di pixel (cfr. Corrispondenza LED-Pixel e modello dei colori RGB) ordinata secondo la logica del sistema di riferimento cartesiano con origine degli assi nell'angolo alto sinistro.



Supponiamo che ogni pixel possieda un numero che indichi la sua posizione originale all'interno dell'immagine. I computer e gli schermi in genere, renderizzano le immagini secondo questa logica.

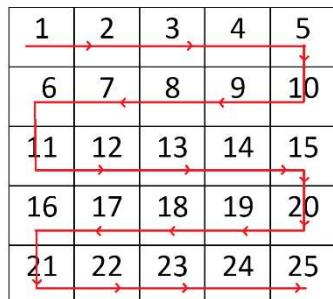
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Poiché in ingresso alla striscia LED bisogna fornire una lista di colori, è necessario convertire ogni immagine che, come abbiamo visto, è sostanzialmente una tabella, in una lista. Per farlo dividiamo l'immagine in righe e concateniamo le righe una dopo l'altra.



La tabella presa in considerazione è una 5x5, per un totale di 25 celle che corrispondono a 25 pixel.

Supponiamo di avere a disposizione una striscia composta da 25 LED. Poiché la struttura della striscia impone che per coprire uno schermo sia necessaria una configurazione a serpentina, otteniamo:

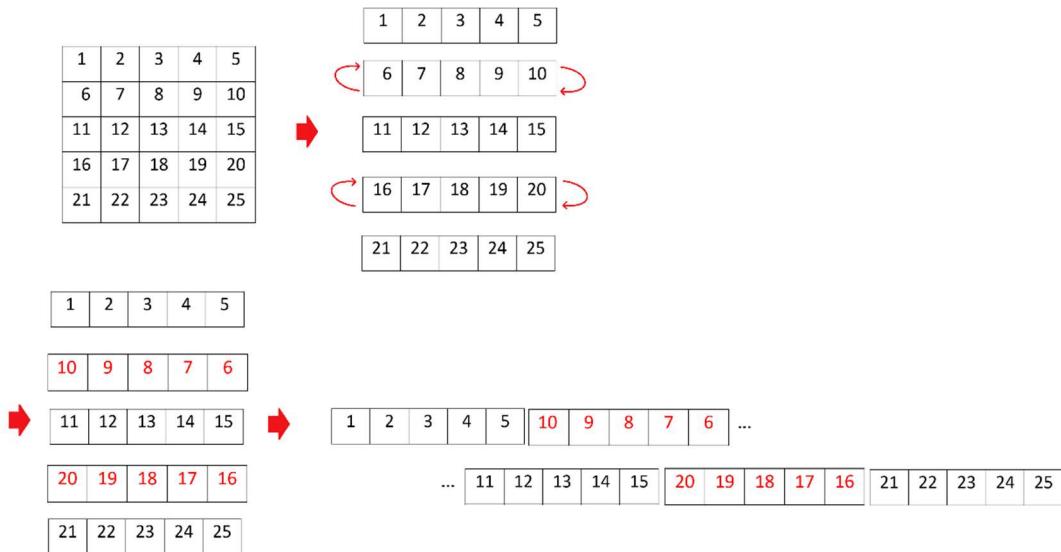


Se ogni LED si accendesse con il pixel dell'immagine corrispondente alla serpentina sovrapposta, tuttavia, si otterrebbe un'immagine avente i pixel appartenenti alle righe pari in posizione ribaltata:

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16
21	22	23	24	25

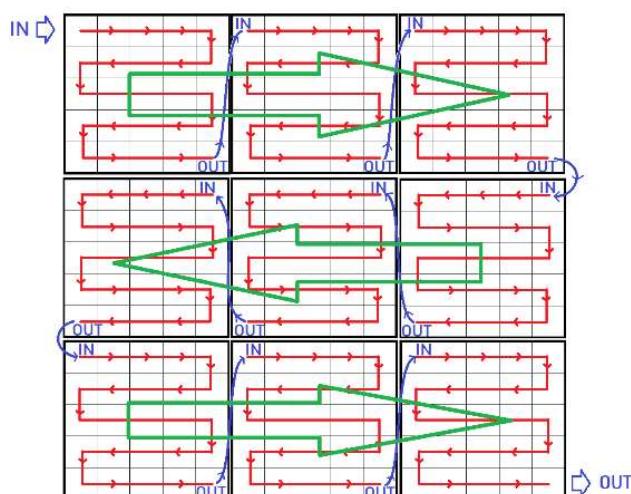
È dunque fondamentale ribaltare le righe anomale **prima** di inviarle alla striscia LED, in modo che l'immagine che dovrà apparire sui pannelli LED possa essere riprodotta nel modo corretto.

Ci si aspetta, quindi, che l'output finale da fornire in ingresso ai LED abbia subito il seguente processo di elaborazione:



Questione critica, quindi, è quella di programmare un algoritmo in grado di ribaltare le righe corrette a prescindere dalle dimensioni del pannello e dalle dimensioni dello schermo.

Uno schermo, infatti, è composto da più pannelli su cui sono montate strisce LED a configurazione di serpentina. Dunque, a causa della natura hardware delle led-strips, non solo alcune righe sono da ribaltare all'interno di un pannello, ma anche le righe di pannelli che compongono uno schermo sono da considerare in modo ribaltato:



4 Controllo dei LED

Giunti a questo punto, siamo riusciti a creare una corrispondenza corretta tra pixel e LED da accendere attraverso il nostro Algoritmo di ri-ordinamento.

Tuttavia, è importante trovare il modo di far eseguire questo algoritmo ad un calcolatore capace di far fronte alle seguenti criticità:

- *implementare l'Algoritmo di ri-ordinamento.*
- *creare un segnale di controllo da inviare ai LED.*

Tale dispositivo deve essere in grado di controllare i LED, ma anche di alimentarli.

Dunque, serve che il dispositivo di controllo sia anche dotato di uscite di potenza.

L'idea è quella di utilizzare delle schede Arduino o Raspberry.

4.1 Arduino vs Raspberry

Arduino, come già detto¹², è una piattaforma hardware dotata di un microcontrollore.

Esso possiede il cavo seriale ed esistono versioni con connettività wireless, tuttavia diventa ostico riprogrammare ogni volta la scheda, quindi eseguire cicli di riprogrammazione per ogni immagine e, allo stesso tempo, hostare un piccolo programma con accesso web. Non avendo potenti mezzi di memoria, sarebbe necessario ogni volta swappare gli elementi dell'applicazione web dalla memoria centrale alla memoria secondaria, con un pessimo risultato in termini di prestazioni e un sovraccarico del lavoro per la scheda Arduino.

Inoltre, le schede Arduino sono monoprocessore, andando incontro a funzioni bloccanti che sospenderebbero l'esecuzione dell'intero sistema.

D'altra parte, Arduino offre un IDE semplice e intuitivo. E possiede anche delle uscite di potenza oltre che le uscite di controllo dati.

¹² Cfr. "La tesi precedente come era strutturata? Cosa faceva?"

Raspberry PI è un computer single-board che quindi, come Arduino, ha dimensioni ridotte. Tutti i componenti sono montati su una scheda.

A differenza di Arduino, però, le schede Raspberry PI non bypassano il bootloader (come succede con la programmazione tramite cavo seriale ICSP di Arduino), ma sono state progettate per ospitare sistemi operativi basati su kernel Linux o RISC OS¹³.

In entrambi i casi stiamo parlando di schede economiche ma, a parità di dimensioni e prezzo simile, raspberry Pi offre la potenza di calcolo di un piccolo computer.

Tra le opzioni disponibili, avevamo già a disposizione una scheda Raspberry PI 2^f.



La scheda Raspberry PI 2 offre un processore Quad-core ARM Cortex-A53, una CPU a 900 MHz e una GPU. Una memoria centrale di 1 GB LPDDR2 e la possibilità di eseguire il boot di un Sistema Operativo direttamente da una scheda Micro SD.

Oltre ad avere la possibilità di comandare delle uscite dati e PWM, possiede connettori Ethernet, uscite HDMI e Composite RCA, output audio, connettore per un display e, ovviamente, una porta Micro USB per l'alimentazione (5V, 2A), ma la cosa più importante è la possibilità di realizzare una rete Wi-Fi locale. Quindi, Raspberry PI può sia connettersi ad internet, sia operare da server locale.

Nella seguente tabella viene messa a confronto la scheda Arduino UNO usata nel progetto Matelight con la scheda Raspberry PI 2, usata per il progetto LedPlate:

¹³ RISC OS: Reduced Instruction Set Computing Operating System è un Sistema Operativo sviluppato inizialmente dalla Acorn Computers per calcolatori ad architettura ARM. È un sistema operativo a utente singolo, supporta il multitasking di tipo cooperativo (non-preemptive). Il core del SO risiede in una memoria ROM, cosa che rende i tempi di avvio estremamente rapidi.

	Arduino UNO	Raspberry PI 2
Architettura	Atmel ATmega328	Quad-core ARM Cortex-A53
Pin	14 digitali e 6 ingressi analogici	40
Memoria	Flash 32KB, SRAM 2KB, EEPROM 1KB	1GB LPDDR2
Frequenza di clock	16 MHz	900 MHz
Connessione wireless	NO	SI

4.2 Progettazione del sistema

Vincoli:

- I LED devono riprodurre immagini, siano esse statiche o frame appartenenti a GIF o video. Il sistema, quindi, deve essere in grado di leggere i formati specifici di immagini, GIF e video, elaborarli, scomporli, sotoporli all'algoritmo di riordinamento, inviare le immagini appositamente codificate ai driver dei LED, che poi si occuperanno di fornire la corrente adatta alle tre uscite PWM per poi poter accendere correttamente i tre LED appartenenti al LED RGB.
- Deve essere possibile comandare i LED da remoto. Il dispositivo deve essere quindi collegabile alla rete, oppure deve essere in grado di fornire una propria rete a cui sia possibile collegarsi.
- Bisogna implementare un metodo in grado di gestire l'Algoritmo di riordinamento, senza bloccare l'intero funzionamento del sistema.

Scegiamo di utilizzare la scheda Raspberry PI 2, per via della sua potenza e versatilità.

Decidiamo dunque di strutturare l'intero sistema in modo che sia sempre possibile modificarlo e ampliarlo in modo semplice e veloce.

Dividiamo l'intero progetto in due parti:

- 1- Controllo dei LED
- 2- Gestione immagini

Costruiamo quindi una piccola architettura Client-Server.

4.3 Il Server

Questa parte del progetto, minuziosamente curata dal collega Simone Ciocca, prevede la realizzazione e la codifica del server.

Il server viene hostato dalla scheda Raspberry PI 2 su cui è, quindi, installato un software scritto in Python in grado di interrogare delle porte TCP per ricevere dati da un Client.

Il server, inoltre, deve anche essere anche in grado di controllare i pin a cui sono collegati i LED (sulla base dei dati ricevuti dalla rete) in modo che sia possibile alimentare correttamente i LED dello schermo formato dai pannelli ad essi collegati.

In aggiunta, il server ospiterà ed eseguirà il codice della Progressive Web App, la cui realizzazione è discussa in queste pagine. L'idea è quella di predisporre una pagina statica sempre visibile al momento della connessione con la rete privata creata dalla scheda Raspberry. La scheda, infatti, non appena viene accesa, tra le varie configurazioni di avvio accende ed inizializza una rete WiFi privata a cui è possibile accedere con qualunque dispositivo, sia esso uno smartphone o un computer.

4.4 Il Client (linea di comando)

Il sistema lato Client, invece, si occupa della gestione delle immagini e dell'invio dei dati al server.

Importante: l'intero ragionamento si basa sui concetti di LED>>pannelli>>schermo.

Assumiamo di utilizzare sempre pannelli della stessa forma e della stessa dimensione. Ciò significa avere uno schermo modulare, in quanto componibile sfruttando delle unità base: il pannello LED.

Per inizializzare il sistema, dunque, serve conoscere la risoluzione dello schermo, che viene ricavata dal numero di LED e di pannelli che lo compongono.

Nella configurazione iniziale il client chiederà le proporzioni dello schermo: quanti pannelli costituiscono lo schermo? Quanti lungo l'asse x e quanti lungo l'asse y? Ogni pannello quanti LED possiede?

Avendo a disposizione questi dati è possibile inizializzare le dimensioni dello schermo.

Le immagini (o i frame nel caso di GIF e video) dovranno avere le stesse dimensioni dello schermo. Ricordiamoci, infatti, che ad ogni pixel corrisponde uno e un solo LED¹⁴. Si rende ovviamente necessario eseguire un ridimensionamento dell'immagine.

Dunque, l'utente dovrà inserire i dati di configurazione e scegliere il file da riprodurre. Dopodiché il software si occupa del ridimensionamento sulla base delle dimensioni dello schermo. Viene poi applicato l'algoritmo di ri-ordinamento e, infine, vengono configurati i dati per l'invio al server.

La comunicazione con il server prevede l'apertura di un socket TCP attraverso cui viene inviata la stringa che rappresenta l'immagine ridimensionata e ri-ordinata.

Questa stringa prevede che ci sia un header: un valore numerico posto in testa al messaggio inviato al server, che indica che tipo di riproduzione deve predisporre il server.

Header	Messaggio che rappresenta cosa riprodurre
--------	---

Se l'header è 0, siamo nel caso di un'immagine e il messaggio è costituito da una sola stringa di dati. Se l'header è un valore diverso da 0, allora quel valore indica gli fps del video/GIF da riprodurre, il messaggio è costituito da una lista di stringhe e il sistema predispone la riproduzione di quel tipo di contenuto multimediale.

I colori vengono inviati come una stringa ricavata dalla concatenazione degli interi di R|G|B. Con R, G e B canali del modello matematico che crea corrispondenza tra il colore reale e quello codificato.

Il server, poi, esegue dei semplici shift logici bit a bit per ricavare i singoli valori dei colori e comunicarli ai LED.

La prima versione è stata sviluppata in Java con l'utilizzo di Eclipse come ambiente di sviluppo. L'interfacciamento con l'utente avviene attraverso linea di comando.

Questa versione è in grado di gestire e riprodurre Immagini, GIF e Video.

¹⁴ Cfr. Corrispondenza led-pixel etc

L'obiettivo primario è stato quello di realizzare le funzionalità con priorità maggiore:

- Input immagini, GIF e video
- Ridimensionamento delle immagini
- Scomposizione dei frame e delle immagini in liste di colori
- Applicare l'Algoritmo di ri-ordinamento alle liste
- Invio dei dati al server.
- implementazione di un protocollo di comunicazione tra client e server.

L'intera gestione del server e del client sarebbe stata praticamente impossibile da realizzare con un Arduino UNO.

5 Interfaccia Grafica

Inizialmente abbiamo pensato di usare una GUI in Java da installare su un pc. Rimaneva scoperto il problema dei dispositivi mobili. Java è un linguaggio di programmazione tipicamente usato per il back-end, dunque, per quanto riguarda la programmazione grafica, è più obsoleto e ha meno assistenza e, soprattutto, non offre la possibilità di creare applicazioni multiplattaforma. Sarebbe sempre necessario installare la JVM sul dispositivo in questione.

Altra idea è stata quella di realizzare un'applicazione Android. Rimangono però esclusi i dispositivi iOS e i computer.

La soluzione è ricaduta su qualcosa di nuovo e con maggiore possibilità di evoluzione: realizzare una Progressive Web App.

5.1 Progressive Web App

Per capire che cosa si intende con i termini Progressive Web App (PWA), bisogna prima capire il significato di due elementi fondamentali:

che cos'è una web app?

Il web è un mix di tecnologie, dispositivi e sistemi operativi interconnessi tra loro per mostrare informazioni e dati ad un utente. Il fatto che né la sua definizione né la sua implementazione sia controllata da una singola azienda, lo rende una piattaforma su cui sviluppare software unica nel suo genere. In combinazione con la sua proprietà intrinseca di collegare attraverso link qualsiasi risorsa, è possibile cercare e condividere qualunque cosa, con chiunque e ovunque. Le applicazioni web sono quindi software in rete che possono raggiungere chiunque, ovunque e su qualsiasi dispositivo, tutto a partire da una singola porzione di codice. Le web app sono ottimizzate per essere mostrate su ogni tipo di browser e per mostrare ogni tipo di dato. Vincolo stringente è la presenza di una connessione a internet.

che cos'è un'app nativa?

Le applicazioni native sono caratterizzate da un elevato numero di funzionalità e, soprattutto, dalla loro **affidabilità**. Sono sempre presenti e funzionanti a prescindere dalla disponibilità della connessione a internet. Esse vengono lanciate dai dispositivi in modo autonomo. Possono leggere e scrivere file sul file system, possono accedere e gestire l'hardware del dispositivo su cui sono installate, gestire le connessioni via USB o bluetooth e pure interagire con i dati salvati sul dispositivo, come i contatti o gli eventi del calendario. Le applicazioni native si comportano come se facessero proprio parte del dispositivo su cui stanno funzionando.

Se pensiamo alle app native e alle web app in termini di capacità (intesa come abilità di elaborazione) e portata, le app native rappresentano il meglio in termini di capacità di elaborazione e affidabilità, mentre le web app rappresentano il meglio in termini di portata. Sorge quindi spontaneo chiedersi:

Ma le Progressive Web App come vengono catalogate?

Le PWA vengono costruite per raggiungere capacità, affidabilità e installabilità simili a quelle delle app native e contemporaneamente riuscire a raggiungere chiunque, ovunque e su qualunque dispositivo con una singola porzione di codice, proprio come le web app. Le PWA sono un ibrido tra app nativa e web app, tenendo conto dei pregi di ognuna delle due categorie: è proprio questo che si vuole trasmettere con il termine Progressive.

Google mette a disposizione numerosi strumenti per poter realizzare delle buone PWA a partire dalla documentazione sino ai tool necessari per il test di questo tipo di applicazioni web.

Prima di tutto, molto importante, fornisce una checklist di comportamenti e caratteristiche che l'applicazione deve possedere. Per vedere tale checklist è sufficiente aprire lo strumento *Audit Lighthouse* che viene messo a disposizione direttamente dal browser Google Chrome, premendo sulla voce *ispeziona*, raggiungibile cliccando il tasto destro in qualunque pagina web. Avviando l'analisi, lo *Lighthouse* genera un report contenente informazioni riguardo le performance del sito web in questione. È possibile

selezionare proprio la voce Progressive Web App e visualizzare l'elenco dei requisiti che il nostro sito ha soddisfatto e quali devono essere modificati affinché il sistema lo riconosca a tutti gli effetti come PWA. L'obiettivo è quello di adempire alle seguenti caratteristiche:



Progressive Web App

These checks validate the aspects of a Progressive Web App. [Learn more](#).

- Fast and reliable**
 - Page load is fast enough on mobile networks
 - Current page responds with a 200 when offline
 - start_url responds with a 200 when offline
- Installable**
 - Uses HTTPS
 - Registers a service worker that controls page and start_url
 - Web app manifest meets the installability requirements
- PWA Optimized**
 - Redirects HTTP traffic to HTTPS
 - Configured for a custom splash screen
 - Sets a theme color for the address bar.
 - Content is sized correctly for the viewport
 - Has a `<meta name="viewport">` tag with width or initial-scale
 - Contains some content when JavaScript is not available
 - Provides a valid `apple-touch-icon`

Additional items to manually check (3) — These checks are required by the baseline [PWA Checklist](#) but are not automatically checked by Lighthouse. They do not affect your score but it's important that you verify them manually.

5.1.1 Velocità e affidabilità

Le performance giocano un ruolo significativo per il successo di ogni esperienza online. Siti web altamente performanti ingaggiano e trattengono gli utenti meglio di quelli a basse prestazioni. Ecco perché la prima voce della checklist richiede che il caricamento della pagina sia sufficientemente rapido sulle reti mobili.

La rapidità è critica per poter far usare l'applicazione all'utente. Infatti, siccome i tempi di caricamento di una pagina vanno da uno a dieci secondi, la probabilità che un utente lasci tale pagina aumenta del 123% nei primi 10 secondi¹⁵. È dunque importante che le performance del sito siano di alto livello in ogni situazione e per ogni condizione di rete.

Le performance non si fermano al solo evento di caricamento della pagina. L'utente deve anche essere sempre a conoscenza dell'esito delle sue interazioni con la pagina web. Ecco che entra in gioco il concetto di affidabilità. Per esempio, cliccando un bottone, l'utente deve sapere se l'azione è stata registrata o meno. Lo scorrimento della pagina e le animazioni devono essere fluidi. Quindi il sistema deve rispondere sempre secondo le modalità previste in fase di realizzazione. Le performance colpiscono l'intera esperienza utente, da come esso percepisce l'applicazione, sino a come essa veramente performi.

In relazione all'affidabilità e alla necessità di comunicare all'utente l'esito delle sue azioni, è importante che il sito web risponda con un HTTP 200¹⁶ quando l'applicazione si trova offline.

5.1.2 Installabilità

Se in questo momento si prendesse in mano il proprio smartphone, lo si mettesse in modalità aereo e si provasse a lanciare una qualunque app presente sul dispositivo, nella maggior parte dei casi essa fornirebbe una robusta esperienza utente anche in modalità offline. Gli utenti, in generale, si aspettano proprio quel tipo di esperienza robusta dalle proprie app. Il web non dovrebbe essere da meno.

Le Progressive Web App, infatti, devono essere progettate con l'esperienza offline come scenario principale. Effettuare un design offline-first può decisamente migliorare le

¹⁵ "We also trained a deep neural network [...] with a large set of bounce and conversions data. The neural net, which had a 90% prediction accuracy, found that as page load time goes from one second to 10 seconds, the probability of a mobile site visitor bouncing increases 123%". (<https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/>).

¹⁶ Il protocollo HTTP prevede che vi siano delle richieste e delle risposte tra client e server. Le risposte sono strutturate secondo codici di stato enumerati da 1xx a 5xx. La risposta 200 indica una conferma: il server ha fornito correttamente il contenuto richiesto dal client.

performance di una web app, riducendo quindi il numero di richieste di rete da effettuare. Per quanto riguarda le risorse, invece, esse possono essere precaricate nella cache e poi servite direttamente dalla cache locale. Pur avendo la connessione di rete più veloce del mondo, servire i dati direttamente dalla cache locale sarà sempre il metodo più veloce.

Allora ecco che risulta fondamentale rendere le web app installabili.

Le progressive Web App, infatti, sono installabili e vivono nella schermata home dell'utente, senza la necessità di passare attraverso un app store. Per farlo, è necessario fare uso dei service workers, i quali non sono altro che l'equivalente di un processo che lavora in background.

L'*Audit Lighthouse* di Google richiede esplicitamente la presenza di un service worker che gestisca la pagina iniziale del sito.

Affinché una web app sia installabile sono richiesti altri due requisiti:

1. Una connessione che segua il protocollo HTTPS

Questo protocollo impedisce ad intrusi di manomettere o ascoltare passivamente le comunicazioni tra l'app e i suoi utenti.

2. La presenza di un file manifest scritto in json che soddisfi i requisiti di installazione.

Il web app manifest permette di controllare il comportamento e l'aspetto dell'app nel momento in cui essa viene lanciata. I browser, infatti, sono in grado di richiedere agli utenti di aggiungere la web app alla home del proprio dispositivo in modo proattivo.

Il manifest^g di un sito web deve necessariamente possedere queste proprietà:

- *short_name* o *name*.

Name rappresenta il nome della web app così come viene normalmente mostrato all'utente (ad esempio tra un elenco di altre applicazioni).

Short_name rappresenta, invece, il nome della web app che viene mostrato all'utente se non c'è abbastanza spazio per mostrare la stringa contenuta nel

campo *name* (per esempio nel caso dell'etichetta per un'icona sulla home di uno smartphone).

- Una voce *icons* che deve includere almeno un'immagine grande 192x192 px, necessaria per l'icona da mostrare nella home, e una di dimensioni 512x512 px, da mostrare nello splash screen.

Icons specifica un array di oggetti che rappresentano file immagine che possono servire all'applicazione per diversi contesti. Per esempio, possono essere usate per rappresentare la web app all'interno di un elenco di altre applicazioni. La specifica sulle dimensioni delle due immagini ha il fine di poter mostrare sempre un'icona che rappresenti l'app con un'ottima risoluzione.

- *Start_url* indica l'URL della pagina iniziale della web app. Spesso si tratta di un file *index.html* oppure di un indirizzo web. Consiste nella prima URL predefinita che deve essere caricata quando l'utente lancia la web app.
- La voce *display* deve essere settata su *standalone*.

Questo permette di rimuovere la barra di ricerca del browser e dare l'aspetto delle app native anche alle web app installate. Questo può includere che l'applicazione venga aperta in una finestra dedicata o che possieda una propria icona nell'application launcher. In questa modalità l'user agent¹⁷ esclude gli elementi dell'interfaccia utente che si occupano del controllo della navigazione, ma può includere invece altri elementi come una barra di stato.

Come esempio riporto proprio il contenuto del file *manifest.json* che ho scritto per l'applicazione LedPlate:

¹⁷ User agent: applicazione installata sul computer dell'utente che si connette ad un processo server. Esempi di user agent sono i browser web, i lettori multimediali e i programmi client.

```

{
    "name": "LedPlate App",
    "short_name": "Led Plate",
    "start_url": "index.html",
    "display": "standalone",
    "orientation": "portrait",
    "background_color": "orange",
    "theme_color": "grey",
    "description": "A simple FLoutter PWA",
    "icons": [
        {
            "src": "icons/icon-192x192.png",
            "sizes": "192x192",
            "type": "image/png"
        },
        {
            "src": "icons/icon-512x512.png",
            "sizes": "512x512",
            "type": "image/png"
        }
    ]
}

```

Le voci aggiuntive che troviamo nell'esempio sono *description*, il cui significato si spiega da solo, *orientation* che in questo caso si occupa di comunicare al browser che mostrare l'applicazione con orientamento verticale e, infine, *background_color* e *theme_color* che sono richiesti per scegliere i colori che l'applicazione deve renderizzare¹⁸.

La proprietà *icons*, come abbiamo già detto, è un array che contiene oggetti che rappresentano immagini. Nel dettaglio, è necessario fornire il percorso attraverso cui è possibile recuperare l'immagine (*src*), le dimensioni dell'immagine (*sizes*) e l'estensione dell'immagine con il tag *type*.

Per avere installabilità è anche fondamentale che la pagina indicata dal tag *start_url* risponda con un 200 quando l'app di trova offline, in modo che sia possibile visualizzare informazioni e dati anche in assenza di connessione.

¹⁸ Nella computer grafica il rendering identifica il processo di “resa”, ovvero di generazione di un’immagine a partire da una descrizione matematica di una scena, interpretata da algoritmi che definiscono il colore di ogni punto dell’immagine digitale.

È importante sottolineare che la presenza di un Web App Manifest è necessaria affinché l'app sia installabile, tuttavia non è condizione sufficiente. Infatti, deve anche essere necessariamente presente un service worker attivato in modo corretto.

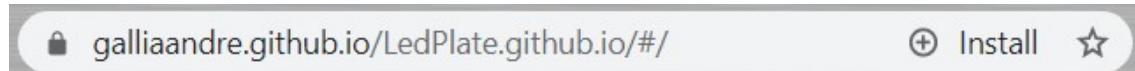
Gli utenti possono usare qualunque browser per accedere alla web app prima che venga installata.

Le PWA sono prima di tutto web app, ciò significa che hanno la necessità di funzionare attraverso qualunque browser, non solo su uno.

Bisogna sempre tenere in considerazione che l'utente che utilizzerà il sito potrà attingere da un vasto spettro di dispositivi e browser. Non bisogna tenere in considerazione solo quelli top di gamma.

Da questo consegue anche la necessità di costruire applicazioni completamente responsive, per avere un'interfaccia grafica ottimizzata per ogni dispositivo.

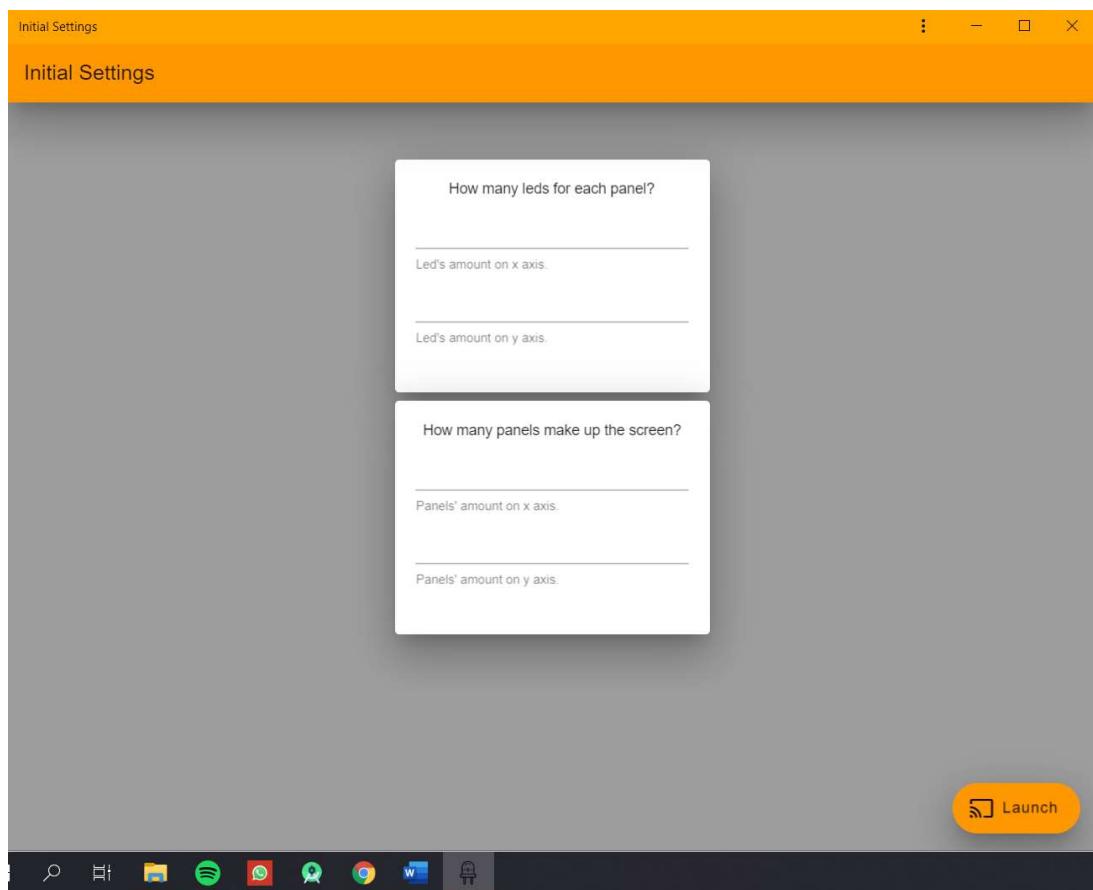
Una volta che l'applicazione web soddisfa tutti i requisiti di installabilità, allora il browser è in grado di predisporre un pulsante nella barra degli indirizzi che attiva l'installazione dell'app sul desktop del dispositivo che si sta utilizzando:



Se tale bottone viene premuto, il browser apre un dialogo di conferma per l'installazione:



All'installazione il browser separa direttamente la pagina in questione dalla viewport con gli indirizzi e crea una nuova schermata, nel nostro caso *standalone* in cui viene eseguita l'app. Ecco uno schermata:



Notare l'icona con il disegno di un LED in basso nella barra delle applicazioni: è proprio l'icona di dimensioni 512x512 px di cui si è parlato precedentemente.

L'applicazione è a tutti gli effetti installata ed indipendente dal browser.

Se invece ci si collega di nuovo alla pagina web dell'applicazione da browser e l'app è già stata installata, apparirà l'icona che ci ricorda che abbiamo già effettuato l'installazione dell'app e che ci suggerisce di visualizzare la pagina direttamente dall'app.



Nel caso in cui si stia utilizzando un dispositivo mobile, è sufficiente salvare la pagina nella schermata Home del telefono direttamente dalle opzioni elencate nel three-dots menu del browser tipicamente posizionato in alto a destra.

Alternativamente, lo sviluppatore ha la possibilità di creare e programmare un bottone interattivo direttamente nella pagina web. A livello commerciale si predilige invitare direttamente ed esplicitamente l'utente ad installare la web app sul proprio dispositivo.

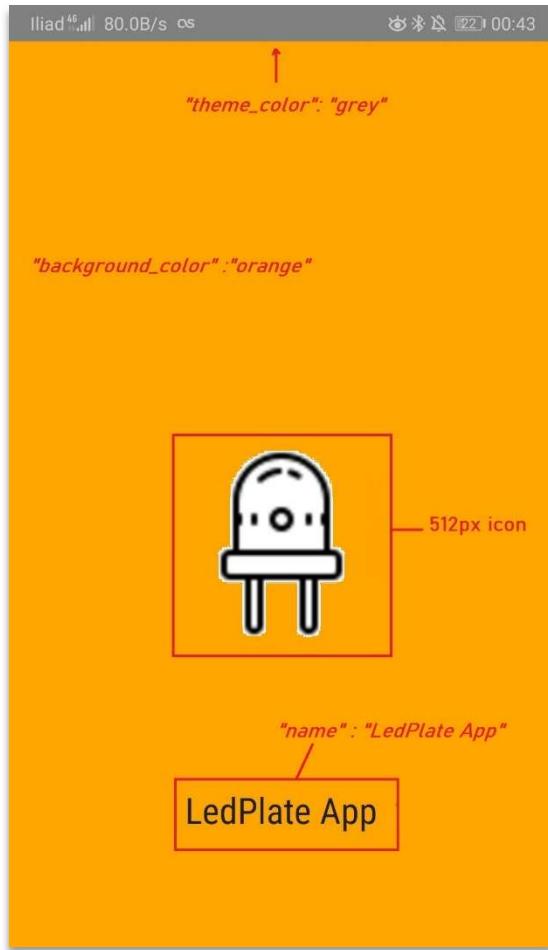
5.1.3 PWA Optimized

In questa sezione, *Audit Lighthouse* esegue il check di tutte le voci e di tutti i requisiti che rendano efficiente e funzionale la PWA in questione.

La prima voce che compare riguarda la sicurezza della connessione. Si richiede che l'applicazione reindirizzi il traffico HTTP su HTTPS. Ovvero che tutto il traffico proveniente da una sorgente non protetta venga reindirizzato su una comunicazione protetta. Per controllare che ciò avvenga, *Lighthouse* cambia l'URL della pagina in HTTP, carica la pagina e attende che il Chrome Remote Debugging Protocol¹⁹ comunichi che la pagina è sicura. Se *Lighthouse* non riceve tale comunicazione entro dieci secondi, l'audit fallisce.

La seconda voce chiede che l'app sia configurata con uno splash screen custom. Ma cos'è uno splash screen? All'apertura della PWA, il sistema Android mostra di default una schermata che rimane bianca finché l'applicazione non è pronta per essere mostrata. L'utente può vedere questa schermata bianca vuota per un massimo di 200 millisecondi. Configurando una schermata iniziale personalizzata, è possibile mostrare all'utente uno sfondo colorato con al centro l'icona della PWA, fornendo una migliore esperienza utente.

¹⁹ Strumento che Google mette a disposizione degli sviluppatori per testare e gestire i progetti che appoggiano la loro realizzazione al browser Chrome.



Uno splash screen, dunque, permette di avere una sensazione simile a quella che trasmette un'app nativa pure quando si ha a che fare con una PWA.

Come creare uno splash screen? La versione Android di Chrome mostra automaticamente uno splash screen, purchè nel Web App Manifest esistano il tag *name*, *background_color* sia impostato su un valore di colore CSS valido e che l'array *icons* contenga un'icona di dimensioni almeno 512x512 px e che sia codificata in PNG.

Poi abbiamo la richiesta di dimensionare correttamente il contenuto delle pagine rispetto alle dimensioni del viewport. Il viewport è la sezione della finestra del browser in cui è visibile in contenuto della pagina. Quando la larghezza del contenuto è più grande o più piccola della larghezza del viewport, esso potrebbe non essere renderizzato correttamente sugli schermi degli smartphone. Per esempio, se il contenuto è troppo largo, potrebbe succedere che esso venga ridimensionato in modo che possa adattarsi

alle dimensioni, rendendo però eventualmente difficile la lettura del testo. Di conseguenza è anche richiesto che nel `<head>` del file html della pagina, sia presente il meta tag con le seguenti informazioni:

```
<meta name="viewport"  
content="width=device-width, initial-scale=1.0, maximum-scale=5.1, user-scalable=yes">
```

Così facendo comunichiamo al browser le istruzioni riguardanti il dimensionamento del viewport, che nel caso specifico deve: avere larghezza uguale alla larghezza del dispositivo, un rapporto di scala iniziale pari a uno, scalabilità massima di un fattore 5.1 e con la possibilità di scalare manualmente la schermata attivata (zoom).

Nel caso in cui, per qualunque motivo, JavaScript non fosse disponibile, deve essere predisposto un contenuto di fallback che sia possibile vedere in qualunque condizione. Per quest'app è stata predisposta una semplice animazione CSS che sta a indicare il caricamento della pagina.

Infine, viene richiesto di fornire una *apple-touch.icon* valida. Per un aspetto ideale, quando gli utenti aggiungono una PWA alla home di un dispositivo iOS, il sistema richiede una specifica icona da utilizzare. Essa deve essere un'immagine PNG quadrata di dimensioni 180px o 192px.

Nell'`<head>` del file html viene dunque inserito il tag:

```
<link rel="apple-touch-icon" href="icons/icon-192x192.png">
```

5.2 Service workers

Un service worker^h è uno script che viene eseguito in background dal browser. La sua esecuzione avviene in modo separato dalla pagina Web, apreendo la porta a funzionalità aggiuntive che non hanno necessità di interazioni con un utente.

Attualmente queste funzionalità includono strumenti come le notifiche push o la sincronizzazione in background. La caratteristica principale che interessa una PWA è l'abilità di intercettare e gestire richieste di rete, inclusa la gestione programmatica di una cache di risposte.

La ragione per cui un service worker è così importante è data dal fatto che permette di realizzare applicazioni web che supportino il funzionamento di alcune sezioni della pagina anche offline, dando allo sviluppatore un controllo più completo riguardo l'esperienza utente.

Un service worker è un JavaScript Worker, oggetto che permette di creare programmi concorrenti, di conseguenza non può accedere direttamente al DOM²⁰. Tuttavia, un service worker può comunicare con le pagine di cui ha il controllo attraverso l'invio di messaggi all'interfaccia *postMessage*. Le pagine controllate poi, con le informazioni contenute in questi messaggi, possono manipolare il DOM secondo le necessità.

I service worker possono essere considerati come una proxy²¹ programmabile, permettendo allo sviluppatore di controllare direttamente il modo in cui sono gestite le richieste di rete della pagina in questione.

I service worker vengono terminati quando non sono in uso e riavviati quando sono necessari, di conseguenza non è possibile fare affidamento sul global state degli *onFetch* e *onMessage* handlers del service worker. Nel caso in cui fosse necessario salvare delle informazioni in modo persistente per poi riutilizzarle dopo il riavvio, il service worker deve poter accedere all'API *Indexed DB*ⁱ.

²⁰ Document Object Model: è un'interfaccia di programmazione per documenti HTML e XML. Esso rappresenta il documento in questione come nodi e oggetti, quindi una struttura gerarchica ad albero, a cui il programmatore può accedere per modificare struttura, stile e contenuti.

²¹ Un server proxy è un'applicazione che agisce da intermediaria per gestire le richieste dei Client cercando le risorse nei Server che soddisfino tali richieste.

L'Indexed DB è un'API di basso livello per l'archiviazione di significative quantità di dati strutturati, inclusi file o blob, lato client. Quest'API offre ricerche ed esplorazione dei dati ad alte prestazioni grazie alla sua struttura indicizzata.

Infine, i service worker fanno un uso intensivo di *promises*^j.

5.2.1 Promise

JavaScript è a single thread, ciò significa che due frammenti di script non possono essere eseguiti contemporaneamente, ma esclusivamente in modo sequenziale uno dopo l'altro. In genere, JavaScript si trova nella stessa coda di riproduzione delle immagini, dell'aggiornamento degli stili e della gestione delle azioni dell'utente (come l'evidenziare il testo o l'interagire con i controlli di una form). L'esecuzione e l'attività di uno di questi elementi ritarda gli altri elementi della coda.

Gli esseri umani sono multithread, ciò significa che per noi è possibile eseguire molte azioni contemporaneamente. L'unica azione bloccante del nostro corpo è lo starnutire, azione per cui tutte le attività in esecuzione fino a quell'istante devono essere sospese per tutta la durata dello starnuto. Uno sviluppatore non vuole scrivere codice che sia affetto da starnuti.

Tipicamente, per la programmazione concorrente a un thread, si ricorre all'uso degli eventi. Ciò significa che il flusso del programma viene determinato dalle risposte del sistema ad eventi interni o esterni come, ad esempi, l'interazione con un utente, click del mouse, la pressione di un pulsante o il messaggio di altri thread.

Tuttavia, gli eventi sono un ottimo strumento per cose che possono avvenire più volte per lo stesso oggetto. Nel caso di eventi multipli, non ci interessa sapere cosa è successo prima che fosse creato l'oggetto listener, ovvero l'oggetto in grado di intercettare un evento. Ma nel caso in cui si ha a che fare con chiamate asincrone è necessario conoscere l'eventuale successo/fallimento di una azione. Questo è quello sostanzialmente fa che una promessa (o promise). In estrema esemplificazione le promesse sono simili agli event listeners, ad eccezione di:

- Una promessa può solamente avere successo o fallire una volta sola. Non accadrà mai che essa fallisca due volte, oppure nemmeno che essa possa cambiare stato da successo a fallita o viceversa.
- Se una promessa ha avuto esito positivo o negativo e successivamente si aggiunge un callback con riferimento a quella promessa, verrà richiamato il callback corretto, anche nel caso in cui l'evento si sia verificato precedentemente.

L'utilizzo di questo strumento è molto utile in quanto non si è interessati all'esatto istante di tempo in cui qualcosa diventa disponibile, ma si è più interessati alla reazione per tale risultato.

Una promessa può essere:

- **Soddisfatta** (fulfilled) – l'azione relativa alla promessa è riuscita.
- **Respinta** (rejected) – l'azione relativa alla promessa è fallita.
- **Pendente** (pending) – la promessa non è ancora né soddisfatta né respinta.
- **Stabilita** (settled) – la promessa è in stato di soddisfatta o di respinta.

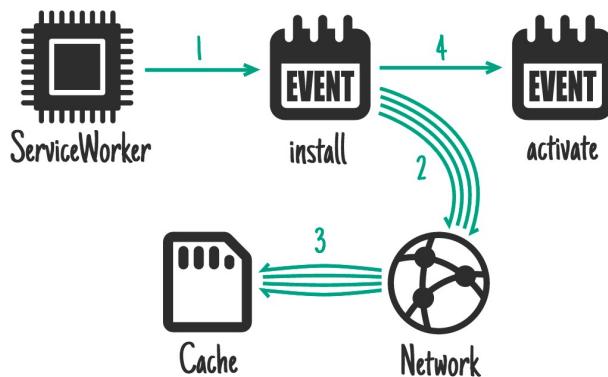
5.2.3 Service worker life cycle

Un service worker, come già detto, possiede un ciclo vitale^k che è separato da quello della pagina web. Tale ciclo vitale è composto prevalentemente da quattro eventi:

***Install* event**

Il primo evento a cui è esposto un service worker è *install*. Questo evento è innescato non appena il worker viene eseguito e viene chiamato solo una volta per ogni singolo service worker. Se lo script di un service worker viene alterato, il browser lo considera come un worker differente e gli fornirà il suo personale evento *install*.

Tipicamente l'evento *install* è usato per salvare nella cache tutto ciò che è necessario all'app per funzionare.



Activate event

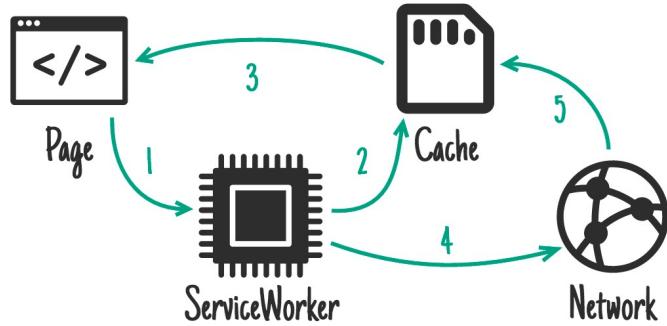
Il service worker rileva un evento *activate* ogni volta che esso viene richiesto. Lo scopo principale dell'evento *activate* è di configurare il comportamento del service worker, ripulire ogni risorsa lasciata indietro dalle esecuzioni precedenti (ad esempio vecchie cache) e rendere pronto il service worker per gestire richieste di rete (come, per esempio, l'evento *fetch* descritto qui sotto).

Fetch event

L'evento *fetch* permette al service worker di intercettare e gestire ogni richiesta di rete. Esso può connettersi alla rete per recuperare una risorsa, prenderla direttamente dalla propria cache, generare una risposta random, o una innumerevole quantità di altre differenti opzioni. Google fornisce una pagina web dedicata alle **strategie di caching** per la gestione delle risorse da fornire alla pagina che deve funzionare offline chiamata Offline Cookbook!.

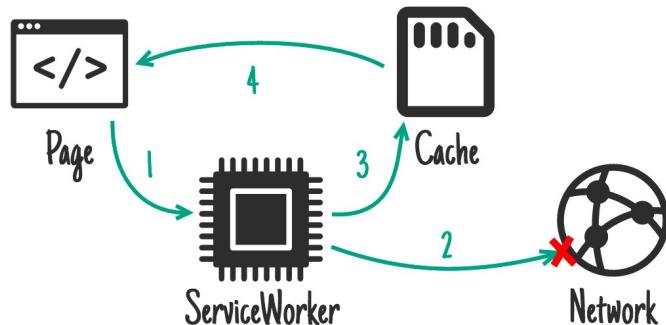
Tra le molteplici strategie disponibili ho scelto di utilizzare quella denominata *Stale-while-revalidate*. Questa strategia è ideale per quelle applicazioni in cui non è strettamente necessario avere a disposizione l'ultima versione del sistema.

Il principio di funzionamento:



Online: Se nella cache è disponibile una versione, usala, ma nel frattempo recupera la versione aggiornata per il prossimo utilizzo.

Offline: Il service worker prova ad accedere alla rete, non c'è connessione. Controlla nella cache e utilizza la versione disponibile in memoria.



Updating a service worker

Mediante l'evento *fetch* del service worker, il browser controlla la disponibilità di una nuova versione del service worker ad ogni caricamento della pagina. Se esso trova una nuova versione, la nuova versione viene scaricata e installata in background, ma senza attivarla. La nuova versione del service worker permane in uno stato di attesa fino a quando non esistono più pagine aperte con la vecchia versione del service worker. Una volta che tutte le finestre con la vecchia versione sono chiuse, il nuovo service worker è attivato e può prendere il controllo.

5.2.4 Gestione e uso di un service worker per il controllo di una pagina web

Per controllare una pagina web è necessario installare un service worker, ma per poterlo fare bisogna registrarlo nel JavaScript della pagina. La registrazione di un service worker fa sì che il browser avvii la fase di installazione in background.

Nel file *index.html*, che nel nostro caso corrisponde proprio alla *start_url* presente nel *manifest.json*, sono presenti le seguenti linee di codice, che hanno appunto il compito di registrare il service worker nel browser:

```
<script>
  'use strict';
  if ('serviceWorker' in navigator) {
    navigator.serviceWorker
      .register('./flutter_service_worker.js');
  }
</script>
```

Durante la fase di installazione, invece, è tipicamente consigliato memorizzare nella cache le principali risorse statiche necessarie per il funzionamento della pagina web. Se tutti i file vengono memorizzati correttamente, allora il service worker viene installato. Se anche solo uno qualsiasi dei file da scaricare, per qualunque ragione, non viene memorizzato correttamente, la fase di installazione non va a buon fine e il service worker non viene attivato e, quindi, non viene proprio installato.

Ciò significa che un service worker installato implica la presenza di determinate risorse statiche nella cache. Le risorse da memorizzare vengono scelte manualmente dal programmatore in fase di sviluppo. Nel file *flutter_service_worker.js* devono quindi essere presenti

- uno script che si occupi dell'installazione del worker
- delle variabili che indichino quali file memorizzare nella cache

```

var CACHE_NAME = 'ledplate-pwa-cache';
var FILES_TO_CACHE = [
  "index.html",
  "main.dart.js",
  "main.dart.js.map",
  "manifest.json",
  "assets/AssetManifest.json",
  "assets/FontManifest.json",
  "assets/fonts/MaterialIcons-Regular.ttf",
  "assets/packages/cupertino_icons/assets/CupertinoIcons.ttf",
];
/* Start the service worker and cache all of the app's content */
self.addEventListener('install', function(e) {
  console.log('[ServiceWorker] Install');
  e.waitUntil(
    caches.open(CACHE_NAME).then(function(cache) {
      return cache.addAll(FILES_TO_CACHE);
    }));
});

```

Una volta eseguita l'installazione, segue la fase di attivazione, in cui ci si occupa anche della gestione di vecchie cache. Nel nostro caso la cache viene svuotata completamente:

```

self.addEventListener('activate', (e)=>{
console.log('[ServiceWorker] Activate');

//Remove previous cached data from disk.
e.waitUntil(
  caches.keys().then((keyList) => {
    return Promise.all(keyList.map((key) => {
      if (key !== CACHE_NAME) {
        console.log('[ServiceWorker] Removing old cache', key);
        return caches.delete(key);
      }
    }));
  })
);
self.clients.claim();
});

```

Dopo la fase di attivazione, il service worker è in grado di controllare tutte le pagine che rientrano sotto il suo dominio di gestione. Tuttavia, la pagina che ha registrato il service worker per la prima volta, non sarà effettivamente controllata fino a quando non verrà

caricata di nuovo. Una volta che il service worker ha effettivamente il controllo, risiede in uno dei due stati: o viene interrotto per risparmiare memoria o gestisce gli eventi di fetch e di messaggio che si verificano quando viene effettuata una richiesta (di rete o di messaggio) dalla pagina.

Infine, bisogna ricordare di gestire l'evento *fetch*, che è quello decisivo per il funzionamento dell'applicazione web anche offline.

Nella porzione di codice riportata si ha un esempio pratico di implementazione della strategia di caching stale-while-revalidate:

```
/* Serve cached content when offline or new (if there are) when online.*/
self.addEventListener('fetch', function(e) {
    console.log('[ServiceWorker] Fetch', e.request.url);

    e.respondWith(
        caches.open('CACHE_NAME').then(function(cache) {
            return cache.match(e.request).then(
                function(response) {
                    var fetchPromise = fetch(e.request).then(function(networkResponse) {

                        cache.put(e.request, networkResponse.clone());
                        return networkResponse;
                    })
                    return response || fetchPromise;
                })
            )));
});
```

Per soddisfare la voce *You need to use HTTPS* della sezione Installabilità della checklist per le PWA, è necessario che il service worker venga registrato solo in presenza di pagine servite attraverso il protocollo HTTPS, in questo modo si sa che il service worker ricevuto dal browser non è stato manomesso durante il suo viaggio attraverso la rete.

Un ottimo strumento per testare, sviluppare ed hostare PWA sono le GitHub Pages.

Ho infatti creato una repository appositamente sia per salvare il codice prodotto per l'intera applicazione, sia per avere una demo funzionante da testare.

La demo della PWA LedPlate è online e pubblicamente visibile all'indirizzo:

<https://galliaandre.github.io/LedPlate.github.io/>

5.3 Flutter

Flutter è un SDK rilasciato ufficialmente da Google nel 2018. Esso permette la creazione di app ad alte prestazioni per iOS, Android, web e desktop partendo direttamente da un solo codice. L'obiettivo di questo SDK è quello di permettere agli sviluppatori di rilasciare applicazioni performanti e che trasmettano naturalezza di esecuzione in tutte le diverse piattaforme coinvolte. Permette quindi di creare app native senza la necessità di codificare una applicazione per ogni singola tipologia di applicazione. Vengono mantenuti, infatti, tutti i comportamenti caratteristici di ogni dispositivo quali gli effetti dello scorrimento delle pagine, la tipografia o le icone caratteristiche.

Le applicazioni sono scritte in Dart, linguaggio di programmazione rilasciato da Google nel 2011, che permette di sfruttare la programmazione orientata agli oggetti per creare interfacce grafiche multipiattaforma, efficienti ed affidabili.

Tra i numerosi punti di forza di Flutter troviamo la possibilità di:

- Essere altamente produttivi
 - Sviluppando per iOS e Android con un solo codice
 - Fare di più con meno codice, anche se il codice è destinato ad un solo sistema operativo, mediante l'approccio dichiarativo di Dart
 - Sfruttando la possibilità di prototipizzare ed iterare facilmente si riescono a sperimentare le modifiche apportate al codice ricaricandolo direttamente durante l'esecuzione dell'applicazione (con l'*hot reload*).
- Creare interfacce grafiche altamente customizzate
 - Grazie ad un ricco set di widget di Material Design e Cupertino (iOS-like) creati utilizzando il framework di Flutter.
 - Oppure realizzare personalmente la grafica senza essere vincolati dalle limitazioni imposta dai widget OEM²².

²² OEM: Original Equipment Manufacturer, azienda che realizza un'apparecchiatura che poi installata in un prodotto finito, sul quale il costruttore finale appone il proprio marchio, utilizzando integralmente o quasi componenti prodotti da fornitori (chiamati OEM).

Flutter include un framework in stile React²³, un motore di rendering 2D, widget precostruiti e svariati strumenti di sviluppo. Questi componenti, insieme, aiutano nel design, nella costruzione, nel test e nel debug delle applicazioni.

5.3.1 Widgets

Concetto chiave di questo SDK è: “everything’s a widget”. Ed è proprio così, i widget sono i mattoni fondamentali dell’interfaccia utente di un’applicazione realizzata in Flutter. A differenza di altri frameworks che separano la view, i view controllers, i layout e altre proprietà, Flutter sfrutta un oggetto elementare consistente ed unificato: il widget.

Un widget può definire:

- Un elemento strutturale (come un bottone o un menu)
- Un elemento stilistico (come un fonte o uno schema di colori)
- L’aspetto di un layout (come il padding)
- Molto altro ancora...

I widget costituiscono una gerarchia basata sulla composizione. Ognuno di essi, nidifica all’interno, o eredita proprietà dal proprio genitore della gerarchia. Non esiste un oggetto “applicazione” creato separatamente, al contrario, esiste un widget che svolge questo ruolo: il widget radice della gerarchia.

È possibile rispondere ad eventi, come l’interazione con un utente, semplicemente dicendo al framework di sostituire un widget appartenente alla gerarchia con un altro widget. Il framework confronta il nuovo widget con quello vecchio e aggiorna efficientemente l’interfaccia utente.

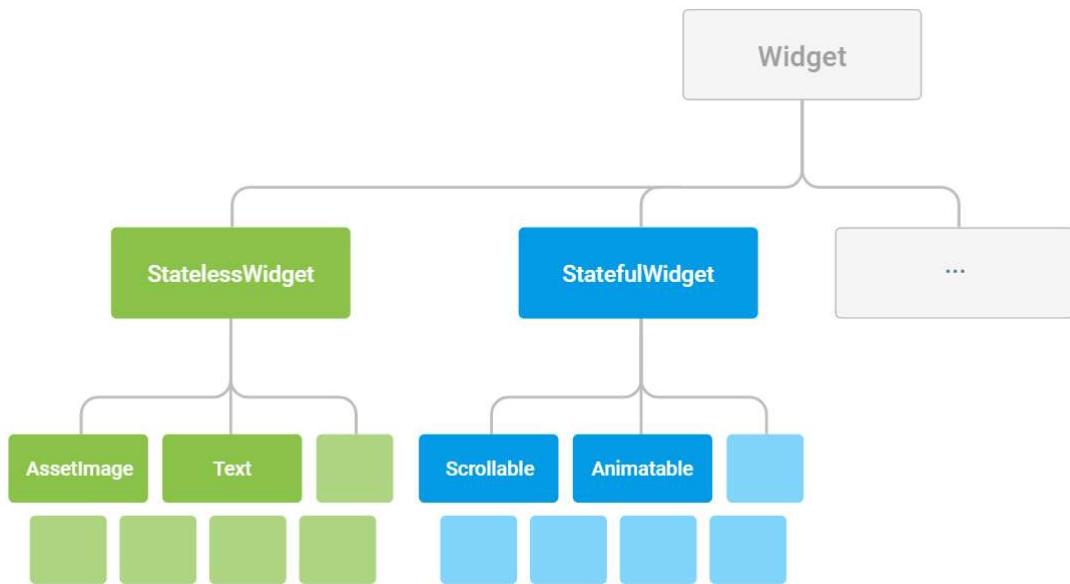
I widget, a loro volta, sono spesso composti da piccoli widget in grado di soddisfare solo singole funzionalità, che si combinano per produrre un widget con funzionalità superiori.

Per esempio, consideriamo un widget molto utilizzato: il *Container*. Esso è costituito da numerosi widget, ognuno dei quali è responsabile del layout, della posizione, delle dimensioni o di tutto quello che riguarda l’aspetto cromatico. Nello specifico, un

²³ React: libreria JavaScript per creare interfacce utente.

Container è costruito attraverso l'uso dei seguenti widget: *LimitedBox*, *ConstrainedBox*, *Align*, *Padding*, *DecoratedBox* e *Transform*. Invece di creare una sottoclasse di *Container*, è possibile comporre questi e/o altri widget messi a disposizione dal framework in nuovi modi, permettendo allo sviluppatore di creare degli effetti grafici personalizzati.

La gerarchia delle classi è generica e molto ampia nei contenuti per massimizzare il numero di combinazioni possibili:



Nell'immagine in cui si riporta un frammento della gerarchia dei widget, per farne capire il concetto, ho inserito StatelessWidget e StatefulWidget, che costituiscono le due principali alternative per iniziare a programmare lo scheletro dell'interfaccia di un'applicazione. Più avanti approfondirò questo argomento.

Bisogna, infine, aggiungere che è possibile controllare il *layout* di un widget non solo sfruttando le proprietà della gerarchia, ma anche componendolo con altri widget.

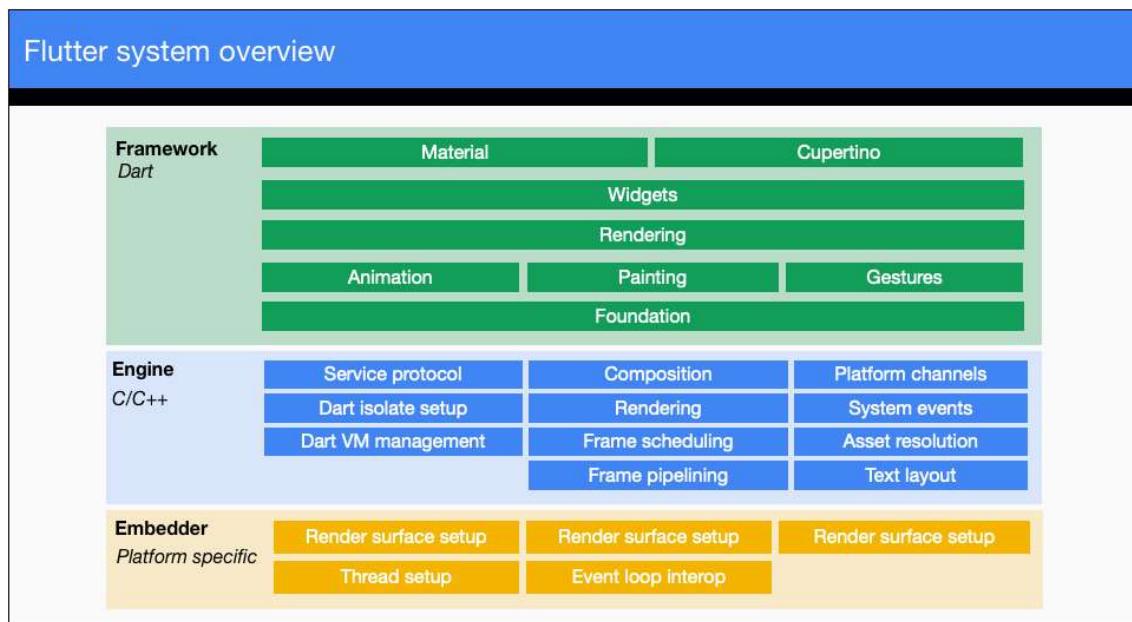
Per esempio, per far sì che un widget compaia al centro della schermata, è possibile avvolgere la sua parte di codice all'interno di un widget di tipo *Center*.

Esistono widget dedicati al padding, all'alignment, righe, colonne e griglie. Questi layout widget non possiedono una vera e propria rappresentazione visuale, bensì il loro unico scopo è quello di controllare uno o più aspetti del layout di un altro widget. Molto

spesso, infatti, per capire meglio il motivo per cui un widget venga renderizzato con determinate caratteristiche, è consigliabile non concentrarsi solo sul widget in questione, ma anche analizzare il codice dei widget che lo avvolgono o che gli sono vicini.

5.3.2 Panoramica del sistema

Il framework Flutter è organizzato seguendo la tipica struttura a strati in cui ogni livello è costruito sul livello precedente.



L'obiettivo di questo design è di aiutare lo sviluppatore a fare più cose con meno codice. Per esempio, lo strato *Material* è costruito unendo le funzionalità di widget basilari offerti dallo strato *Widget*, e lo strato *Widget* è a sua volta costruito gestendo oggetti di livello più basso offerti dallo strato *Rendering*.

Gli strati mettono a disposizione numerose opzioni per la costruzione di applicazioni. È sempre possibile scegliere un approccio personalizzato per sbloccare la piena potenza espressiva del framework, oppure usare i blocchi predefiniti dal livello dei widget, oppure fare un po' di entrambe le cose. È infatti possibile comporre i widget già pronti forniti da Flutter o creare i propri widget personalizzati utilizzando gli stessi strumenti e le tecniche utilizzate dal team di Flutter per creare il framework.

5.3.3 Costruire un widget

Le caratteristiche uniche di un widget vengono definite implementando una funzione *build()* che restituisce un albero (o gerarchia) di widget. Questo albero rappresenta la parte dell’interfaccia utente del widget in termini più concreti. Ad esempio, un widget Toolbar potrebbe avere una funzione build() che restituisce un layout orizzontale che contiene del testo e vari pulsanti. Il framework, quindi, chiede ricorsivamente a ciascuno di questi widget di eseguire il proprio metodo build() fino a quando tale processo non si esaurisce in widget completamente concreti, che il framework, quindi, ricuce in un albero.

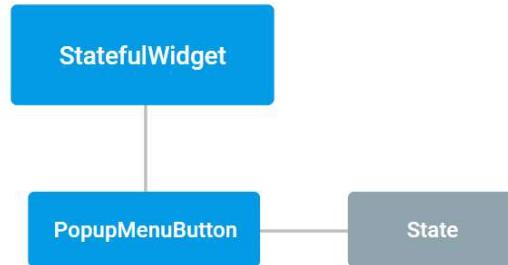
La funzione build() di un widget dovrebbe essere priva di effetti collaterali. Ogni volta che viene richiesto di essere eseguita, il widget dovrebbe restituire un nuovo albero di widget indipendentemente da ciò che il widget ha precedentemente restituito. Il framework fa il duro lavoro di **confrontare** la build precedente con la build corrente e determinare quali modifiche devono essere apportate all’interfaccia utente.

Questo confronto automatizzato è piuttosto efficace, consente la creazione di app interattive ad alte prestazioni. Il design della funzione build() semplifica di molto il codice, permettendo di concentrarsi prevalentemente sul dichiarare da cosa deve essere costituito il widget, piuttosto che sulla complessità dell’aggiornamento dell’interfaccia utente da uno stato e all’altro.

5.3.4 Gestione delle interazioni con l’utente

Se le caratteristiche di un widget devono cambiare in base alle interazioni con un utente o in base ad altri fattori, quel widget è definito *stateful*. Ad esempio, se un widget possiede un contatore che incrementa il proprio valore ogni volta che l’utente tocca un pulsante, il valore del contatore è lo stato di quel widget. Quando tale valore cambia, il widget deve essere ricostruito per aggiornare l’interfaccia utente e mostrare il valore corrente dello stato.

Questi widget sono sottoclassi di `StatefulWidget` (anziché `StatelessWidget`) e memorizzano il loro stato variabile in una sottoclassificazione di `State`.



Ogni volta che un oggetto State subisce una variazione (ad esempio, incrementando il contatore), è necessario chiamare la funzione `setState()` per segnalare al framework di aggiornare l'interfaccia grafica chiamando di nuovo il metodo `build()` dell'oggetto State.

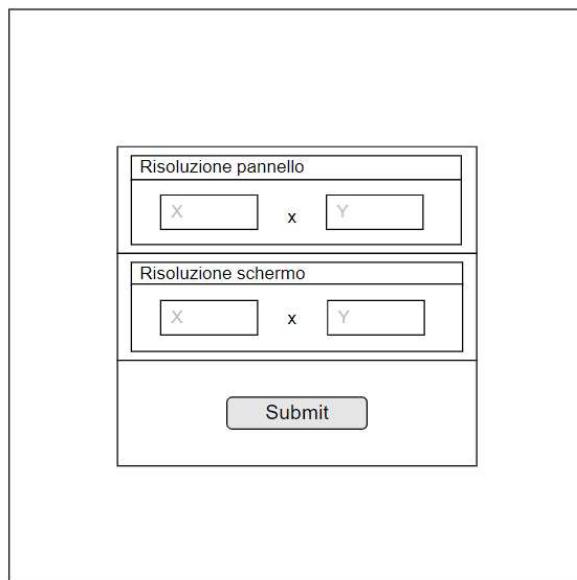
Avere separati gli stati e dai widget consente di trattare i widget *stateless* e *stateful* allo stesso modo, senza doversi preoccupare della possibilità di perdere lo stato. Invece di dover preservare un widget figlio per poter mantenere il suo stato, il widget padre è libero di creare una nuova istanza del widget figlio, senza perderne lo stato persistente (che è gestito da un oggetto State). Il framework fa tutto il lavoro per trovare e riutilizzare in modo opportuno oggetti State .

5.4 Prima Bozza UI

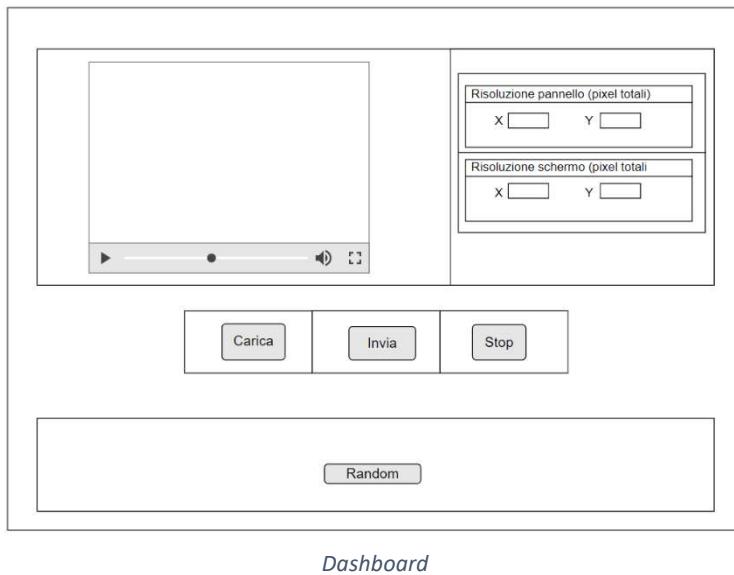
Prima di iniziare a programmare l'interfaccia grafica, ho abbozzato in linea di massima quali schermate realizzare e che aspetto dovessero avere.

Per farlo ho utilizzato un tool online: Moqups. Questo tool mette a disposizione dell'utente numerosi strumenti per la progettazione grafica. È possibile creare finestre, bottoni, inserire aree di testo. È addirittura possibile simulare il comportamento delle pagine assegnando specifici comportamenti attraverso l'uso dei collegamenti tra le pagine da assegnare ai bottoni.

Al link: <https://app.moqups.com/knF57txOaX/edit/page/ad64222d5> è possibile vedere il progetto delle bozze per la mia applicazione. Qui di seguito ho riportato le due schermate principali: la prima immagine mostra la bozza della form iniziale. Si tratta della prima schermata visibile dall'utente. In questa sezione bisogna inserire i dati necessari per l'inizializzazione del sistema, ovvero le dimensioni dei pannelli LED, utili per la gestione dei dati nell'algoritmo di ri-ordinazione dei pixel.



Schermata impostazioni iniziali



Dashboard

La seconda schermata rappresenta la bozza della dashboard. Qui sono presenti le informazioni riguardanti la struttura dei pannelli LED collegati al sistema, la preview dell'immagine da riprodurre e quattro bottoni:

- **Carica** si deve occupare di aprire un dialogo con il file system e permettere di scegliere un file multimediale da riprodurre, una volta scelto il file, l'applicazione si occupa di elaborare l'immagine e di mostrarla nel riquadro di preview.
- **Invia** si occupa di inviare i dati necessari alla riproduzione dell'immagine al server
- **Stop** blocca la riproduzione di ogni immagine mostrando una schermata nera e inviando al server il comando di bloccare la riproduzione e spegnere ogni LED.

È presente un ulteriore bottone che, se premuto, crea un'immagine assegnando ad ogni pixel un colore casuale.

5.5 L'Applicazione

Alla luce delle proprietà offerte dalle tecnologie appena viste: Flutter e Progressive Web App, ho scelto di utilizzarle per realizzare l'interfaccia grafica del nostro progetto. Questa combinazione, inoltre, permette di creare del codice utile per la costruzione di app native Android e iOS e, non solo, grazie al sistema dart2js²⁴di ottenere un file JavaScript che si comporta allo stesso modo della nostra applicazione nativa, dunque è possibile ottenere una versione web della stessa applicazione Android o iOS. Il tutto partendo da un codice sorgente unico.

Spiego la realizzazione e la codifica dell'applicazione mostrando i principali problemi che ho incontrato di volta in volta.

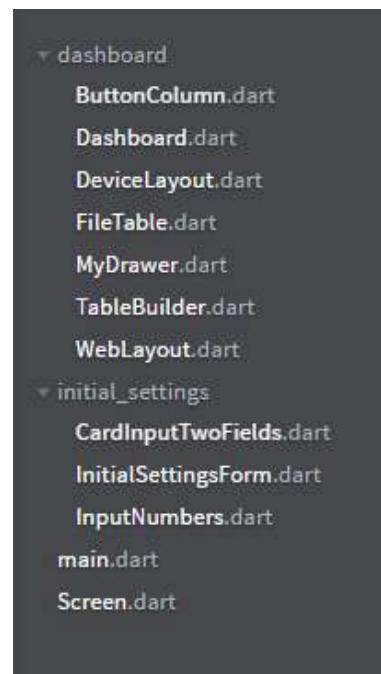
Problema 0: Inizializzazione IDE ed impostazioni.

Flutter va installato sia per linea di comando (flutter doctor aiuta molto) e poi installato come plugin di Android Studio. Il sito ufficiale di Flutter mette a disposizione documentazione e tutorial ben fatti. È quindi molto semplice poter iniziare a lavorare con questo strumento.

Problema 1: Questione più difficile è stata riuscire a capire le logiche di Flutter.

Infatti, questo framework vincola la programmazione al linguaggio Dart, che è leggermente differente dai tradizionali linguaggi di programmazione imparati in università. Tuttavia, le nozioni apprese nei vari corsi della triennale hanno aiutato nell'apprendimento di questa nuova tecnologia.

È stato altresì difficile imparare a codificare e gestire in modo efficace il funzionamento dei widget. In questa fase ho strutturato i file del progetto in package. Come si può ben vedere dall'immagine, il tutto è suddiviso nei package *dashboard*, *initial_settings*, mentre nel package principale troviamo il main dell'applicazione e la classe Screen.



²⁴ Dart2js: tool che permette di compilare del codice Dart in JavaScript distribuibile.

La logica di base che ho seguito è stata quella di suddividere i file per appartenenza alle pagine, ovvero, per esempio, tutti i file necessari per la gestione della visualizzazione e i comportamenti della pagina Dashboard, sono contenuti nel package *dashboard*.

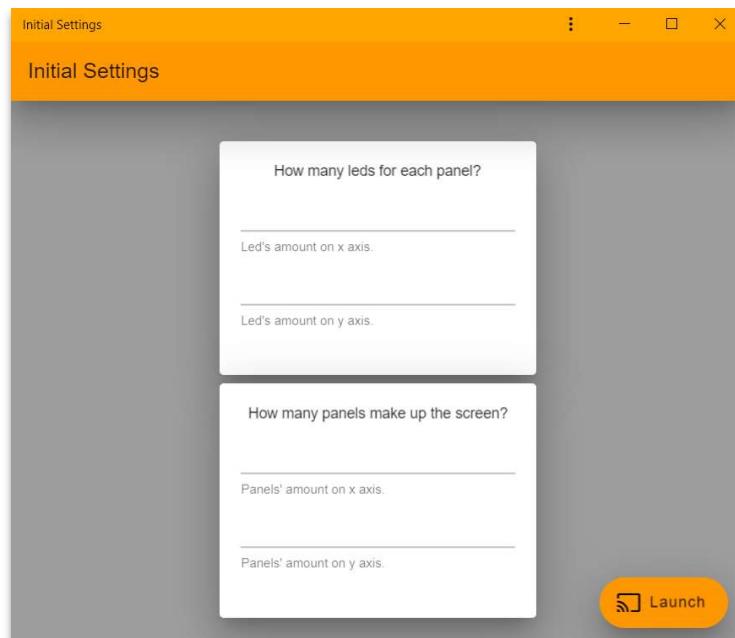
Problema 2: Navigazione tra più pagine.

Dopo aver impostato la struttura delle diverse pagine, aver codificato i widget necessari per le funzionalità basilari e seguito la bozza grafica, è necessario riuscire a collegare tra loro tali pagine e rendere l'applicazione interattiva.

Flutter permette di collegare le diverse pagine tra loro attraverso le *Route* messe a disposizione dall'oggetto *Navigator* e i suoi metodi *push()* e *pop()*.

Il collegamento tra le pagine viene tipicamente comandato mediante l'uso di un bottone. Ogni bottone possiede un attributo *onPressed* a cui viene associata la possibilità di codificare un illimitato numero di azioni.

Nel caso di LedPlate abbiamo due pagine: la schermata iniziale di configurazione (*initial_settings*) e la schermata dashboard in cui è visibile la preview dello schermo LED e i tasti di controllo. Dalla schermata delle impostazioni si passa alla dashboard premendo il bottone “Launch”:



Il bottone in questione è un *floatingActionButton*, caratteristico di molte applicazioni Android. Quando viene premuto, il sistema esegue queste azioni:

```

floatingActionButton: FloatingActionButton.extended(
    onPressed: () {
        print("Main: Floating button pressed. Initializing Dashboard... ");
        /// Validate returns true if the form is valid, or false otherwise.
        if (formKey.currentState.validate()) {
            ///unfocus the keyboard when FloatingActionButton is pressed
            SystemChannels.textInput.invokeMethod('TextInput.hide');
            screen = new Screen(
                int.parse(InitialSettingsFormState.controllerLedX.text),
                int.parse(InitialSettingsFormState.controllerLedY.text),
                int.parse(InitialSettingsFormState.controllerPanelX.text),
                int.parse(InitialSettingsFormState.controllerPanelY.text)
            );
            Navigator.push(formKey.currentContext,
                MaterialPageRoute(builder: (context) => Dashboard()));
        }
    },
    label: Text("Launch"),
    icon: Icon(Icons.cast),
    elevation: 10,
    tooltip: "Create a new LedPlate and connect to the server",
),

```

Le righe di codice evidenziato sono proprio le responsabili del collegamento tra una pagina e l'altra. Sostanzialmente stiamo comunicando al sistema di prendere in considerazione il contesto in cui stiamo eseguendo la pagina corrente, salvare lo stato (attraverso la variabile *formKey* che si occupa di gestire i dati inseriti nei campi da compilare) e di eseguire la funzione *build()* della pagina *Dashboard* e non più quella del *main*.

Non solo, alla pressione del bottone *Launch* controlliamo la correttezza dei dati inseriti nella form delle impostazioni e creiamo un nuovo oggetto di tipo *Screen* (che è la rappresentazione virtuale del nostro schermo LED).

Inoltre, per far sì che le impostazioni inserite vengano effettivamente salvate, è necessario che ogni *TextField* (area dello schermo in cui è possibile inserire del testo) sia associata ad un controller.

Problema 4: Rendere l'intera applicazione responsive.

Ogni widget possiede delle proprietà, tra cui quelle riguardanti le dimensioni. Se si vuole che il sistema sia completamente responsive, bisogna usare il widget *LayoutBuilder*. Esso

ha la caratteristica di costruire da zero l'intera struttura delle pagine gestendo i margini tenendo in considerazione le dimensioni della schermata in cui deve renderizzare l'applicazione. In questo modo è possibile avere una versione desktop e una versione app delle schermate direttamente controllando una sola funzione. LayoutBuilder, infatti, non solo richiede che gli venga indicato quale widget figlio deve eseguire, ma mette a disposizione un campo appositamente studiato per permettere allo sviluppatore di codificare metodi o funzioni specifiche per gestire direttamente il comportamento della schermata.

Problema 5: generare una preview dello schermo.

Flutter mette a disposizione il widget *Table* per rappresentare delle tabelle. L'idea è quella di realizzare una tabella che abbia le stesse dimensioni dello schermo e che ogni cella corrisponda ad un LED. Di conseguenza è come realizzare un'immagine in cui le celle della tabella sono i pixel. Tuttavia, le dimensioni della preview variano di volta in volta, in base allo schermo che abbiamo collegato al sistema. Si rende necessario codificare un tabella che venga generata in modo dinamico, ecco perché ho scritto la classe *TableBuilder*. Per creare questa classe ho utilizzato la classe della libreria *TableRow* del file *table.dart*, che ho dovuto modificare, in quanto la libreria standard non permette di effettuare particolari operazioni di cui io invece avevo necessità. Le modifiche sono dei semplici cambiamenti nella dichiarazione delle variabili. Il costruttore deve essere cambiato togliendo *const* dalla sua dichiarazione:

```
const TableRow({ this.key, this.decoration this.children })  
↓  
const TableRow({ this.key, this.decoration this.children })
```

Similmente bisogna togliere il qualificatore *final* dal campo *children* del costruttore *TableRow*:

```
final List<Widget> children; → final List<Widget> children;  
Questo per permettermi di accedere al campo children attraverso la scrittura t.children.
```

Questo passaggio è necessario per eseguire queste operazioni di creazione dinamica della preview:

```
//Rows initialization
for (int j = 0; j < screenY; j = j + screenX) {
    t = new TableRow();
    Iterable<Container> range = containers.getRange(i, i + screenX);
    //print(i.toString() + " " + (i + screenX).toString());
    t.children = range.toList();
}
rows.add(t);
}
```

È importante sapere che l'oggetto Screen possiede un attributo *ledColors* che altro non è che una lista di colori. Questa variabile statica rappresenta, in ogni istante, lo schermo che viene visualizzato nella tabella della preview.

Problema 6: integrare il codice Java in Dart.

Bisogna ricordare che la prima versione del sistema è costituita da un software Client scritto in Java. Bisogna garantire gran parte delle stesse funzionalità anche con la PWA.

Ma come? Bisogna rifare tutto?

In flutter è possibile integrare ad esistenti app Android del codice platform-specific scritto in Java. Tuttavia, implicherebbe dover strutturare l'applicazione come sola app Android, perdendo quindi tutti i vantaggi offerti dalle Progressive Web App, primo tra tutti il vantaggio di un'applicazione multipiattaforma. Il framework Flutter verrebbe sfruttato solo per codificare dei singoli moduli (in questo caso che gestiscono l'interfaccia grafica).

Decido, dunque, di riscrivere la parte di Java in Dart, per conservare l'agilità che il framework Flutter darebbe all'intero sistema.

In questo modo, in un colpo solo, posso creare sia una PWA sia delle app native installabili da store Android sia iOS. Questo perché il framework permette di rilasciare le build di Android, iOS o web separatamente, sempre a partire dallo stesso codice.

Problema 7: gestione delle librerie esterne.

Per gestire gli import di nuove librerie aggiuntive, quindi codice scritto da terzi implementabile nel proprio progetto, Flutter mette a disposizione il file *pubspec.yaml* in cui è sufficiente inserire il nome della libreria e la versione nella sezione *dependencies* del file. Nel caso in cui non si inserisca il numero della versione, il sistema ricerca automaticamente l'ultima versione stabile.

```
dependencies:  
  sdk: flutter  
  http: ^0.12.0  
  file_picker:
```

Per LedPlate ho avuto necessità di usare queste due librerie per la gestione del dialogo dell'applicazione con il file system dei dispositivi.

Problema 11: Lettura immagine da file system e assegnamento colore alle celle della preview a partire dall'immagine ridimensionata.

- Apertura della finestra di dialogo con il file system. Predisposizione del file che funge da buffer per l'upload e gestione dell'upload.

```
InputElement uploadInput = FileUploadInputElement();  
  
uploadInput.draggable = true;  
uploadInput.click();  
uploadInput.onChange.listen((e) {  
  final files = uploadInput.files;  
  if (files.length == 1) {  
    final file = files[0];  
    final reader = new FileReader();  
    reader.onLoadEnd.listen((e) {  
      handleResult(reader.result);  
    });  
    reader.readAsDataURL(file);  
  }  
});  
}
```

- Durante la lettura dell'immagine viene chiamata la funzione *handleResult* che si occupa di decodificare lo stream di dati in entrata dal dispositivo:

```

void _handleResult(Object result) {
    setState(() {
        _bytesData = Base64Decoder().convert(result.toString().split(",").last);
        _selectedFile = _bytesData;
        setScreenColors(_bytesData);
    });
}

```

Questa funzione, inoltre, si occupa di chiamare *setScreenColors()* che riceve come parametro *_bytesData*. Sostanzialmente dopo aver decodificato in bytes il file letto poco prima, lo passiamo come vettore di bytes alla funzione responsabile di settare i colori corretti alla lista *ledColors* attributo dell'oggetto Screen che rappresenta lo schermo di LED.

La funzione *setScreenColors* si occupa di:

- Creare un oggetto di tipo Image, ridimensionare l'immagine decodificata dai bytes e ridimensionare l'immagine sulla base delle dimensioni dello schermo LED

```

image = IMG.decodeImage(bytes);
resizedImage = IMG.copyWith(
    width: LedPlate.screen.screenResolutionX,
    height: LedPlate.screen.screenResolutionY);
//This list should be sent sorted to the server
dataColors = resizedImage.data;

```

- Dopo di che, prendere la lista *dataColors* dei pixel dell'immagine ridimensionata, il cui colore è codificato nel formato #AABBGGRR e creare gli oggetti Color(int r, int g, int b, int a) che vanno inseriti nella lista *ledcolors*, attributo di Screen:

```

int r, g, b, a;
for (int i = 0, s = 0; i < LedPlate.screen.ledColors.length; s++, i++) {
    r = dataColors[s] & 0xFFFFFFFF;
    g = ((dataColors[s] & 0xFFFFFFFF) >> 8);
    b = ((dataColors[s] & 0xFFFFFFFF) >> 16);
    a = ((dataColors[s] & 0xFFFFFFFF) >> 24);
    LedPlate.screen.ledColors[i] =
        Color.fromRGBO(a, r, g, b).withOpacity(1.0);
}
LedPlate.screen.sortColorsIMG();
});

```

La conversione del colore in intero è importante per la lettura, in quanto Colors vuole campi int. Quindi, è stato necessario convertire il dato contenuto nella lista *dataColors* che è di tipo Uint32, ovvero rappresenta un intero unsigned nativo a

32bit. Poiché questo intero a 32bit codifica un colore nel formato RGBA è necessario considerare solo i bit utili per poi passarli al costruttore di Color.

xxxx							
32	24	16	8	0			
A	B	G	R				

Quindi ogni campo r, g, b e a è codificato da un intero i cui 8 bit più significativi sono estratti dalla lista `dataColors`.

Per farlo, è stato necessario innanzitutto eseguire un'operazione di AND logico bit a bit, che esegue un confronto tra due variabili, dando come risultato una terza variabile che presenta un 1 in quelle posizioni in cui entrambe le variabili di partenza presentano 1 e uno 0 in tutte le altre. Quindi, passiamo da `unsigned32` a `signed64`.

In seguito a questa operazione bisogna selezionare i gruppi di bit corrispondenti ad a,r,g e b tramite l'utilizzo di shift logici. Lo shift è un'operazione che consente di spostare verso destra o verso sinistra la posizione delle cifre di un numero, espresso in questo caso in base 2, inserendo uno zero nelle posizioni lasciate libere.

Il colore in considerazione segue la codifica RGBA. La differenza con la codifica RGB, già illustrata nelle pagine precedenti, è che questo modello dei colori possiede il canale Alpha, il quale indica l'opacità che deve avere il colore. Tuttavia, i nostri WS2811 non possiedono fisicamente la capacità di variare anche l'opacità del colore emesso, di conseguenza non prendiamo in considerazione questo canale.

Il motivo per cui decodifichiamo il colore da RGBA è perché il metodo `IMG.decodeImage` codifica i singoli pixel dell'immagine letta da file proprio secondo il modello dei colori RGBA.

- La lista `ledcolors` viene letta dalla classe `TableBuilder`, che si occupa di creare la preview dello schermo, per assegnare il colore corretto ai Container che compongono la tabella. Inoltre, `ledcolors` deve essere utilizzato per controllare fisicamente i LED del pannello: esso verrà passato alla funzione che riordinerà la lista secondo l'algoritmo da noi implementato e, infine, lo spedisce al server.

Problema 13: collegamento con il server.

L'invio dei dati al server avviene attraverso la funzione asincrona `send()`:

```
void send() async {
    Response response = await post("http://127.0.0.1:3000",
        body: {'message': LedPlate.screen.colors4Server.toString()});
}
```

Questa funzione invia una lista, attributo statico della classe Screen, sotto forma stringa di interi. Questi numeri sono ricavati attraverso il metodo `sortColorsIMG()` che riordina i pixel e li salva nella lista `colors4Server`, il cui compito è proprio quello di mettere a disposizione i dati da inviare allo schermo per la riproduzione.

L'invio di tale lista, dunque, viene eseguito sotto forma di messaggio JSON attraverso il metodo `post` del protocollo HTTP.

JSON (JavaScript Object Notation) è definito come *lightweight data-interchange format*. Il che significa che è uno semplice strumento atto a gestire lo scambio di informazioni tra due o più dispositivi informatici. Questo strumento prevede che i dati siano strutturati come una collezione di coppie chiave/valore.

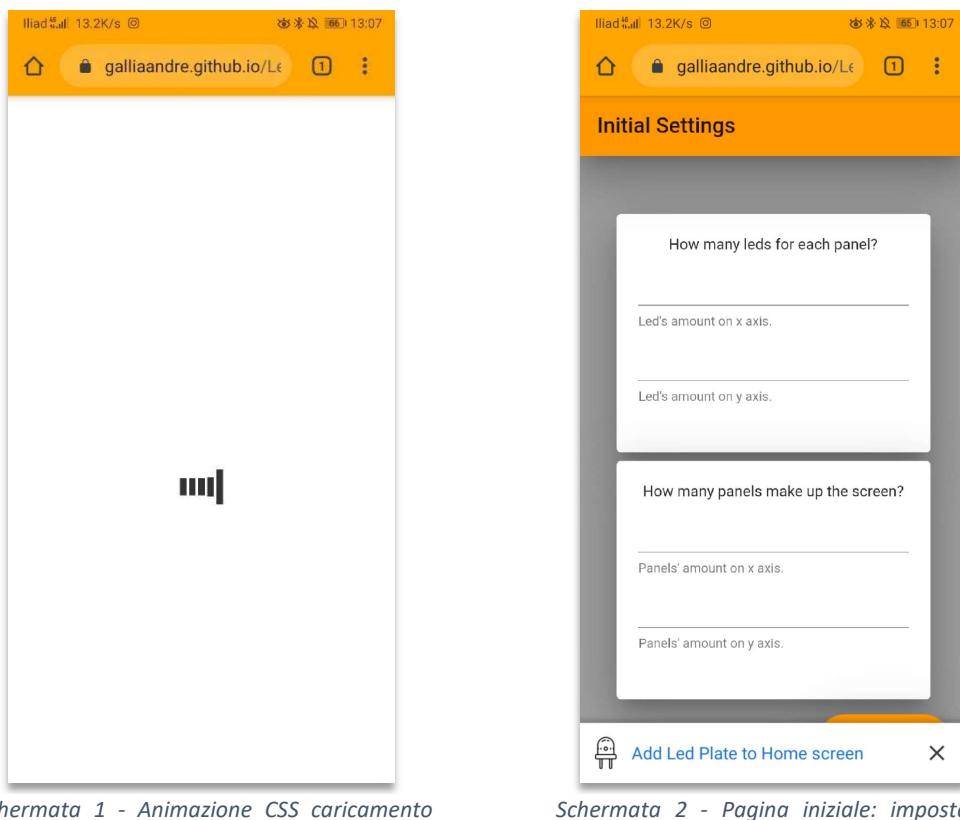
Nel nostro caso, quindi, abbiamo una chiave `message` che corrisponde ad un valore `LedPlate.screen.colors4Server.toString()`, ovvero una lista di numeri.

Una nota riguardo i dati inviati. Dal punto di vista informatico non si tratta di una lista di `int`, ma di una lista di `String`, quindi di caratteri testuali.

6 Funzionamento del sistema e schermate dell'app

In questo capitolo metto in mostra il risultato finale dell'applicazione riportando tutte le schermate che è possibile incontrare durante l'utilizzo della PWA LedPlate.

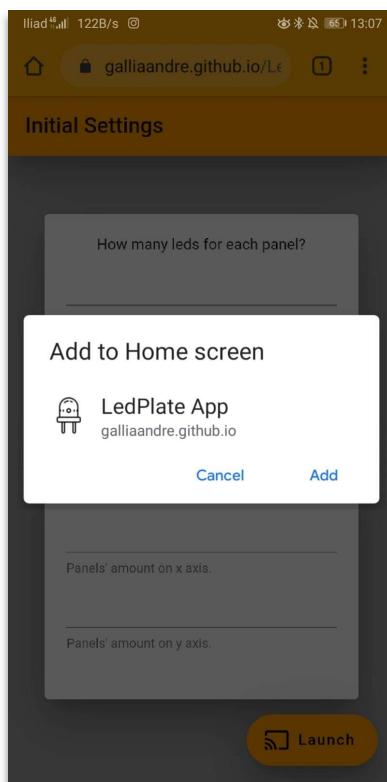
Dopo aver predisposto i pannelli LED e acceso il server, la scheda Raspberry rende disponibile una rete WiFi a cui è possibile connettersi. Una volta connessi, si deve aprire il browser dal proprio dispositivo e collegarsi all'indirizzo 10.3.141.1. A questo punto nel browser si apre la pagina principale della PWA:



L'animazione di caricamento è realizzata direttamente in CSS, e non in JavaScript, in quanto si tratta di una codifica di immagini, quindi viene caricata immediatamente dal browser. Un'animazione in JavaScript avrebbe comportato un'attesa di caricamento ed elaborazione del codice. Questa accortezza è utile per poter rispettare i vincoli richiesti dall'*Audit Lighthouse* di Google per la costruzione di una PWA.

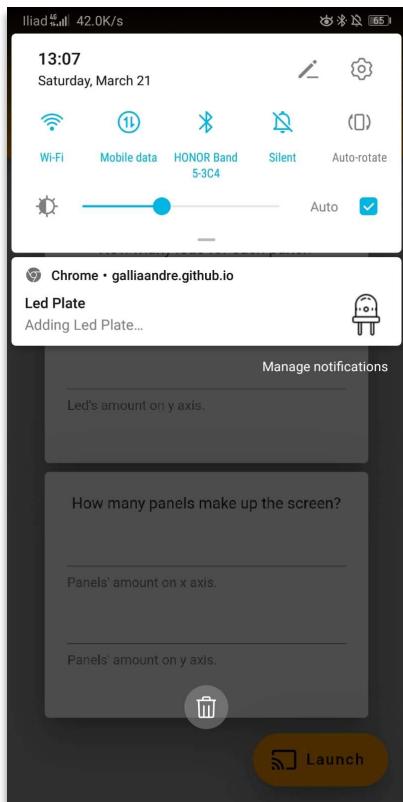
Come si può ben vedere dalla *Schermata 2*, al primo accesso il browser mostra in modo automatico il banner che suggerisce di installare la PWA nella home del proprio dispositivo. L'applicazione funziona correttamente anche da browser, senza l'obbligo di doverla installare. Tuttavia, eseguire le operazioni da browser richiede che lo sforzo di elaborazione dei dati sia tutto a carico del server. Che già si deve occupare del controllo e dell'alimentazione dei LED. Un'app installata, invece, distribuisce più efficientemente il lavoro facendo svolgere il caricamento, il ridimensionamento e la preview delle immagini direttamente al dispositivo su cui è installata l'applicazione, alleggerendo il carico di lavoro svolto dal server.

Una volta premuto il banner di installazione, il browser mostra la seguente finestra di dialogo con l'utente:

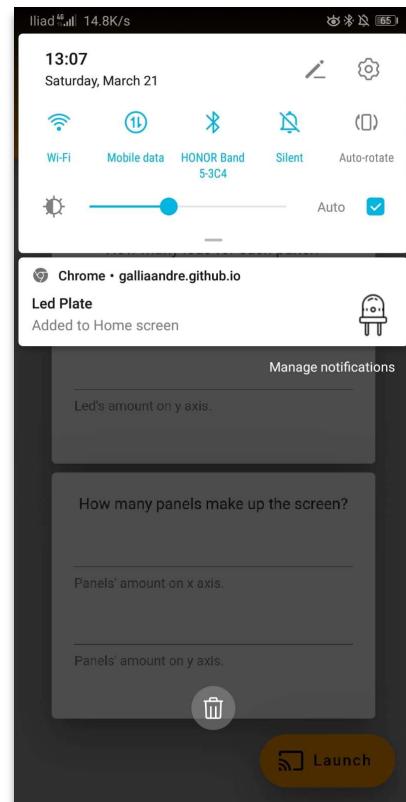


Schermata 3 - Conferma installazione app

Una volta confermata la richiesta di installazione, il browser esegue la funzione *install* del service worker appartenente alla pagina. Oltre ad inizializzare i dati necessari al funzionamento dell'applicazione , il browser comunica all'utente lo stato di avanzamento dell'installazione mostrando le informazioni nell'area delle notifiche del dispositivo:



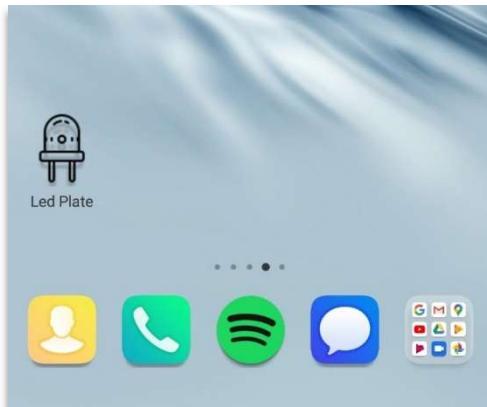
Schermata 4 - Notifica: Installazione in corso



Schermata 5 - Notifica: Installazione completata

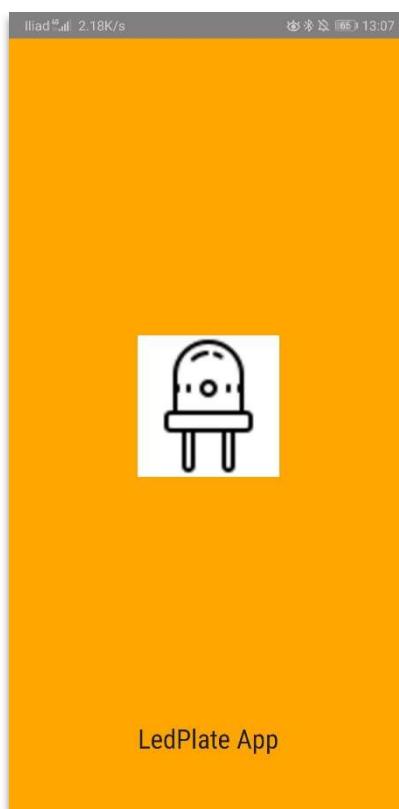
Ad installazione completata il browser lancia direttamente l'applicazione, il service worker passa in stato di *activate* ed esegue la funzione *fetch* per confrontare i dati contenuti nella cache con quelli serviti dal server.

Se il browser venisse chiuso, nella schermata home del dispositivo compare l'icona dell'applicazione. Premendola essa viene avviata, come accade con qualunque applicazione nativa.



Schermata 6 - Dettaglio icona schermata Home

Avviando la PWA dalla schermata home per prima cosa viene mostrato lo *splash screen*, poi la pagina iniziale di caricamento con l'animazione CSS ed infine la pagina iniziale delle impostazioni. A differenza del caso precedente, qui non è più presente la barra degli indirizzi: l'app è eseguita in modo standalone sia in presenza di connessione, sia in assenza di connessione.

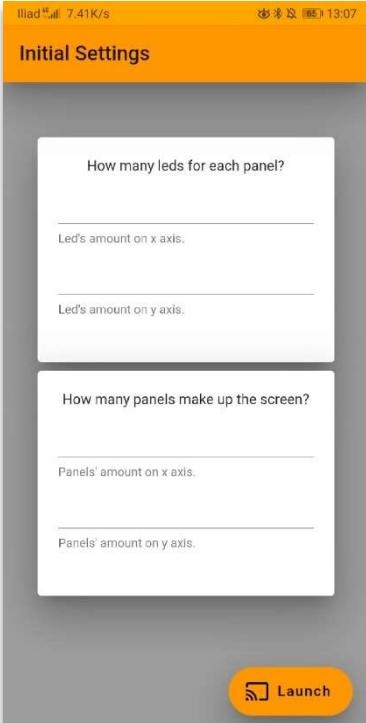


Schermata 7 - Splash screen

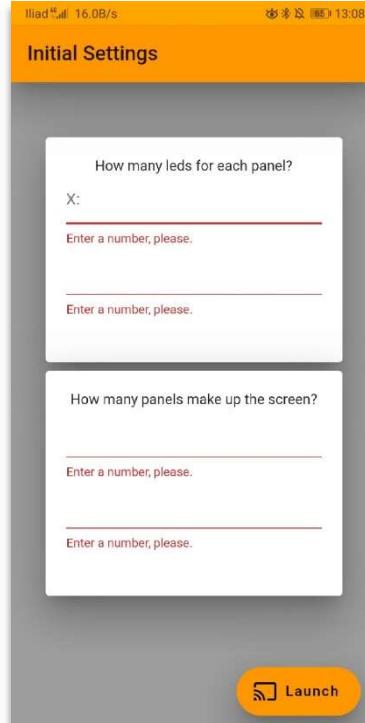
Sorge allora spontaneo chiedersi: *ma che senso ha rendere offline un'applicazione che deve necessariamente inviare dati ad un server per poter far funzionare lo schermo LED?*

Tuttavia, predisporre questo metodo di funzionamento rende possibile la realizzazione di una release più complessa. Supponiamo di avere la possibilità di aggiornare il nostro sistema e slegare l'applicazione dal server. Abbiamo in gioco tre entità: la scheda che controlla i pannelli LED, un dispositivo da cui vogliamo comandare tali pannelli e un sito web pubblicato online. L'applicazione è predisposta al collegamento con l'indirizzo TCP della scheda Raspberry che si occupa della riproduzione, quindi avremmo la possibilità di installare una PWA la cui cache viene aggiornata attraverso la funzione *fetch* ogni volta che ci si collega ad internet. In questo modo è possibile tenere sempre aggiornata l'applicazione rilasciando nuove versioni senza chiedere all'utente di eseguire alcuna operazione. Il sistema mostra in modo automatico la versione più aggiornata dell'app.

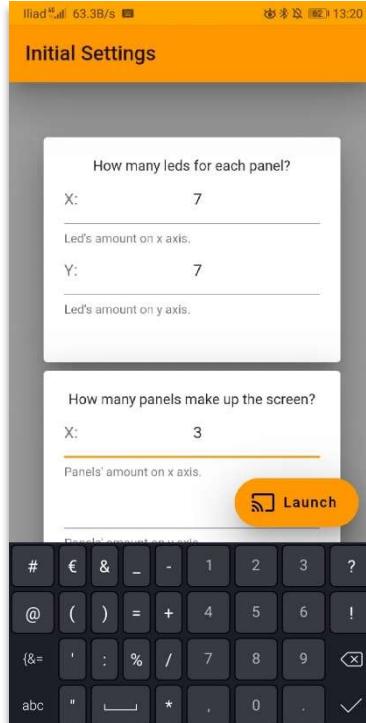
Subito dopo aver visualizzato lo splash screen, il sistema mostra la schermata di inizializzazione dei pannelli LED. Le aree di input del testo accettano solo valori numerici, in caso di campi lasciati vuoti il sistema non inizializza la dashboard e comunica all'utente di compilare i campi mancanti.



Schermata 8 - Impostazioni iniziali

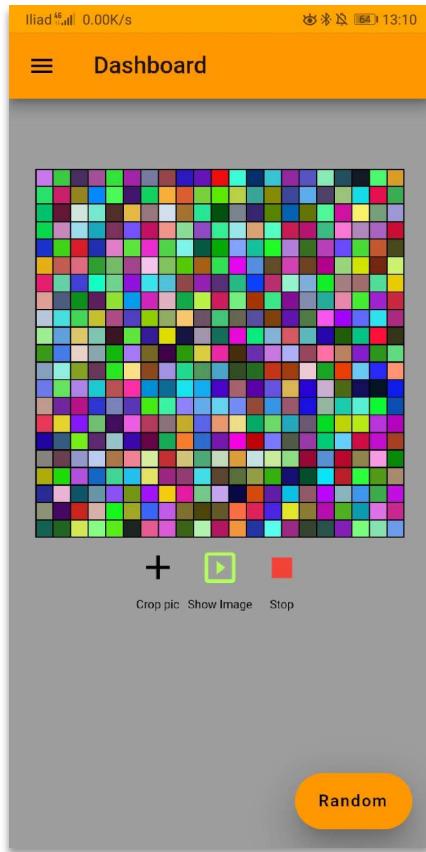


Schermata 9 - Campi non compilati

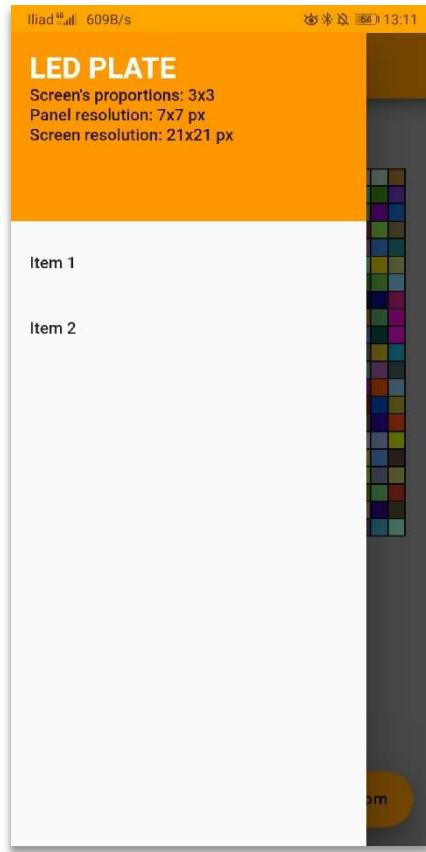


Schermata 10 - Inserimento dati

Supponendo di inizializzare uno schermo composto da 9 pannelli (3x3) di dimensioni 7x7 LED, otteniamo la seguente Dashboard:



Schermata 11 – dashboard

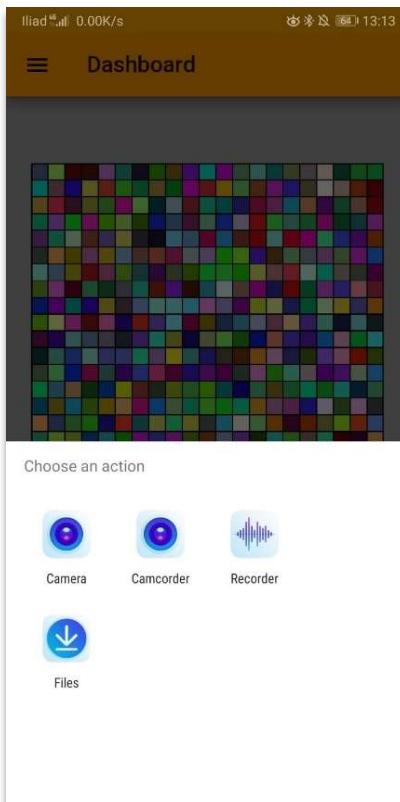


Schermata 12 - Drawer Menu

La preview viene inizializzata con dei colori casuali per ogni LED. Aprendo il drawer menu, invece, è possibile vedere le informazioni riguardanti la struttura dello schermo.

Da questa schermata è possibile eseguire le principali operazioni messe a disposizione dal sistema: upload e preview di un'immagine (bottone *Crop Pic*), invio dell'immagine allo schermo per essere riprodotta (*Show Image*), bloccare la riproduzione (*Stop*) e settare colori random alla preview.

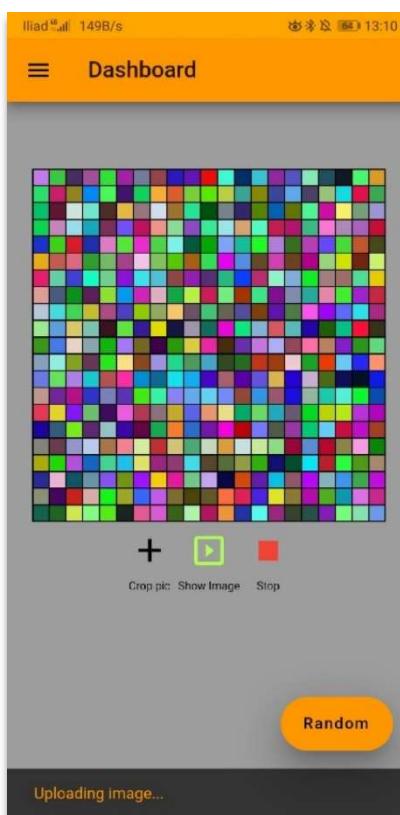
Premendo il tasto *Crop Pic* viene aperta la finestra di dialogo con il file system del dispositivo, è possibile navigare nella galleria dello smartphone oppure, da computer, in qualunque cartella disponibile. È anche possibile aprire la fotocamera e catturare un'immagine, come fatto nel caso riportato qui sotto. Durante l'elaborazione dell'immagine scelta, l'applicazione mostra uno *snackbar* (barra di testo nella porzione inferiore dello schermo) che comunica all'utente che l'elaborazione è in corso.



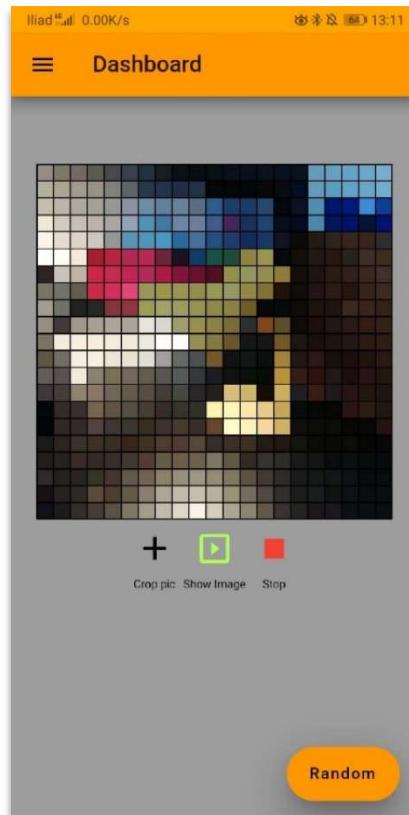
Schermata 13 - Dialogo con file system



Schermata 14 - Fotocamera



Schermata 15- Uploading image

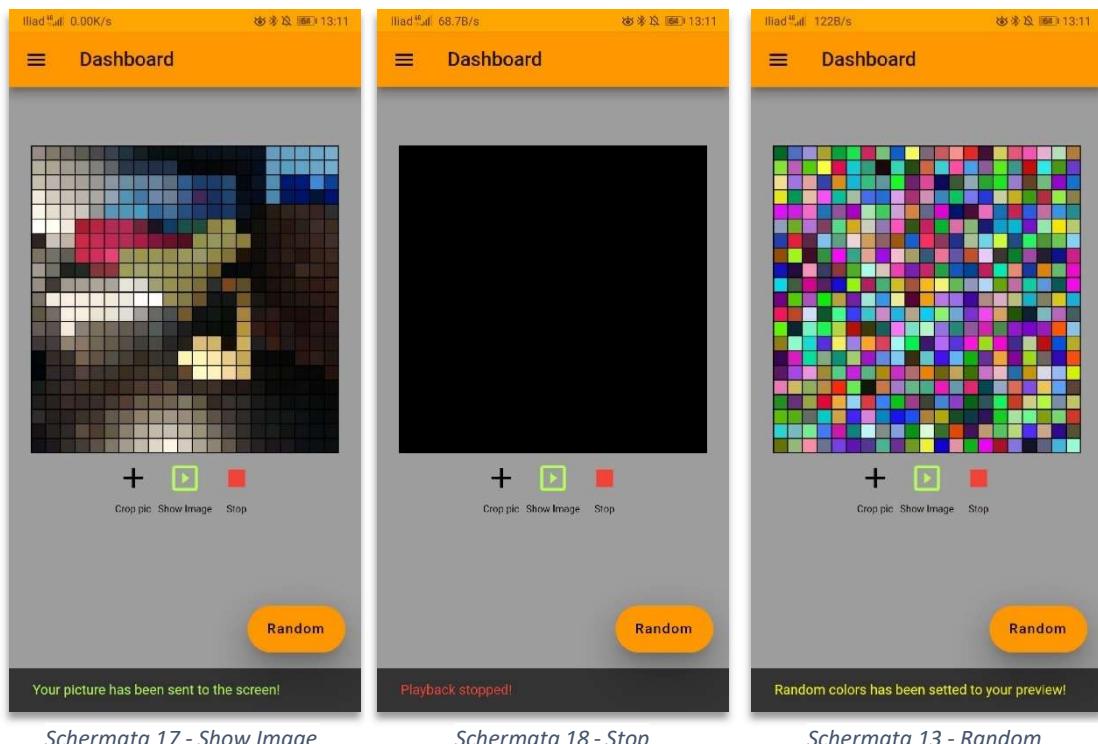


Schermata 16 - Preview dell'immagine caricata

Selezionata l'immagine, il sistema elabora il file e lo mostra ridimensionato simulando il risultato che si otterrà sullo schermo LED reale. In questa fase l'immagine da riprodurre non è ancora stata inviata allo schermo. Per farlo è necessario premere il bottone *Show Image*, che si occupa di chiamare la funzione *send()* dell'applicazione.

Infine, vediamo il comportamento dell'applicazione alla pressione degli altri bottoni:

- *Send Image* invia i dati allo schermo e mostra uno *snackbar* che comunica all'utente il successo dell'operazione.
- *Stop* setta a 0 (nel modello RGB corrisponde al nero, e il nero per un LED significa essere spento) tutti i colori da inviare allo schermo, li invia, mostra gli stessi colori nella preview e comunica all'utente l'avvenuta operazione attraverso uno *snackbar* dedicato.
- *Random* setta un colore casuale ad ogni pixel, lo mostra solo nella preview e comunica l'avvenuta operazione all'utente.



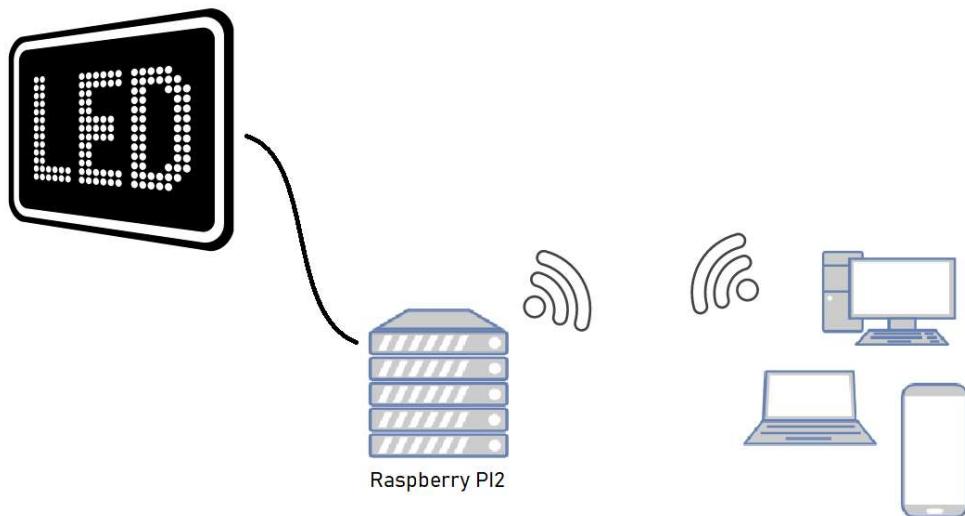
Schermata 17 - Show Image

Schermata 18 - Stop

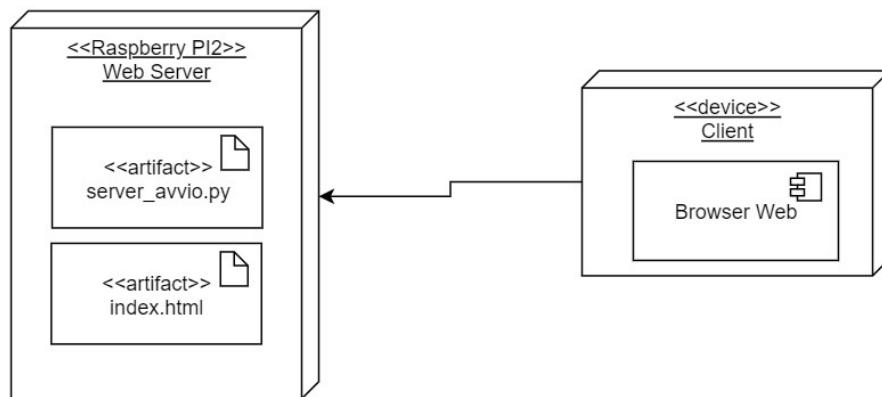
Schermata 13 - Random

7 Architettura di sistema con GUI in Flutter

La topologia del sistema è costituita dai pannelli LED, dal server hostato su un Raspberry PI2 e da un dispositivo che possa collegarsi via WiFi alla rete creata dal server.



Quello sottostante, invece, è un semplice deployment diagram che illustra sinteticamente l'architettura software del progetto.



Conclusioni

Abbiamo visto come realizzare una semplice Progressive Web App. Questo tipo di prodotto software è molto versatile e ha una grande potenzialità di sviluppo. In poco tempo questa tecnologia sta cambiando il concetto di sito web permettendo di unire le peculiari caratteristiche di tecnologie che sono consolidate da tempo alla necessità di prodotti versatili e affidabili.

Nel caso di LedPlate è stato possibile adattare questo *modus operandi* per un progetto che non è tipicamente legato al mondo web.

La mia idea è stata proprio quella di provare a trasportare una tecnologia tipicamente commerciale in un contesto di controllo elettronico, quasi industriale. La forza dell'informatica sta proprio nella possibilità di sperimentare e adattare ogni idea a nuovi contesti. Quella illustrata in questo elaborato non è sicuramente un'idea rivoluzionaria, ma ha messo in luce molti aspetti innovativi e mi ha dato la possibilità di approfondire ulteriormente campi tecnologici che ancora oggi sono in prima fase di sviluppo. Il framework Flutter ne è un esempio: lo sviluppo web di questa tecnologia è ancora in versione beta, di conseguenza, a livello commerciale, non possiede la stessa stabilità e funzionalità di tecnologie più consolidate che vengono usate da anni. Ho potuto affrontare in prima persona le problematiche conseguenti ad una tecnologia ancora in via di sviluppo. Questo mi ha allenato in qualche modo a saper modificare rapidamente un progetto e trovare soluzioni diverse da quelle iniziali, ma che potessero portare agli stessi obiettivi prefissati.

Abbiamo approfondito aspetti riguardo il funzionamento elettronico di un LED, fino al loro controllo e alla creazione di una Progressive Web App.

Ricapitolando: all'avvio del sistema la scheda che controlla i pannelli LED accende la propria rete wireless. Con un dispositivo qualunque è possibile connettersi a tale rete e accedere alla pagina web di controllo. In quel momento la pagina web lancia il file *index.html*, il quale permette di eseguire il codice JavaScript di controllo dei service worker. Quasi contemporaneamente il browser esegue il codice JavaScript, ottenuto dalla conversione attraverso il tool *dart2js*, dell'applicazione codificata in Dart attraverso

il framework Flutter. Quest'ultimo mostra la nostra vera e propria applicazione, la quale, grazie al service worker, è installabile e sempre funzionante, sia online che offline.

LedPlate è sicuramente un punto di inizio per un progetto che ha potenzialmente grandi evoluzioni future.

Un'idea potrebbe essere quella di trovare un modo di separare lo schermo LED dal controllore. Attualmente i pannelli vanno collegati manualmente via cavo ai pin della scheda Raspberry. Si potrebbe predisporre un connettore costruito *ad hoc* oppure, meglio ancora, si potrebbe pensare ad un modo per separare la connessione fisica delle due entità e sfruttare una connessione wireless. Attualmente esistono in commercio dispositivi, come Arduino LoRa, che permettono di sfruttare reti wireless a banda larga per realizzare network IoT. Sarebbe molto semplice realizzare delle interfacce che consentano di usare il sistema LedPlate con una scheda simile e poter installare a distanza uno schermo LED di qualunque dimensione.

Altro miglioramento potrebbe essere quello di migliorare la PWA trattata in questo elaborato e aggiungere nuove funzionalità, come la gestione delle GIF e dei Video. Oppure, la possibilità di programmare una coda di riproduzione e automatizzare il processo di invio delle immagini al server. Quindi avere la possibilità di elaborare, convertire i file multimediali e infine salvarli per poterli utilizzare in futuro. Io consiglio di usare l'*indexedDB API* per salvare e gestire dati persistenti con i service worker.

Si potrebbe anche avere la possibilità di comandare più schermi LED separati direttamente da una singola interfaccia.

Nel mondo dello spettacolo i famosi *ledwall* sono realizzati da dispositivi che costano migliaia di euro. Sviluppare e approfondire una tecnologia che permetta di ottenere un prodotto simile a quelli usati nel mondo dello spettacolo, ma con costi molto più contenuti, permette di rendere una tecnologia simile alla portata di tutti.

Sitografia

-
- a <https://seclab.unibg.it/tesi/matelight>
 - b <https://it.wikipedia.org/wiki/Club-Mate>; https://it.wikipedia.org/wiki/Ilex_paraguariensis
 - c <https://store.arduino.cc/arduino-uno-rev3>; http://hep.fi.infn.it/CIBER/ARDUINO/arduino_1.pdf
 - d https://cal.unibg.it/wp-content/uploads/ing_sist_controllo/Attuatori_2018.pdf slide 62
 - e https://it.wikipedia.org/wiki/Modello_di_colore
 - f <https://docs.rs-online.com/dada/0900766b8156853c.pdf>
 - g <https://developer.mozilla.org/en-US/docs/Web/Manifest/>
 - h https://developers.google.com/web/fundamentals/primers/service-workers#you_need_https
 - i https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
 - j <https://developers.google.com/web/fundamentals/primers/promises>
 - k <https://codelabs.developers.google.com/codelabs/your-first-pwapp/#5>
 - l <https://developers.google.com/web/fundamentals/instant-and-offline/offline-cookbook/>