Tirocinio Computer Sistemi

2021/2022

Matteo Pulcinelli

Made with LaTeX

Indice

1 Specifica del problema

Si richiede di realizzare un'applicazione con lettura qrcode su smartphone. Aprendo l'applicazione deve essere possibile leggere il qrcode e scrivere sul database l'informazione letta ed il timestamp del momento come "data ora inizio". Alla successiva lettura dello stesso qrcode deve riprendere lo stesso record ed inserire il timestamp del momento come "data ora fine". Una volta strutturata questa funzionalità base, deve essere possibile implementare altre caratteristiche, quali la quantità da versare, una serie di "non conformità" ai fini qualitativi, ed altre informazioni che verranno definite durante il normale percorso del progetto.

2 Studio del problema

2.1 Scelta del framework e del linguaggio

Per la scrittura dell'applicazione si è optato per il `Flutter`, framework open-source creato da Google per la creazione di interfacce native per iOS e Android. `Flutter` usa come linguaggio di programmazione il `Dart`, sviluppato anch'esso da Google che ha l'ambizione di sostituire il `Javascript`.
Il vantaggio del `Flutter` è quello di poter scrivere applicazioni in modo rapido, con ottime performance che si avvicinano molto a quelle di un'app nativa.

2.1.1 Identificativo smartphone

Bisogna trovare un sistema per identificare in modo univoco ciascun dispositivo per poter poi inserire le entry all'interno del database. La soluzione ci viene data da una libreria in flutter che oltre a fornire informazioni dettagliate del device, comprende anche un identificativo diverso per ogni dispositivo.

2.2 Comunicazione tra applicazione e server

Uno dei punti critici più importanti da risolvere è sicuramente la comunicazione tra applicazione e server. Nel nostro caso l'applicazione deve comunicare con un server PostgreSQL o SQL server.

La comunicazione non può avvenire in maniera diretta

$$\texttt{Applicazione} \longleftrightarrow \texttt{PostgreSQL/SQLserver}$$

ma deve essere presente un server in mezzo che prende le richieste HTTP POST e GET dell'applicazione e le inoltra al server PostgreSQL/SQLserver. Questo è fondamentale per non lasciare le richieste in chiaro e quindi permettere una maggiore sicurezza. L'applicazione quindi comunica con un server che ha il servizio web delle RESTful api che a sua volta comunica con il server PostgreSQL/SQL server.

$$\texttt{Applicazione} \longleftrightarrow \texttt{RESTfulApi} \longleftrightarrow \texttt{PostgreSQL/SQLserver}.$$

2.3 Server con RESTful API

Altra decisione importante ricade sulla scelta delle RESTful API. Come soluzione si è adottata la scrittura di queste utilizzando `Node.js` con framework web `Express` flessibile e leggero che garantisce ottime prestazioni e velocità di scrittura.

`Node.js` è un framework ampiamente utilizzato per lo sviluppo back-end, fa uso del linguaggio `Javascript` e risulta un'ottima soluzione per lo sviluppo e la creazione di microservizi e API.

3 Analisi e progettazione

3.1 Architettura Flutter

L'applicazione è scritta in Dart usando il framework Flutter ed è composta da una `SplashScreen`, da una schermata di `SetUp`, una di caricamento `Loading` e infine da una `HomeScreen`.

All'avvio dell'applicazione abbiamo l'animazione del logo che è contenuta nella `SplashScreen`, subito dopo viene avviata la schermata `SetUp`, questa, è dedicata all'impostazione della connessione. L'interfaccia che viene presentata all'utente è composta da due textbox le quali dovranno esser popolate con indirizzo e porta del server sul quale stanno girando le Restful API (node nel nostro caso). Per facilitare l'uso dell'applicazione, una volta che si stabilisce correttamente la connessione, vengono salvati IP e porta all'interno di un file, così che, ad nuovo avvio, non sarà più necessario inserire di nuovo i dati nei rispettivi campi. Le textbox sono poi dotate di controlli degli input con l'ausilio del RegExp. Una volta inseriti i valori corretti, si continua cliccando il tasto con la didascalia "`CONNECT`" che prima richiamerà una nuova schermata `Loading`, il cui unico scopo è far capire all'utente che sta avvenendo un caricamento, e , se tutto va a buon fine si passa alla schermata successiva: `HomeScreen`.

Questa schermata, è il cuore dell'applicazione e dà la possibilità di scannerizzare il QR. L'utente cliccherà il tasto con la dicitura "SCAN", così da aprire la fotocamera. Una volta effettuata la scansione, verranno presi i dati quali il contenuto del QR e un id univoco del dispositivo e verranno inviati al server che restituirà una risposta. Questa risposta verrà mostrata in un box situato sopra il tasto "SCAN". È presente anche un led in alto a sinistra dedicato allo stato della connessione.

In `HomeScreen`, all'interno di initState, troviamo una funzione che viene richiamata periodicamente in modo tale da verificare costantemente la connessione con il server. La funzione in questione è `checkConnection` ed è di tipo async poiché alcuni punti dell'esecuzione (segnati con il costrutto await) non possono proseguire prima del completamento di un processo. Nel nostro caso, dobbiamo aspettare la risposta del server per poter proseguire.

La funzione `LetturaQr` comprende la scansione del qrCode, trova il codice univoco del dispositivo e una volta ottenuti entrambi, invia i dati al server. Anche questa funzione deve seguire un flusso di esecuzione sensato e quindi anche qui va usato il costrutto await.

Sia la scansione del codice QR, sia l'estrapolazione del codice univoco del dispositivo, vengono risolti attraverso due librerie esistenti, rispettivamente `barcodeScan` e `deviceInfo`.

3.2 Architettura Node.js

Le Restful API come sopraccitato sono scritte in `Javascript` usando il framework `Express` e sono composte da 2 file:

- Il primo `connection.js` che dichiara un client dove troviamo tutti i dati necessari per la connessione al server postgres.
- Il secondo `api.js` dedicato all'applicazione vera e propria

Entrambi i file sono replicati due volte: ci sono due cartelle, una dedicata a postgreSQL con all'interno `api.js` e `connection.js` e l'altra dedicata a SQLserver di microsoft con gli stessi file.

3.2.1 Api.js

All'interno di questo file troviamo tutte le funzioni che riceveranno e risponderanno alle richieste da parte del server postgres/SQLserver e client. Troviamo inoltre variabili string al cui interno sono scritte le varie query col fine di non riscriverle ogni volta generando ridondanza nel codice. Attraverso la funzione listen di Express, ci mettiamo in ascolto sull'indirizzo locale specificando ip e porta. Viene poi usato il file connection.js per connettere il server node.js al server postgres/SQLserver.

La funzione `app.head` è necessaria per testare la connessione con il server postgres/SQLserver: viene inviata una query semplice che, in caso di esito negativo, restituirà un errore, questo errore verrà inoltrato al client (all'applicazione) attraverso il codice 500. Nel caso invece in cui la richiesta sia andata a buon fine, viene inviato il codice 200 che sta ad indicare un OK.

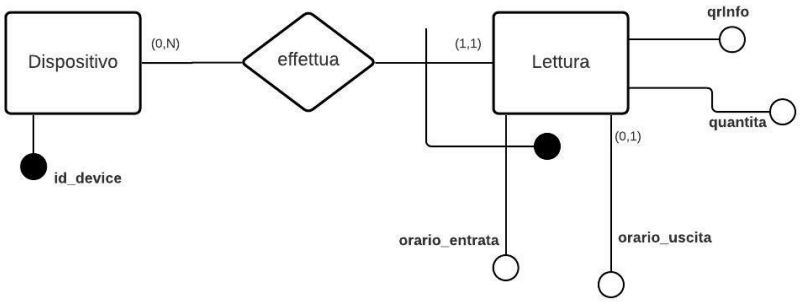
Più interessante è la funzione `app.post` che prende le richieste dei client, in base a queste richieste comunica col il server e restituisce dei valori di nuovo al client. Quando avviene il click sul bottone all'interno dell'applicazione, viene eseguita la funzione che richiamerà la query opportuna di verifica o inserimento. Finita la query verrà inoltrata una risposta al client (che rappresenta l'esito della query).

3.3 Architettura Server Postgres e SQL server

3.3.1 Modello E/R

Come primo, si è tradotta la specifica in uno schema E/R, individuando le entità principali con i vari attributi, le cardinalità e chiavi primarie e esterne.

Lo schema risultante è il seguente.



3.3.2 Concetti principali

I concetti principali individuati sono quindi qui descritti.

dispositivo:

- ***iddevice***

Ogni dispositivo può avere più letture.

lettura:

- ***iddevice***
- ***orarioentrata***
- orariouscita
- quantita
- qrInfo

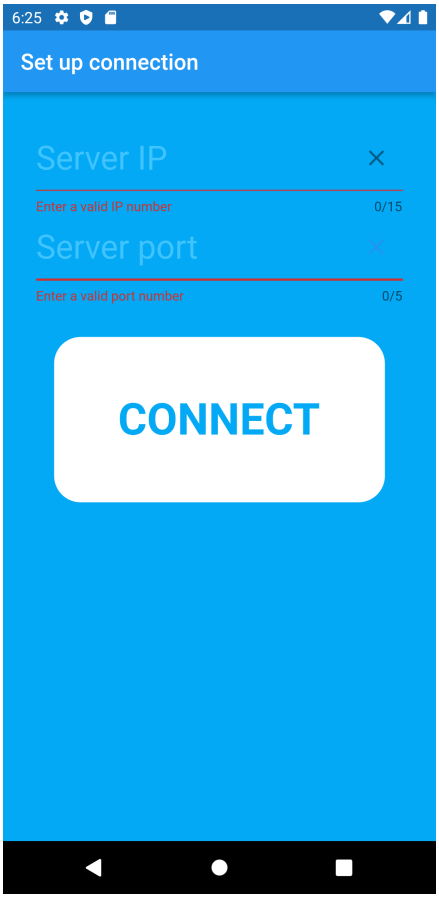
Ogni lettura deve avere uno e un solo dispositivo.

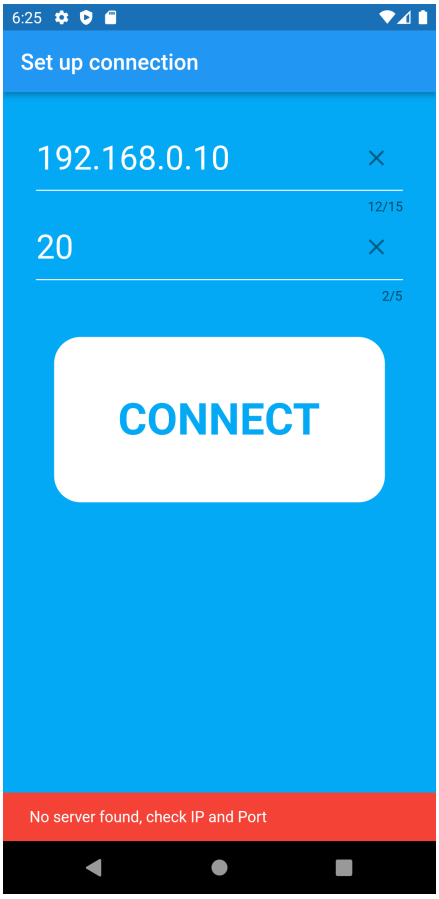
Non è stata fatta normalizzazione poichè lo schema è già normalizzato

4 Testing

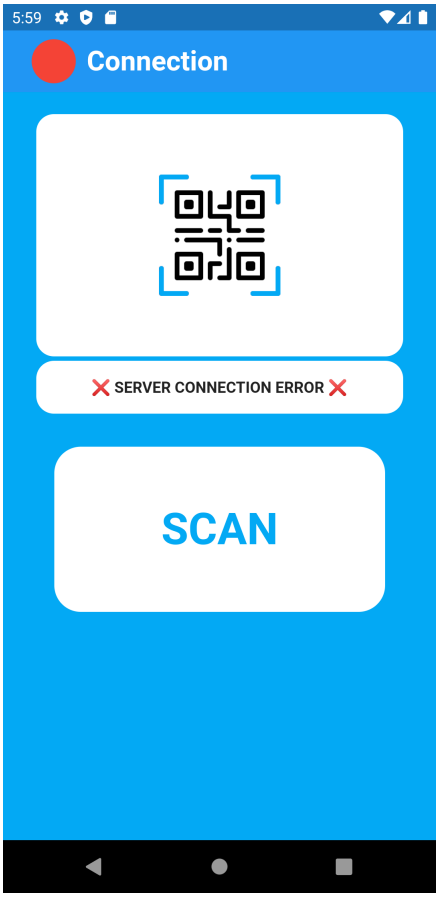
Di seguito sono riportati diversi screen.

- Caso in cui IP e porta sono assenti o puntano al server errato



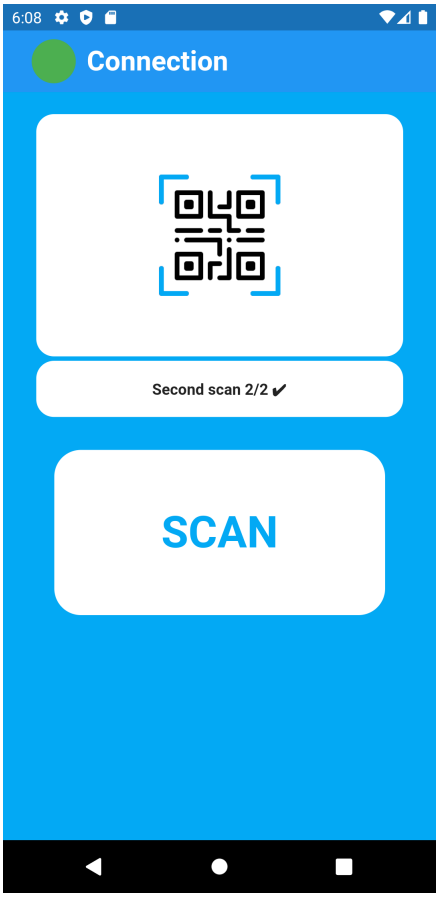


- Caso in cui la connessione al server non è andata a buon fine:



Il led rosso indica che la connessione al server è assente.

- Caso in cui viene scannerizzato per la seconda volta lo stesso qrcode:



Se interroghiamo il database postgres, dopo la prima scansione vedremo

7	8b80c95e4d5003f8	2021-10-20 17:48:33.696017	2021-10-20 17:48:52.244404	[null]	Prova scansione qr
8	8b80c95e4d5003f8	2021-10-20 17:48:58.976412	2021-10-20 17:49:02.598691	[null]	Prova scansione qr
9	8b80c95e4d5003f8	2021-10-20 17:52:15.082452	2021-11-10 12:46:59.142284	[null]	Prova scansione qr
10	8b80c95e4d5003f8	2021-11-10 12:47:04.215046	[null]	[null]	Prova scansione qr

E dopo la seconda scansione

7	8b80c95e4d5003f8	2021-10-20 17:48:33.696017	2021-10-20 17:48:52.244404	[null]	Prova scansione qr
8	8b80c95e4d5003f8	2021-10-20 17:48:58.976412	2021-10-20 17:49:02.598691	[null]	Prova scansione qr
9	8b80c95e4d5003f8	2021-10-20 17:52:15.082452	2021-11-10 12:46:59.142284	[null]	Prova scansione qr
10	8b80c95e4d5003f8	2021-11-10 12:47:04.215046	**2021-11-10 12:48:16.493961**	[null]	Prova scansione qr

Se interroghiamo il database SQLserver, dopo la prima scansione vedremo

	id_device	orario_entrata	orario_uscita	quantita	qrInfo
1	2nyuegr3726r81bg32	2022-03-05 18:06:45.120	2022-03-05 18:06:47.633	NULL	prova
2	2nyuegr3726r81bg32	2022-03-05 18:06:53.153	2022-03-05 18:06:59.463	NULL	prova
3	2nyuegr3726r81bg32	2022-03-05 18:07:00.340	2022-03-05 18:07:00.803	NULL	prova
4	2nyuegr3726r81bg32	2022-03-05 18:07:06.150	NULL	NULL	prova

E dopo la seconda scansione

	id_device	orario_entrata	orario_uscita	quantita	qrInfo
1	2nyuegr3726r81bg32	2022-03-05 18:06:45.120	2022-03-05 18:06:47.633	NULL	prova
2	2nyuegr3726r81bg32	2022-03-05 18:06:53.153	2022-03-05 18:06:59.463	NULL	prova
3	2nyuegr3726r81bg32	2022-03-05 18:07:00.340	2022-03-05 18:07:00.803	NULL	prova
4	2nyuegr3726r81bg32	2022-03-05 18:07:06.150	2022-03-05 18:08:27.597	NULL	prova

5 Link

Tutto il codice scritto è reperibile attraverso seguenti link:

- Front-End: https://github.com/Matteo-293473/qr_code_front_end
- Back-End: https://github.com/Matteo-293473/qr_code_back_end