Elaborato di
INFORMATICA e SISTEMI E RETI

Sestito Simone

Ipotesi aggiuntive

Il servizio si impone lo scopo di aiutare i commercianti a gestire l'accesso contingentato all'interno dei propri locali.
Per permettere agli utenti la più pratica consultazione dei negozi convenzionati nelle vicinanze e della loro situazione di coda, si decide di utilizzare la visualizzazione tramite **mappa**, partendo dalla posizione corrente dell'utente rilevata tramite GPS integrato nei dispositivi mobili. L'utente è bene che sia anche in grado di **cercare** i negozi per nome, visualizzando sulla mappa tutti i risultati.
Si sceglie di dare la libertà al commerciante di annullare le prenotazioni effettuate fino a quel momento.
Per una rapida consultazione e utilizzo del servizio da parte dell'utente, si ritiene necessario la possibilità dell'utente di aggiungere dei negozi ai suoi **preferiti**.

Al fine di evitare assembramenti e non procurare fastidi all'utente, si sceglie di implementare **notifiche push** per avvisare l'utente sullo stato della sua prenotazione (poche persone davanti, turno chiamato, prenotazione annullata). Stesso discorso per quanto riguarda la spesa con ritiro in negozio.

Dato che account fittizi possono portare ad un notevole peggioramento della qualità del servizio, si impone all'utente che si registra per la prima volta di procedere a **verificare il proprio indirizzo email** tramite l'apertura di un link monouso inviato contestualmente alla registrazione.

Un commerciante può gestire esattamente quando chiamare i clienti in coda virtuale, permettendogli di gestire la coda fisica fuori dal negozio o locale nel modo più efficiente.
In caso lo desideri, l'uso di questo servizio non vieta al commerciante di gestire una doppia coda virtuale e fisica.
Inoltre, può consultare le ordinazioni di spesa che ha ricevuto dai clienti e, dopo averle preparate, può contrassegnarle come tali. L'utente verrà avvisato dell'accaduto. Il pagamento della spesa avverrà in fase di ritiro, direttamente con il commerciante. Il servizio non si deve occupare di gestire questo aspetto.

Analisi funzionale

Tutti gli utenti di ogni ruolo potranno interfacciarsi con il servizio in questione mediante **l'utilizzo di una applicazione mobile** Android. Un'app, rispetto ad una piattaforma accessibile via browser, risulta più comoda a un utente tradizionale, con una maggiore facilità di accesso a funzionalità di sistema come la geolocalizzazione e le notifiche. Stesse motivazioni di praticità d'uso valgono per il commerciante. Discorso, invece, di libertà per l'amministratore, potendo gestire il servizio in mobilità.
Al fine di garantire a quante più persone possibili di utilizzare il servizio, l'app supporta da Android 5.0, risultando compatibile sul 94,1% dei dispositivi Android (fonte: Google, 05/2020). Gli aspetti di accessibilità e usabilità sono approfonditi nell'ultimo punto del presente documento.

Requisiti per l’utente

Senza accesso con un account di qualunque tipo, non è consentito l'utilizzo dell'app.
L'utente, previa registrazione, ha a disposizione una mappa con evidenziati gli esercizi commerciali convenzionati nelle vicinanze e l'elenco degli esercizi segnati come preferiti. Potrà effettuare una ricerca in base al nome del negozio convenzionato presso cui vuole prenotarsi. Potrà effettuare una prenotazione su un punto vendita, consultare il numero di persone già in coda e vedere l'elenco delle prenotazioni da lui effettuate, nonché annullarle.
L’utente può scegliere di non prenotarsi in coda e ordinare la spesa al negozio. Dovrà scegliere quali prodotti del negozio acquistare. Non è prevista la gestione dei pagamenti che dovrà essere effettuata con il commerciante di persona al momento del ritiro della spesa.
Può vedere il **numero di persone in coda prima di sé**, per autodeterminare soggettivamente i tempi di attesa. L'app non può fornire un tempo di attesa esatto in quanto ogni cliente può restare nel negozio per un periodo di tempo anche ampiamente diverso. Una stima non accurata può abbassare la qualità del servizio. L'utente potrebbe riporre troppa fiducia nella stima, poi rivelatasi errata e rischiare di perdere il proprio turno o rimanere frustrato dall'accaduto.

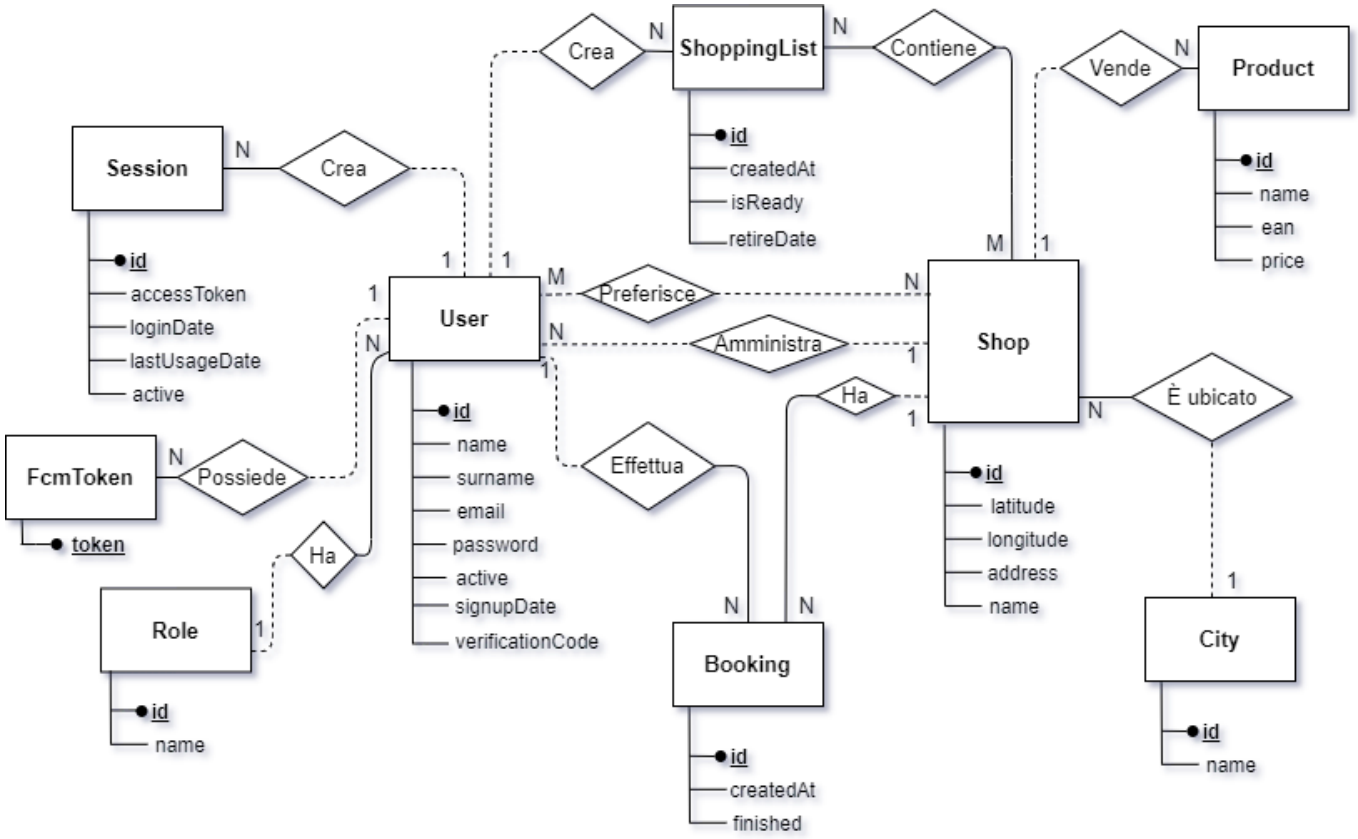
Requisiti per il commerciante

L'esercente, dopo aver ricevuto un account apposito da un amministratore del servizio, è pronto per lavorare. Il suo negozio comparirà tra quelli convenzionati a tutti gli utenti dell'app.
Può consultare l'elenco delle persone prenotate e chiamare la prossima persona in coda. In caso lo desideri, può provvedere ad annullare tutte le prenotazioni finora effettuate dai vari utenti. Questi ultimi verranno avvisati. Un commerciante può gestire un unico negozio. Per gestire più negozi si devono usare account diversi, l’app provvederà al salvataggio delle credenziali di ciascuno di essi per un rapido passaggio da uno all’altro.
Il commerciante può vedere le prenotazioni di spesa effettuate dai clienti e spuntarle come pronte. L’utente sarà avvisato. Il commerciante gestisce autonomamente i prodotti che ha in listino.

Requisiti per il fornitore/amministratore del servizio

L'amministratore dell'app è libero di eseguire operazioni CRUD (creazione, lettura, aggiornamento ed eliminazione) su utenti e negozi. Non può vedere informazioni sensibili come password: motivazioni di tale effetto nel paragrafo sulla sicurezza informatica.

Progetto della base di dati



L'entità **User** rappresenta l'utente. Si prevede che effettui l'accesso al sistema tramite indirizzo email e password. Dato che l'indirizzo e-mail può variare nel corso del tempo, si è scelto di utilizzare un ID numerico auto incrementante come attributo identificatore. È presente un attributo active per segnalare se l'utente è attivo o meno. Ricordiamo che l'utente diventa attivo se lo modifica un amministratore o verifica il proprio indirizzo email. VerificationCode è il codice di verifica inviato all'utente via e-mail per la verifica. Una volta utilizzato, assumerà valore null.

Role è l'entità che rappresenta i vari ruoli che l'utente può assumere. Nel database finale, la tabella Role conterrà i tre ruoli: admin, commerciante e utente semplice.

Shop è l'entità rappresentante dei negozi. Attributi molto importanti sono latitude e longitude, ovvero le coordinate del negozio. La **città** è vista come entità a sé, rendendo possibile anche l'aggiungi di ulteriori attributi sulla città o tradurre i nomi delle città in altre lingue.

Booking rappresenta le prenotazioni per la coda del negozio. Troviamo l'indicazione sul momento di inserimento in coda (createdAt) e se l'utente è già stato chiamato, quindi la prenotazione terminata (attributo finished).

Product è l'entità che rappresenta i prodotti inseriti in listino dai commercianti. Ogni prodotto ha il suo prezzo e nome. Non si è scelto di utilizzare il codice EAN come attributo identificatore: a parità di prodotto fisico, un commerciante può attribuirgli un prezzo e un nome diversi.

ShoppingList rappresenta la "lista della spesa" inviata da un utente al negozio per l'ordine che può ritirare di persona una volta pronto. La preparazione dell'ordine è segnalata dall'attributo isReady, mentre in retireDate è contenuta la data e ora del ritiro dell'utente.

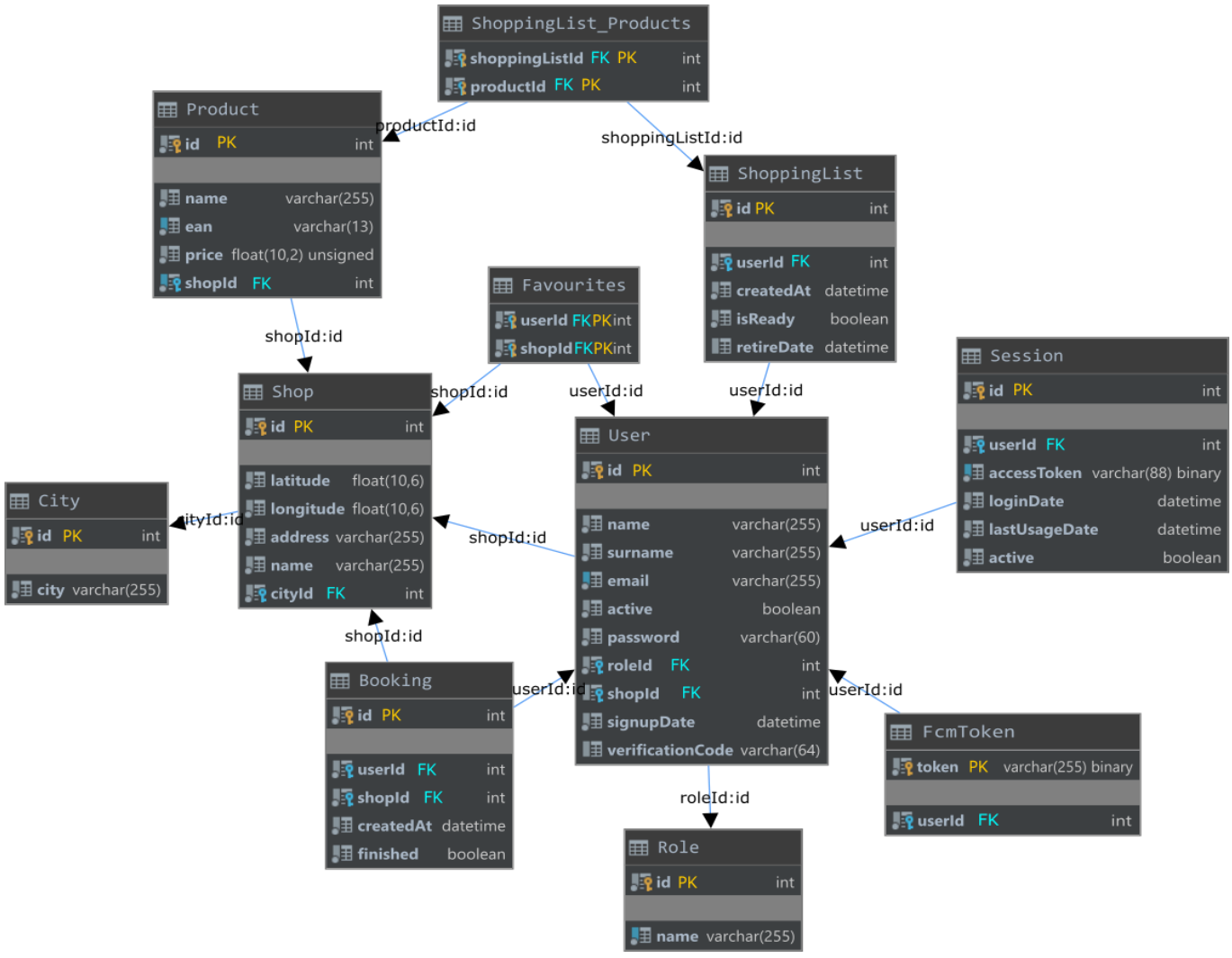
Non contiene informazioni sul negozio, essendo queste ritenute ridondanti perché sono contenute nei prodotti. Al programma server è affidato il compito di far sí che i prodotti di una lista siano tutti dello stesso negozio.

Dato che nelle ipotesi aggiuntive è stato previsto di notifiche push agli utenti, e dato che si tratta di un'applicazione Android, l'invio delle notifiche è affidato al servizio Firebase Cloud Messaging di Google. Per funzionare, assegna ad ogni app installata, per ogni dispositivo, un codice identificativo (token). Tornando al punto di vista dell'applicazione, questo token deve essere associato a un utente per utilizzarlo successivamente per inviargli notifiche push. L'entità **FcmToken** si occupa di risolvere questo aspetto, memorizzando il token, che verrà usato come attributo identificatore.
Infine, troviamo l'entità **Session** che identifica la sessione di un utente. Nonostante i dettagli sul suo funzionamento siano discussi più avanti nel paragrafo sulla sicurezza informatica, per comprendere questa relazione basta sapere che un utente si autentica nelle richieste dopo il login mediante un access token (attributo accessToken), e per motivi di sicurezza ogni sessione deve contenere indicazioni sul momento in cui è stata creata (loginDate) e quando è stata utilizzata l'ultima volta (lastUsageDate). Infine, dopo il logout, deve essere disattivata, e la sua attivazione è rappresentata dall'attributo active.

Per quanto riguarda le associazioni tra le entità:

- Un utente può amministrare un solo negozio, un negozio deve essere amministrato da uno o più utenti;
- Un ruolo può essere assegnato a più utenti, a un utente deve essere stato assegnato un ruolo;
- Un utente può effettuare più prenotazioni, una prenotazione deve essere effettuata da un solo utente;
- Una prenotazione deve essere effettuata a un solo negozio, un negozio può avere più prenotazioni;
- Una sessione deve essere creata da un utente, un utente può creare più sessioni;
- Un utente può preferire più negozi, un negozio può essere preferito da più utenti;
- Un negozio può vendere più prodotti, un prodotto deve essere venduto da un solo negozio (vedasi considerazione di prima sui prodotti e il codice EAN);
- Una lista della spesa può contenere più prodotti, un prodotto può essere contenuto in più liste della spesa;
- Una lista della spesa deve essere creata da un solo utente, un utente può creare più liste della spesa;
- Un negozio deve essere ubicato in una sola città, in una città possono essere ubicati più negozi;
- Un utente può possedere più FcmToken, un token deve essere posseduto da un solo utente.

Dopo la creazione del modello E/R, si procede con la produzione del **modello logico**.



Per una migliore leggibilità, sono presenti delle frecce che collegano le tabelle con delle FK con la tabella contenente la relativa PK.

Per i campi di testo su cui non è necessario porre uno specifico limite, si assegna dimensione 255 per questioni di efficienza. Infatti, 255 è il massimo numero rappresentabile con 8 bit, così la dimensione sarà memorizzata in un solo byte.
Per le date che devono rappresentare anche l'orario, si utilizza il tipo **DATETIME**.
Per i campi di testo dove si eseguono comparazioni case-sensitive, si specifica VARCHAR **BINARY**.

Inoltre, seguendo le regole di derivazione, le associazioni N:M creeranno una nuova tabella avente, come campi, gli identificatori delle due entità ed eventuali attributi dell'associazione.
Queste tabelle avranno per chiave primaria, la coppia delle foreign key delle due tabelle.
Vedasi le tabelle **ShoppingList_Products** e **Favourites**.

Nella tabella **User**, il campo password ha dimensione 60, ovvero quanto l'output della funzione hash bcrypt usata per le password, le quali non sono quindi salvate in chiaro.
Essendo questo un aspetto di sicurezza, è discusso più avanti nell'apposito paragrafo.

Il campo verificationCode è una stringa di 64 caratteri, dato che il codice di verifica è ottenuto da 256 bits generati in modo pseudo-random, poi codificati in una stringa esadecimale. È stata scelta la codifica esadecimale anziché altre più efficienti in termini di lunghezza della stringa risultante, dato che questa è case-insensitive e URL safe. Pertanto, può essere inserita nel link da aprire nella mail di conferma senza problemi.
Il campo shopId avrà come valore l'ID del negozio che l'utente amministra se l'utente ha ruolo di commerciante, altrimenti avrà valore null. Questa logica è affidata al programma server e non al DBMS.
Una volta implementato nel database, sul campo email verrà aggiunto un indice univoco.

La tabella **Shop** vede i campi per la latitudine e la longitudine come FLOAT(10, 6), dato che 6 cifre dopo la virgola sono sufficienti per rappresentare un punto nella mappa con precisione.

Nella tabella **Session**, per motivi descritti successivamente nel paragrafo della sicurezza, il campo accessToken ha dimensione 88 caratteri (in breve, è la lunghezza di una stringa ottenuta codificando 512 bits in Base64).
Nella tabella **Product**, il campo per il codice EAN è limitato a 13 caratteri. Il campo del prezzo limita le cifre dopo la virgola a 2.

Il codice SQL DDL è presente nel sorgente del programma server, nel file ***db.sql***. Contiene la creazione delle **tabelle** e delle **viste** (usate per semplificare le query nel programma server, aumentando la sua mantenibilitá e leggibilità). Infine, contiene la creazione di una **funzione** per calcolare la distanza di due punti dati latitudine e longitudine.

Interrogazioni SQL

a)

```
SELECT Shop.*, COALESCE(T.count, 0) / 7 AS dailyAverage
FROM Shop
LEFT JOIN (
    SELECT Shop.id, COUNT(Booking.id) AS count
    FROM Shop
    JOIN Booking ON Booking.shopId = Shop.id
    WHERE YEARWEEK(Booking.createdAt, 5) = YEARWEEK(NOW(), 5) - 1
    GROUP BY Shop.id
) T ON T.id = Shop.id
ORDER BY dailyAverage DESC;
```

Per **"flusso medio delle code"** si intende la media di prenotazioni ricevute su 7 giorni.
Prima si esegue la subquery per ottenere il numero delle prenotazioni per ogni negozio che ne ha avute. Non sono inclusi i negozi senza prenotazioni nel risultato della subquery. Tra la tabella Shop e il risultato della subquery (T) si effettua un LEFT JOIN per avere nel risultato della query complessiva tutti i negozi. Dato che LEFT JOIN assegna NULL dove non ci sia una corrispondenza, si usa la funziona COALESCE per visualizzare 0 dove ci sarebbe NULL.
Con YEARWEEK si risale alla settimana di una determinata data e la si compara alla settimana precedente al momento di esecuzione della query (NOW()). Il secondo parametro della funzione YEARWEEK indica che modalità utilizzare. Passando il valore 5, si indica l'utilizzo di settimane che iniziano con il lunedì.

b)

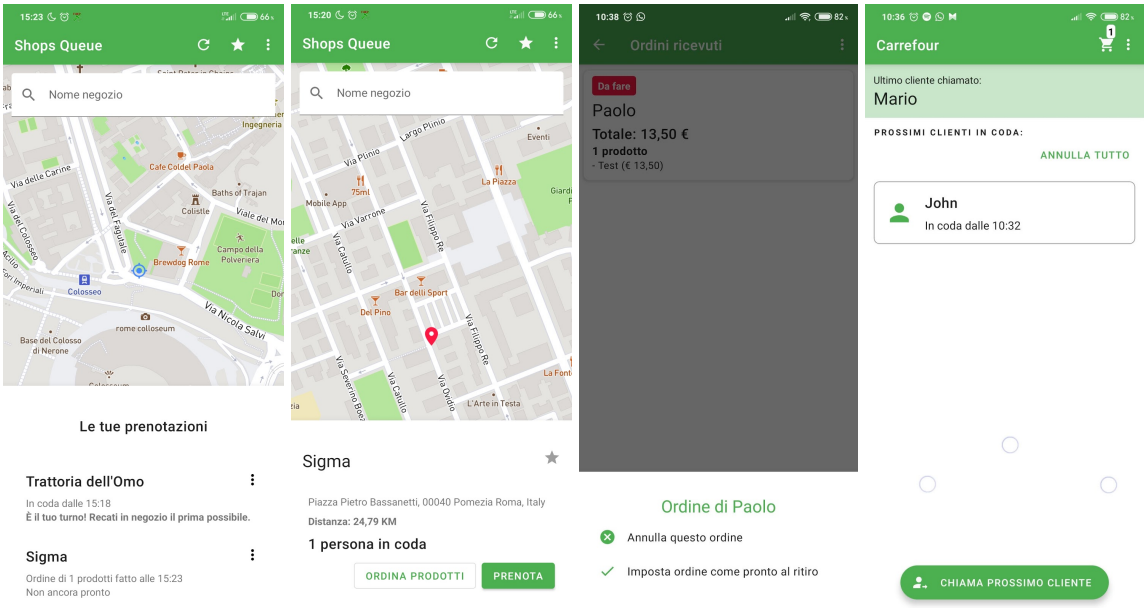
```
SELECT User.*
FROM User
    LEFT JOIN Booking ON Booking.userId = User.id
    LEFT JOIN ShoppingList ON ShoppingList.userId = User.id
WHERE DATEDIFF(NOW(), User.signupDate) >= 15
    AND Booking.id IS NULL
    AND ShoppingList.id IS NULL;
```

Per **"utilizzo dell'app"** si considera l'aver fatto una prenotazione presso un negozio o aver ordinato la spesa. Per **"iscritti"** si considerano gli utenti semplici, essendo gli unici che possono iscriversi volontariamente.
Si effettua un LEFT JOIN tra la tabella degli utenti e delle prenotazioni, poi tra utenti e delle spese ordinate, anziché un JOIN. Restituisce tutti i record della tabella User, indipendentemente dal fatto che siano presenti record nelle altre due tabelle. In caso non siano presenti, i campi delle altre due tabelle vengono impostati a NULL, condizione sfruttata nella clausola WHERE per selezionare i soli utenti senza prenotazioni e senza ordinazioni, insieme alla differenza minima di 15 giorni tra il momento di esecuzione della query (NOW()) e l'avvenuta registrazione (User.signupDate).

Implementazione

Sono state implementate le interfacce per il lato **cliente**, lato **commerciante** e lato **amministratore** del servizio, oltre alla registrazione e al login.
Per la realizzazione dell'app si utilizza il linguaggio Java, studiato durante l'A.S. 2018/2019 come parte del programma di 4^. Per il backend è usato il linguaggio PHP per il programma server, studiato durante l'A.S. 2019/2020 come parte del programma di 5^, e il linguaggio SQL per la gestione del database relazionale su cui si poggia il servizio, scelto per garantire **integrità e consistenza dei dati**. Questa tipologia di database si basa sull'algebra relazionale e usa un insieme di vincoli per imporre l'integrità dei dati nel database.
Per la **messa in servizio** del backend, si è usato un VPS (Virtual Private Server) fornito da Microsoft Azure gratuitamente agli studenti. Nei punti seguenti, sono elencate sia le misure adottate, sia ideali in un contesto reale.
Per agevolare la comprensione del programma lato server in PHP, le interazioni con il database si trovano in "dao" (Data Access Object), le classi che rappresentano le informazioni scambiate tra client e server in "dto" (Data Transfer Object). Infine in "controller" si trovano tutte le classi responsabili della gestione delle richieste del client, divise logicamente in macro categorie. Ad esempio, la logica di login è situata in controller/AuthController. index.php contiene la logica di inizializzazione comune a tutte le richieste.

Alcuni screenshots dell'app, nella visualizzazione come utente semplice e commerciante:



Architettura generale

L’applicazione è progettata secondo un'architettura di tipo client/server. Lato server, il servizio è esposto come Web Service conforme all’architettura REST, quindi:

- sono sfruttati metodi già presenti nel protocollo HTTP (GET, POST, DELETE, PUT);
- ogni risorsa è identificata da una URI (es: /shops/1/bookings)
- la gestione degli errori avviene tramite lo status code HTTP della risposta;
- l’autenticazione è gestita tramite l’header HTTP “Authorization”;
- non si usano Cookie: le REST API sono stateless.

Si è scelto di utilizzare come linguaggio per i dati scambiati tra client e server il JSON, al posto di altri linguaggi come XML, per questioni di efficienza pur rimanendo human readable.

Sul server è in esecuzione il web server nginx. Preferito rispetto ad Apache per le sue performance. Si occupa di inoltrare le richieste al processo di PHP e gestire una connessione sicura HTTPS con il client. Le richieste e risposte tra i due processi sono in chiaro.
Infine, sul server sono attivi i **log** dei tentativi di accesso via SSH e di tutte le richieste HTTP ricevute.

L'applicazione Android sviluppata in Java agisce da **REST client** (quindi anche client HTTP).
Il client è informato sull’utilizzo di JSON come linguaggio per lo scambio dei dati, mediante l’header HTTP “Content-Type: application/json” presente nella risposta del server.
Il client riceverà il JSON e lo convertirà in un oggetto Java mediante una libreria (Moshi) scelta anch’essa per efficienza rispetto ad altre come Jackson.

Nella corrente implementazione, web server, Application server e DBMS sono eseguiti sullo stesso host. Si parla di configurazione con **due tier e unico host**.
Per come è stato sviluppato attualmente il programma server, essendo stateless, è possibile replicare l'application server in più server senza nessuna modifica richiesta al codice sorgente, anteponendo ad essi un load balancer, per aumentare le prestazioni. Si parla di **RACS shared disk**, in quanto il database è condiviso.
In una situazione ottimale e in caso il traffico dovesse essere particolarmente alto, si potrebbe applicare un approccio di partitioning dove le funzionalità e i dati sono ripartiti tra i vari nodi. Se si vanno a replicare questi nodi, si ha una soluzione di tipo **RAPS**.
Lo scopo di tali soluzioni è aumentare disponibilità del servizio e scalabilità.
Nell'implementazione corrente, non sono state adottate queste ultime soluzioni per motivi di costi.

Il server è posto dietro ad un **firewall stateful** configurato come segue:

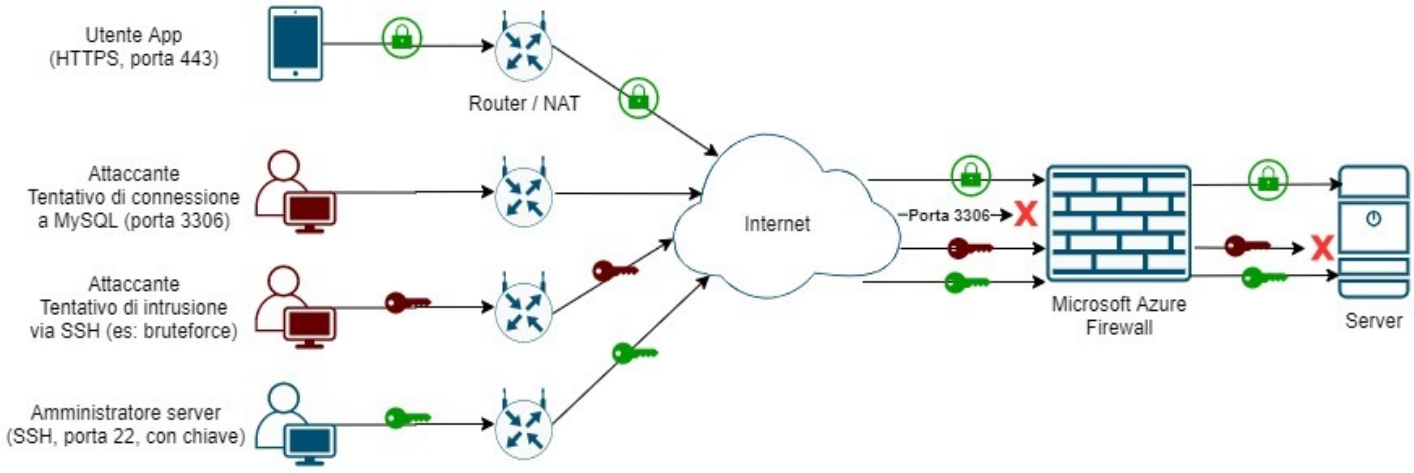
- **INPUT**
    - Default: drop
    - Allow HTTPS (443)
    - Allow SSH (22)
    - Allow connessioni established
- **OUTPUT**: default allow
- **FORWARD**: default drop, nessuna necessità contraria

Output della scansione con **nmap**:
Risultano aperte le sole porte 22 e 443, mentre le altre sono filtrate ovvero bloccate da un firewall con un drop (non deny).

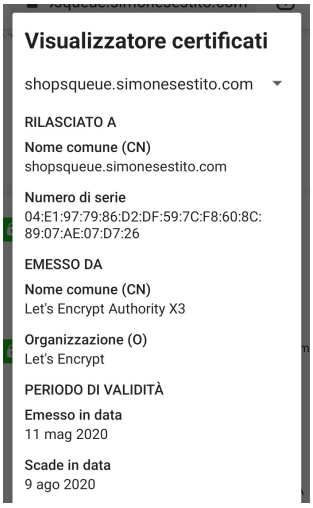
```
simone:~$ nmap shopsqueue.simonesestito.com -Pn

Starting Nmap 7.60 ( https://nmap.org ) at 2020-06-06 10:00 UTC
Nmap scan report for shopsqueue.simonesestito.com (40.89.181.155)
Host is up (0.0028s latency).
Not shown: 998 filtered ports
PORT    STATE SERVICE
22/tcp  open  ssh
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 4.85 seconds
```

Una possibile falla di sicurezza è visibile dall'accettazione del traffico sulla porta 22 da tutti gli IP, ma un eventuale attacco brute force per accedere via SSH e prendere il controllo del server è mitigato dalla necessità di una **chiave RSA 4096 bit** per l'accesso. Non è possibile accedere tramite password.

Il seguente diagramma illustra il funzionamento di diverse richieste di connessione.



Il lucchetto verde rappresenta il traffico su protocollo HTTPS usando un **certificato rilasciato da una CA**. La chiave verde identifica le connessioni SSH che utilizzano la summenzionata chiave RSA per l'autenticazione. La chiave rossa rappresenta credenziali non valide (password o chiave errata).
Il tentativo di connessione al database è bloccato dal firewall, ancora prima di raggiungere il server.
Di lato, informazioni sul certificato utilizzato.



Aspetti di sicurezza, privacy e ottimizzazione

Si è tenuto conto di diversi aspetti al fine di migliorare la sicurezza della soluzione proposta e implementata.
Partendo dalla **registrazione utente**, la password non è salvata in chiaro ma è utilizzata una funzione hash denominata "bcrypt". La si è scelta rispetto ad altre funzioni hash per la lentezza di esecuzione volontaria che mitiga attacchi brute force, e per l'utilizzo integrato di un **salt**. Se più utenti hanno scelto la stessa password, non è rilevabile vedendo il database. Inoltre, la sua esecuzione è facilmente rallentabile esponenzialmente incrementando un parametro: il numero di round.
Nella seguente immagine, ci sono alcuni utenti con la stessa password ma non è riconoscibile quali:

```
mysql> SELECT id, password FROM User;
+----+--------------------------------------------------------------+
| id | password                                                     |
+----+--------------------------------------------------------------+
|  2 | $2y$10$eSBp3KA/wZmljFMH5b.St.QcWH4sMTUNALsLbxQzwSBQKe5fUppRO |
|  3 | $2y$10$rjNJD7IqShVDHIfVsJlE6un2ayKRh6gh3O14iM0rekmu.l7.Xo6l6 |
|  4 | $2y$10$wf3s4cFv3FlGv6KT88Ol/uArln73khRhZmrYQatGUdxpR..UHrJd2 |
|  5 | $2y$10$ne9Jf08Dfx5OJxC8hzCTHu8arzenhANZKVb8uLm9feim4gzoUdRMG |
|  9 | $2y$10$U.0FbEEu/o5B1suMFBdZPucxerE98LmhgnMkdESwKLPXLezy.JQhi |
| 18 | $2y$10$1ejqDo4bVE0InGAEXvcLQe9WvZsvsuM89abrXozm9ZrCrSalzwJt6 |
| 19 | $2y$10$1NWNv14LNQHmD1PGWIm4KeZpQvh9eiYZrJaedsgHgrU96BtLFWSHC |
| 20 | $2y$10$5k.jslna9vFVsQMrLWdLIOstTTGNHJBajJWqiZTCsZPPmHvrkASKq |
+----+--------------------------------------------------------------+
8 rows in set (0.00 sec)
```

Per ciò che riguarda il **login**, vengono utilizzati dei token di accesso. Si tratta di stringhe di 88 caratteri, codificate in Base64, ottenute a partire da 512 bit generati da un **CSPRNG** (Cryptographically Secure Pseudo Random Number Generator). Il token è inviato al client, che dovrà conservarlo con cura. Per prevenire che un eventuale attaccante possa leggere il database e impersonare altri utenti loggati, l'impronta SHA-512 del token è salvata nel database al posto del token stesso.
Di seguito, a sinistra la risposta del server al login, contenente il vero token di accesso (z4WU...), a destra il record realmente inserito nel database (w/Oou…):


```
{
    "user": {
        "shop": {
            "id": 17,
            "longitude": 12.586009,
            "latitude": 41.900032,
            "address": "Viale Giorgio De Chirico
                86, 00155 Roma Roma, Italy",
            "name": "Carrefour",
            "count": 1
        },
        "active": true,
        "id": 5,
        "name": "Gino",
        "surname": "Passalacqua",
        "email": "john.scott@shop.com",
        "role": "OWNER",
        "shopId": 17
    },
    "accessToken":
        "z4WU4hapD1vJQsH1oZoratrwYV9rCTQUe/
        M0a29R8S1SqiPL4E2DZwX5VP/
        b6XYohyj4AuSlJsySg5/niwXfoQ=="
}
```

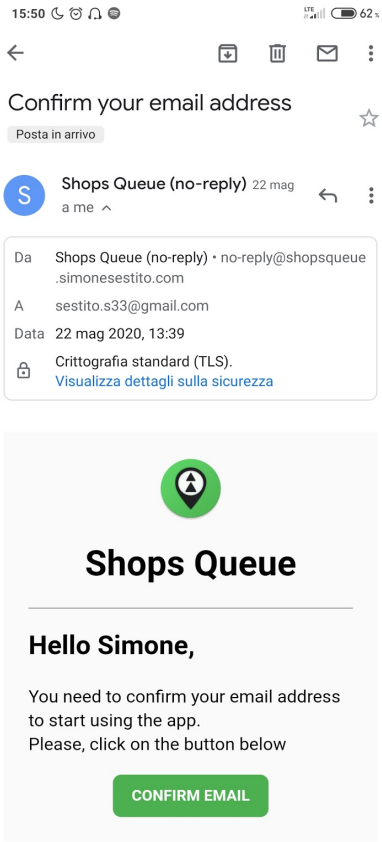

```
mysql> SELECT userId, accessToken FROM Session WHERE
 userId = 5;
+--------+-----------------------------------------
----------------------------------------------+
| userId | accessToken
                                                |
+--------+-----------------------------------------
----------------------------------------------+
|      5 | w/OoumKeAjgfBwF13lB6KKor1XBAN2jfd0nk7pswI
GOB4RfGswOZIjDJ9nvSqq/dOqeSMn9hIUQUnsNQcv18rQ== |
+--------+-----------------------------------------
----------------------------------------------+
1 row in set (0.00 sec)
```

Lato **client**, il token di accesso è memorizzato utilizzando API di sistema Android che permettono la crittografia del token usando una chiave di cifratura di sistema, non accessibile da nessuna app utente e, in base al dispositivo usato, nemmeno accessibile dal sistema Android, essendo conservata su hardware apposito (StrongBox). Come algoritmo di crittografia simmetrica, è usato **AES** con chiave a **256 bit**. Infine, la stringa criptata è salvata in SharedPreferences, un database key-value che è normalmente, ma non sempre, accessibile dalla sola applicazione che lo ha creato.
Usare token custom di questo tipo rispetto alle sessioni PHP porta diversi vantaggi: l'utente ha la possibilità di avere un elenco degli accessi attivi, vedere la data di creazione e ultimo utilizzo, nonché revocare il token.
Di seguito, i dati salvati in SharedPreferences, leggibili (ma incomprensibili) da altre app solo su device Android compromessi (es: rooted):

com.simonesestito.shopsqueue_preferences.xml

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="AWAnghalHzzfRyRxN02psd7sROZerJVe2oLUNN7yoawPfRNY3Q==">AWguXulGkZdtqo3yn4OFSRq1oqUt2LY1OoXcqd+duqMGCOhlGXZZdQ10va664YKGNXPLTH81Ky4
    <string name="__androidx_security_crypto_encrypted_prefs_key_keyset__">12a901d44619007c0233e0744413a0174bd1f0fa788c047276dcd5c56639ab8ccff937ef
    <string name="__androidx_security_crypto_encrypted_prefs_value_keyset__">1288011d048b5f971c5bb8890368c3cb4094749105b3ece89fb12c83f0075c689af056
</map>
```

Durante il **logout**, l'app invia la richiesta di logout al server, il quale provvederà ad invalidare la sessione associata al token. Questo di norma non ha scadenza ma, essendo generato da 512 bit pseudo casuali, un attacco bruteforce richiederebbe provare fino ad un massimo di 2^512 combinazioni. Può essere rallentato limitando il numero di richieste per indirizzo IP in un intervallo di tempo. In ogni caso, visto il database, ci sono tutti gli elementi per implementare un sistema di annullamento di un token automatico dopo un intervallo di tempo di inattività.



Al fine di evitare la creazione di **account fittizi**, è richiesta la verifica dell'indirizzo e-mail utilizzato. Bisognerà aprire un link monouso che verrà inviato via e-mail per attivare l'account. È possibile implementare, ma non implementata, la verifica del numero telefonico per ridurre al minimo la creazione di account spam. Di lato è visibile uno screenshot della mail di conferma.

Nella parte di applicazione lato server in PHP sono utilizzati i **prepared statements** per bloccare attacchi **SQL Injection**.
Non avendo un'interfaccia web, non sono necessari controlli contro attacchi XSS o altre injection di codice HTML o JS.
Tutti i dati inclusi nel body HTTP della richiesta sono validati sia dal client prima di inviare la richiesta per dare un feedback immediato all'utente (**client-side validation**), sia dal server (**server-side validation**) che risponderà con codice HTTP 400 Bad Request in caso di errori di validazione.
Il controllo lato server è sempre necessario in quanto un attaccante potrebbe effettuare le stesse richieste HTTP dell'app inviando dati malformati (tipi errati o altri dati di cui ne è impedito dal client l'invio), oppure eseguire operazioni che non sarebbe autorizzato a fare se il controllo fosse solo lato client.

Ad ogni richiesta, il programma server controllerà la validità del token di accesso fornito dal client e l'autorizzazione dell'utente in questione di effettuare l'azione che sta richiedendo.

Come piano di **Disaster Recovery**, si attuano backup periodici del database, conservati in un posto sicuro esterno al server. Inoltre, il provider utilizzato per il server garantisce una continuità operativa di oltre il 99,9%.

Il traffico tra client e server utilizza obbligatoriamente il protocollo HTTPS, non HTTP.
Lato server, tale caratteristica è garantita dal blocco della porta 80 nel firewall, lasciando aperta solo la 443 (HTTPS) e dal web server che non si trova in ascolto sulla porta 80 (HTTP). Lato client, il sistema operativo Android impedisce all'app di effettuare comunicazioni in plaintext, HTTP incluso.

Ottimizzazione

L'applicazione Android è resa compatibile con i servizi Android per **l'accessibilità**. Per ogni immagine è presente una descrizione che uno screen reader può leggere in caso di un utente non vedente. Se l'utente risulta ipovedente, l'app si andrà ad adattare alla dimensione del font decisa nelle impostazioni di sistema.
Ove supportato dalla versione di Android, l'app andrà ad applicare un tema chiaro o scuro in base a quanto impostato a livello di sistema, migliorandone **l'usabilità**. Sempre in tema usabilità, l'app segue gli standard del Material Design, al fine di garantire la massima intuitività d'uso seguendo linee guida comuni all'intero ecosistema Android e Google.

Per le richieste che possono restituire ingenti quantità di dati, si adotta la **paginazione con offset**. Ogni volta che il client effettua la richiesta (ad esempio per l'elenco degli utenti), dovrà indicare il numero di pagina che sta richiedendo. Il server applicherà un limite e un offset sulla query (LIMIT in SQL) per restituire solo una porzione di dati, insieme al numero di elementi totali.
Un **problema** della paginazione con offset rispetto ad altri tipi come la paginazione a cursore è il seguente: se tra una richiesta e l'altra si sono aggiunti dei dati, la nuova pagina richiesta conterrà degli elementi già inclusi nella pagina precedente. Questo è stato risolto lato client utilizzando strutture dati offerte da Java adeguate alla soluzione, come *LinkedHashSet*, ovvero un set (impedisce di avere duplicati) ma mantiene l'ordinamento esattamente così come è ricevuto dal server.
Adottare un sistema di paginazione permette di ridurre l'utilizzo della batteria, del traffico dati e della CPU per il parsing della risposta JSON.

Esempio di risposta paginata:

```
{
  "page": 0,
  "totalPages": 2,
  "totalItems": 35,
  "data": [ ... ]
}
```

Per l'implementazione delle funzionalità di notifiche push è utilizzato il Web Service di Google, Firebase Cloud Messaging. Per quanto concerne l'invio di email di verifica si utilizza il Web Service SendGrid.