ELABORATO PER LA PROVA D'ESAME

INDIRIZZO: ITIA – INFORMATICA E TELECOMUNICAZIONI
ARTICOLAZIONE "INFORMATICA"
ANNO SCOLASTICO 2019-2020
Discipline: INFORMATICA E SISTEMI E RETI

Traccia

Progettare un sistema che supporti i clienti nelle fasi di accesso e di acquisto presso esercizi commerciali "tradizionali" (anche alla luce delle restrizioni anti-assembramento).
In particolare, fatte le opportune ipotesi, si chiede di descrivere:

- il progetto dell'infrastruttura tecnologica ed informatica necessaria a gestire il sistema, dettagliando:
  - l'architettura di rete e le caratteristiche dei sistemi server
  - le modalità di comunicazione;
  - gli elementi per garantire la sicurezza del sistema;
- l'architettura dei dati e dei software che si intende implementare, dettagliando il modello dei dati e le modalità di interazione;
- una parte significativa dell'applicazione.

Soluzione

Sommario

1. Analisi

Al fine di promuovere il cosiddetto distanziamento sociale, ma anche al fine di agevolare clienti e negozianti nella compravendita di beni e/o servizi tramite via telematica, il sistema in questione mette a disposizione un sito web/applicazione per smartphone, che permette a clienti e negozianti di accordarsi circa il ritiro o la consegna a domicilio della merce, garantendo una pianificazione e schedulazione dei processi di consegna in accordo con le regolamentazioni governative anti-contagio.

L'applicazione offre dunque al negoziante la possibilità di registrare i propri punti vendita all'interno del sistema, nonché la semplice e intuitiva inserzione dei relativi prodotti. Analogamente, il cliente potrà gestire gli acquisti secondo le proprie esigenze; sarà poi compito del negoziante fornire, in base alle quantità disponibili dei prodotti, una risposta al sistema, che informerà il cliente sulla fattibilità dell'ordine. Qualora la quantità di prodotto in-stock non sia sufficiente, il cliente potrà scegliere se annullare l'ordine o richiedere una quantità inferiore di merce.

Il negoziante potrà anche agire come cliente e analogamente un cliente potrà diventare negoziante nel momento in cui registra un proprio punto vendita: l'entità generica è perciò identificata con il termine di "utente". L'utente ha perciò la possibilità di gestire una molteplicità di carrelli, il di cui ciascuno è relativo ad un determinato negozio da cui intende comprare della merce. A differenza di un sistema di tipo tailor-made, questa soluzione presenta il vantaggio di essere flessibile in quanto offre la possibilità a chiunque disponga di uno smartphone di fruirne.

2. Infrastruttura tecnologica e informatica

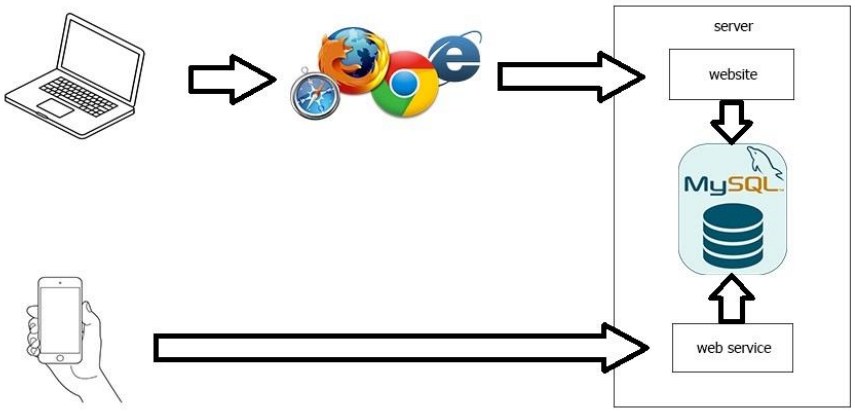
2.1 Sistemi server

Il sistema è di tipo **distribuito**, **centralizzato** e **datacentrico** e perciò costituito da una molteplicità di client che si interfacciano con un server, il quale, interagendo con un database, garantisce un'efficiente, efficace e sicura manipolazione dei dati d'interesse. L'architettura del sistema è perciò di tipo **three-tier**: il server sarà l'unico ad avere accesso al DBMS (che si interfaccia con il database, accedendo ai dati) e dunque gli utenti del sistema accederanno alle informazioni tramite l'interfaccia esposta dal server.

Il sistema server è composto da:

- Apache, utilizzato come webserver, che si occupa di trasferire le pagine web tra client e server, interfacciandosi con l'utente tramite il browser.
- MySQL, utilizzato come DBMS. Ha il compito di gestire il database, in particolare si occupa gestire l'aggiornamento e la ricerca dei dati.

- Tomcat, un servlet container utilizzato come webservice per gestire lo scambio di risorse tra client e server. Viene utilizzato per rendere accessibile il servizio tramite applicazione per smartphone.



2.2 Modalità di comunicazione

L'architettura del sistema consente all'utente di accedere al servizio tramite sito web o applicazione per smartphone; per quest'ultima opzione è preferibile l'implementazione di web-service di tipo **RESTful**. Lo scambio di dati tra client e server avverrà tramite il protocollo **HTTPS**, che, basandosi sui protocolli SSL e TLS, garantisce il criptaggio dei dati (trasferiti tramite ipertesto) lungo il loro transito sulla rete, evitando quindi che vengano intercettati da terzi. Questo processo di cifratura viene detto EE2E (end-to-end encryption). I protocolli SSL e TLS prevedono una fase di setup (handshake), che consiste nella negoziazione tra client e server sul protocollo di crittografia da utilizzare, a cui segue la fase di scambio delle chiavi (che avviene utilizzando chiavi asimmetriche), utilizzate per la comunicazione, che avviene in maniera bidirezionale utilizzando chiavi simmetriche. Il sistema sarà poi analizzato dall'**autorità di certificazione**, che potranno rilasciare il certificato SSL, garantendo così l'affidabilità del servizio.

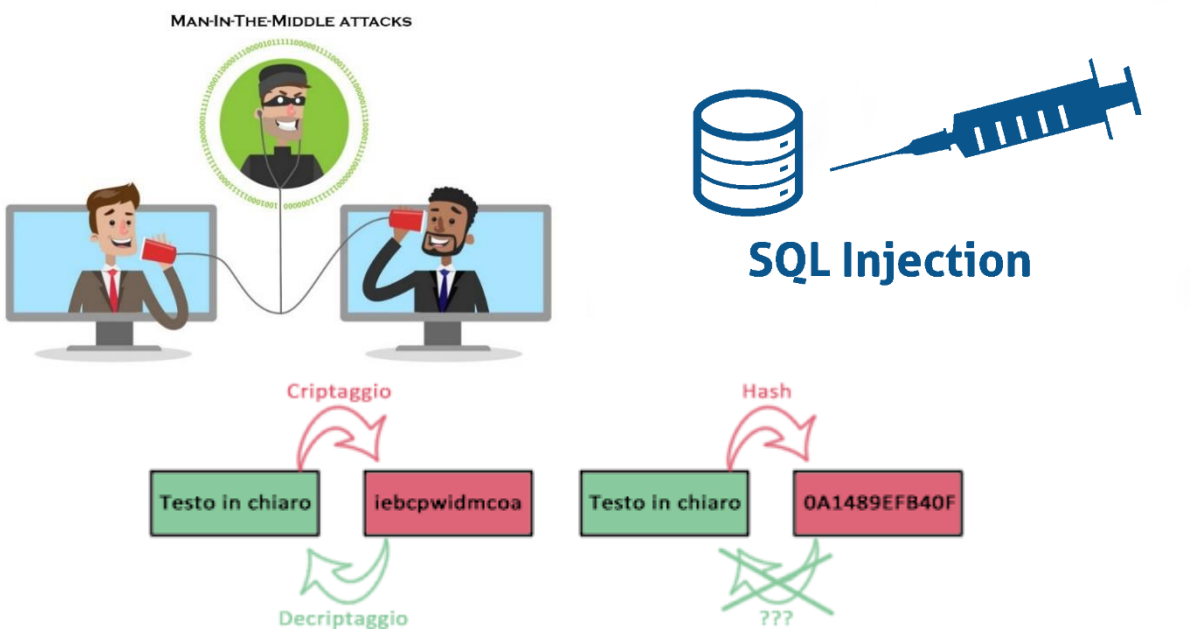
Al fine di garantire l'autenticità del client, il sistema implementa un meccanismo di login, che, qualora vada a buon fine, restituirà al client un **token** di tipo **JWT** da utilizzare per qualsiasi tipo di richiesta HTTPS al server. L'utilizzo del token si presenta come un ottimo compromesso per garantire sicurezza in quanto costituito da una stringa alfanumerica "firmata" (tramite **firma digitale**) dal server attraverso una chiave privata e perciò difficile da falsificare o rubare, anche perché il token ha un tempo di vita limitato. Nel caso in cui venisse modificato, il server se ne accorgerebbe, dal momento che il campo "signature" del token conterrebbe una firma non corrispondente a quella appena calcolata. In questo modo l'utente non deve inviare le proprie credenziali ogni volta che effettua una richiesta al server.

Il sistema, una volta implementato, sarà reso accessibile dal web tramite **server dedicato** oppure **cloud**. Nel caso di server dedicato, sarà necessario acquistare della componentistica hardware fisica, oltre che un indirizzo IP statico (oppure un DNS dinamico, che inoltrerebbe l'indirizzo IP dinamico ad un dominio fisso). Si procederà quindi all'implementazione di un server, che dovrà essere protetto da uno o più **firewall** e richiederà l'implementazione di una **DMZ**; quest'ultima avviene tramite una specifica tecnica di NAT, chiamata port-forwarding, una tecnica utilizzata per permettere ad un utente esterno di raggiungere un host con indirizzo IP privato. La porta su cui risiederà il server viene, come suol dirsi, "aperta", consentendo l'accesso al server dall'esterno. Come precedentemente accennato, al fine di prevenire possibili attacchi, è necessario inserire un firewall che protegga la rete dall'esterno e un altro firewall che protegga il server in questione, garantendo così un accurato filtraggio delle richieste. L'idea del server dedicato, tuttavia, si presenta come una soluzione piuttosto complicata, che richiede molto lavoro e manutenzione, un pesante utilizzo di banda, hardware ad alte prestazioni e ulteriore componentistica hardware che garantisca sistemi di backup dei dati; si tratta dunque di una soluzione tutt'altro che economica. Per questo motivo, è più agevole ed economico ricorrere a sistemi di cloud computing, che presentano diversi vantaggi, tra cui innanzitutto la scalabilità, principio per il quale è possibile aumentare o diminuire le prestazioni hardware in modalità on-demand, affittando, a seconda delle proprie esigenze, una certa quantità e qualità di componentistica. Questo principio fa sì che l'hardware utilizzato dal sistema sia "virtuale", annullando lavoro, manutenzione e costi richiesti dall'implementazione "fisica" dello stesso (tipica dei server dedicati).

2.3 Elementi per garantire sicurezza

Al fine di salvaguardare la sicurezza del sistema, occorre prestare attenzione a una serie di tipologie attacco che potrebbero minacciare l'integrità, riservatezza e persistenza dei dati. Molti di questi riguardano la tipologia **MITM** (man-in-the-middle), tra cui attacchi di **eavesdropping**, **replay attack**, **session hijacking** e **misrouting**, che vengono contrastati grazie all'utilizzo del protocollo HTTPS (anziché HTTP). Quest'ultimo infatti garantisce non solo il criptaggio del payload (dati effettivi), ma anche dell'header, che contiene dati privati utilizzati per l'autenticazione, come il token (contenuto nel campo cookie). Altre tipologie di attacco molto frequenti sono **DoS e DDos**, con cui l'attaccante fa in modo di sommergere la vittima con pacchetti non chiusi che vanno a sovraccaricare il sistema, il quale a un certo punto non riesce più a gestire il carico di lavoro e dunque si blocca. Per affrontare questo problema in genere si utilizzano i **firewall**, che seppur non infallibili, sono in grado di filtrare le richieste, dal momento che analizzano i pacchetti in entrata basandosi su una black-list che contiene gli indirizzi degli host potenzialmente pericolosi (quelli che superano un certo limite di richieste al secondo). Tuttavia, nonostante i pacchetti provenienti da host malevoli vengano respinti, la loro gestione contribuisce comunque alla congestione del sistema; una soluzione potrebbe essere l'utilizzo di un server proxy come **CloudFlare**,

che, in quanto dotato di prestazioni molto elevate, riesce a smaltire il traffico senza particolari problemi. Quanto ai dati, le password non vengono mai salvate in chiaro, ma vengono salvate le corrispondenti funzioni hash, funzioni matematiche di tipo one-way, come **MD5**, da cui non è possibile effettuare la trasformazione inversa. Perciò, il meccanismo di login si basa sul confronto della stringa hash calcolata sulla base della password fornita con quella presente sul database. Occorre inoltre prestare attenzione agli attacchi di tipo **SQL injection**, che coinvolgono i dati all'interno del database; si tratta di uno tra attacchi più pericolosi e distruttivi che possono essere attuati su un sistema data centrico. Una possibile opzione potrebbe essere consistere nell'utilizzo di prepared statements per effettuare le query al database, che sono in grado di filtrare gli input potenzialmente malevoli, sostituendo a parole chiave sql i corrispondenti caratteri di escape.



3 Modello dei dati

3.1 Schema concettuale

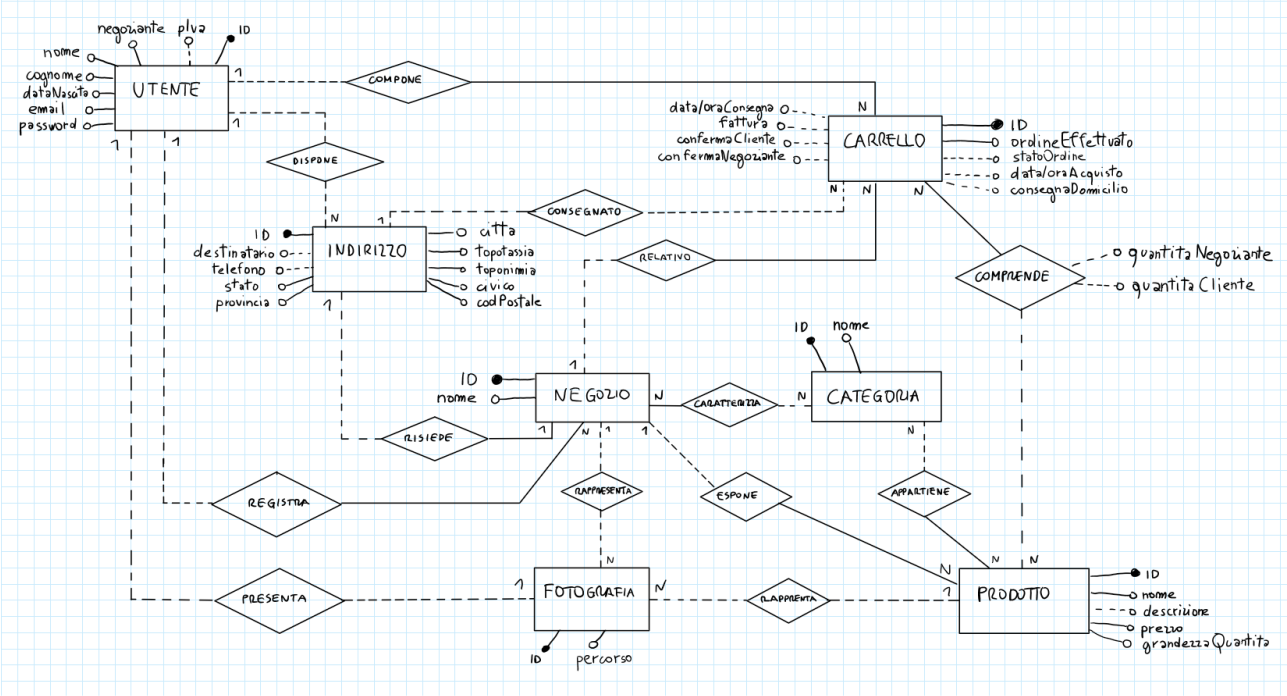
Lo schema ER metterà in evidenza le relazioni esistenti tra le seguenti entità:

- Utente: rappresenta una generalizzazione delle sottoentità cliente e negoziante, dunque un account che può sia acquistare che gestire le vendite.
- Carrello: identifica il "contenitore" dei prodotti che un utente indende comprare da un certo negozio. Nel momento in cui l'utente effettua l'ordine, una serie di campi opzionali (che caratterizzano l'ordine vero e proprio) verranno compilati dal sistema.
- Prodotto: rappresenta la merce acquistabile da un certo negozio e identificabile in una o più categorie di prodotti.
- Negozio: rappresenta il negozio fisico situato in una determinata località, che dispone di merce da vendere.
- Categoria: rappresenta un sottoinsieme dell'insieme "prodotti" o "negozi", che accomuna una parte di questi. Viene utilizzata per implementare un sistema di filtraggio per la ricerca.

- Indirizzo: specifica una determinata località geografica, utilizzata per fornire indicazioni circa il domicilio di consegna o il collocamento di un negozio.
- Fotografia: ha lo scopo di rappresentare graficamente un prodotto, un negozio (o una parte di esso) o un utente (foto profilo).

Le relazioni più importanti sono:

- Comprende: associa un prodotto ad un carrello, denotando il "riempimento" di un carrello della spesa.
- Espone: associa un prodotto ad un negozio.
- Compone: associa un carrello a un utente; avviene quando un cliente inserisce il primo prodotto di un certo negozio in un carrello.



Osservazioni:

- L'associazione "comprende": poiché diventerà una tabella, include alcuni attributi, quali "quantitaCliente" e "quantitaNegoziante", che verranno utilizzati per la negoziazione della quantità di prodotto tra cliente e negoziante. L'unità di misura della quantità di prodotto è espressa dall'attributo "grandezzaQuantita", presente nella tabella "prodotti". Tale attributo si traduce in un enumerativo, che può assumere valori come "pz", che denota il numero di pezzi, "gr", cioè una quantità espressa sottoforma di peso, "lt", una capacità, ecc. La quantità concordata e approvata da cliente e negoziante sarà impostata sull'attributo "quantitaNegoziante" e dovrà essere interpretata sulla base dell'unità di misura specificata dall'attributo "grandezzaQuantita" del relativo prodotto.
- E' possibile constatare che il sistema non tiene traccia della quantità di prodotto presente nel magazzino del negozio. Questo perché demanderebbe al negoziante una notevole complessità di gestione del negozio all'interno del sistema, per cui risulterebbe necessario registrare ciascun incremento/decremento della quantità di

ogni prodotto per ciascun rifornimento/vendita. Inoltre, è da considerare il fatto che il negozio potrebbe già avere un sistema informatico per l'inventario della merce, il che implicherebbe l'integrazione dello stesso al sistema in questione.

- Non appena l'utente effettua l'ordine, gli attributi opzionali presenti sull'entità "carrello" verranno inizializzati "manualmente" dal server; l'attributo "ordineEffettuato", che verrà impostato a "true", servirà a distinguere un ordine vero e proprio da un semplice carrello.
- L'entità "utente" rappresenta un'entità generica utilizzata per descrivere sia un cliente che un negoziante, che si distinguono per il valore booleano indicato dall'attributo "negoziante", grazie a cui cliente e negoziante disporranno di viste diverse (il negoziante potrà accedere a una sezione in cui potrà elaborare gli ordini).

Ristrutturazione

Lo schema ER deve essere ristrutturato prima di procedere con l'operazione di mapping. In realtà, durante la fase di progettazione concettuale (sviluppo dell'ER) ho cercato di progettare l'architettura in marniera ottimizzata, così da ristrutturare lo schema il meno possibile durante la fase sucessiva. Quanto alla scelta delle **chiavi primarie**, ho utilizzato **chiavi artificiali** (ID) per ciascuna tabella; lo stesso discorso vale per le associazioni di tipo N a N, che diventano tabelle, utilizzando sempre una chiave artificiale. Questa soluzione è computazionalmente più efficiente rispetto all'utilizzo di altre **chiavi candidate** composte, ma soprattutto garantisce la **normalizzazione** del database fino alla 3ª forma.

3.2 Schema logico (mapping)

Questa fase è finalizzata all'ottenimento dello schema logico tramite il processo di mapping.

```
Utenti(ID, nome, cognome, dataNascita, email, password,
     negoziante, pIva, *IdFotografia)
Carrelli(ID, ordineEffettuato, statoOrdine, data/oraAcquisto,
     consegnaDomicilio, data/oraConsegna, fattura
     confermaCliente, confermaNegoziante, *IdUtente, *IdIndirizzo,
     *IdNegozio)
Negozi(ID, nome, *IdIndirizzo, *IdNegoziante)
Indirizzi(ID, destinatario, telefono, stato, provincia,
     citta, topossia, toponimia, civico,
     codPostale, *IdUtente)
Prodotti(ID, nome, descrizione, prezzo,
     grandezzaQuantita, *IdNegozio)
Categorie(ID, nome)
Fotografie(ID, percorso, *IdNegozio, *IdProdotto)
Comprende(ID, quantitaCliente, quantitaNegoziante,
     *IdCarrello, *IdProdotto)
Caratterizza(ID, *IdNegozio, *IdCategoria)
Appartiene(ID, *IdProdotto, *IdCategoria)
```

3.3 Definizione dello schema (DDL)

Seguono le definizioni dello schema in linguaggio SQL, tramite **DDL**:

```
CREATE TABLE Utenti (
    ID int(11) PRIMARY KEY AUTO_INCREMENT,
    nome varchar(16) NOT NULL,
    cognome varchar(16) NOT NULL,
    dataNascita date NOT NULL
       CHECK(YEAR(GETDATE() - dataNascita)>=18),
    email varchar(32) UNIQUE,
    password char(32) NOT NULL,
    negoziante tinyint(1) NOT NULL DEFAULT '0',
    pIva char(11),
    IdFotografia int(11),
    FOREIGN KEY (IdFotografia)
        REFERENCES Fotografie(ID)
            ON UPDATE CASCADE
            ON DELETE SET NULL
);


CREATE TABLE Negozi (
    ID int(11) PRIMARY KEY AUTO_INCREMENT,
    nome varchar(32) UNIQUE,
    IdIndirizzo int(11) NOT NULL,
    IdNegoziante int(11) NOT NULL,
    FOREIGN KEY (IdIndirizzo)
        REFERENCES Indirizzi(ID)
            ON DELETE RESTRICT
            ON UPDATE CASCADE,
    FOREIGN KEY (IdNegoziante)
        REFERENCES Utenti(ID)
            ON DELETE SET NULL
            ON UPDATE CASCADE,
);

CREATE TABLE Comprende (
    ID int(11) PRIMARY KEY AUTO_INCREMENT,
    quantitaCliente int (4) NOT NULL DEFAULT '1'
       CHECK(quantitaCliente > 0),
    quantitaNegoziante int(4) DEFAULT NULL
       CHECK(quantitaNegoziante > 0),
);



CREATE TABLE Carrelli (
    ID int(11) PRIMARY KEY AUTO_INCREMENT,
    ordineEffettuato tinyint(1) NOT NULL
       DEFAULT '0',
    statoOrdine ENUM("in elaborazione",
       "completato", "annullato") DEFAULT NULL,
    dataOraAcquisto datetime DEFAULT NULL,
    consegnaDomicilio tinyint(1) DEFAULT NULL,
    dataOraConsegna datetime DEFAULT NULL
       CHECK(dataOraConsegna >= GETDATE()),
    fattura tinyint(1) DEFAULT NULL,
    confermaCliente tinyint(1) DEFAULT NULL,
    confermaNegoziante tinyint(1) DEFAULT NULL,
    IdUtente int(11) NOT NULL,
    IdIndirizzo int(11) DEFAULT NULL,
    IdNegozio int(11) NOT NULL,
    FOREIGN KEY (IdUtente)
        REFERENCES Utenti(ID)
            ON DELETE SET NULL
            ON UPDATE CASCADE,
    FOREIGN KEY (IdIndirizzo)
        REFERENCES Indirizzi(ID)
            ON DELETE SET NULL
            ON UPDATE CASCADE,
    FOREIGN KEY (IdNegozio)
        REFERENCES Negozi(ID)
            ON DELETE SET NULL
            ON UPDATE CASCADE
);
```

```
CREATE TABLE Prodotti (
    ID int(11) PRIMARY KEY AUTO_INCREMENT,
    nome varchar(32) NOT NULL,
    descrizione varchar(256) DEFAULT NULL,
    prezzo int(4) NOT NULL,
    grandezzaQuantita ENUM("pz", "gr", "lt"),
    IdNegozio int(11) NOT NULL,
    FOREIGN KEY (IdNegozio)
        REFERENCES Negozi(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE
);

CREATE TABLE Indirizzi (
    ID int(11) PRIMARY KEY AUTO_INCREMENT,
    destinatario varchar(32) DEFAULT NULL,
    telefono varchar(9) DEFAULT NULL,
    stato varchar(16) NOT NULL,
    provincia char(2) NOT NULL,
    citta varchar(16) NOT NULL,
    topotassia varchar(8) NOT NULL,
    toponimia varchar(32) NOT NULL,
    civico char(3) NOT NULL,
    codPostale char(5) NOT NULL,
    IdUtente int(11) DEFAULT NULL,
    FOREIGN KEY (IdUtente)
        REFERENCES Utenti(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE
);

CREATE TABLE Fotografie (
    ID int(11) PRIMARY KEY AUTO_INCREMENT,
    percorso varchar(64) UNIQUE,
    IdNegozio int(11) DEFAULT NULL,
    IdProdotto int(11) DEFAULT NULL,
    FOREIGN KEY (IdNegozio)
        REFERENCES Negozi(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    FOREIGN KEY (IdProdotto)
        REFERENCES Prodotti(ID)
            ON DELETE CASCADE,
            ON UPDATE CASCADE

);

CREATE TABLE Appartiene (
    ID int(11) PRIMARY KEY AUTO_INCREMENT,
    IdProdotto int(11) NOT NULL,
    IdCategoria int(11) NOT NULL,
    FOREIGN KEY (IdProdotto)
        REFERENCES Prodotti(ID)
            ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (IdCategoria)
        REFERENCES Categorie(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE
)
```

```
CREATE TABLE Caratterizza (
    ID int(11) PRIMARY KEY AUTO_INCREMENT,
    IdNegozio int(11) NOT NULL,
    IdCategoria int(11) NOT NULL,
    FOREIGN KEY (IdNegozio)
        REFERENCES Negozi(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    FOREIGN KEY (IdCategoria)
        REFERENCES Categorie(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE
);
```

```
CREATE TABLE Categorie (
    ID int(11) PRIMARY KEY AUTO_INCREMENT,
    nome varchar(32) UNIQUE
);
```

3.4 Esempi di query

1. Elenco di tutti gli ordini in corso che contengono un certo prodotto, fornito come parametro.

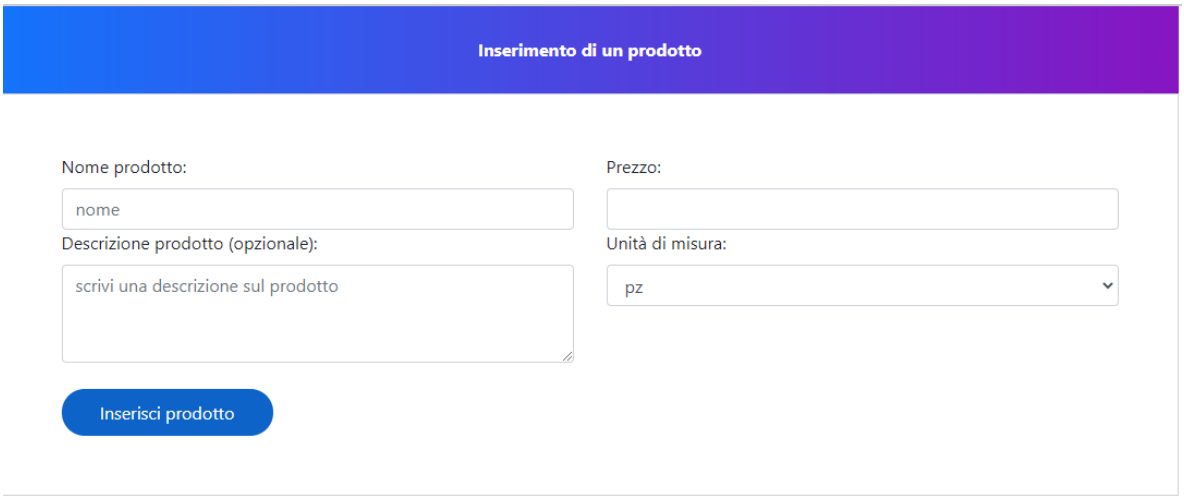
```
SELECT distinct Carrelli.*
FROM Comprende INNER JOIN Carrelli
    ON Comprende.IdCarrello = Carrelli.ID INNER JOIN Prodotti
    ON Comprende.IdProdotto = Prodotti.ID
WHERE Carrelli.statoOrdine = "in elaborazione" AND
   Prodotti.nome = [nomeProdotto]
```

2. Utente che ha effettuato più ordini in un certo anno fornito come parametro.

```
SELECT Utenti.*
FROM Carrelli INNER JOIN Utenti
    ON Carrelli.IdUtente = Utenti.ID
WHERE ordineEffettuato = '1' AND
   YEAR(dataOraAcquisto) = [annoParametro]
GROUP BY IdUtente
HAVING COUNT(*) >= ALL (
    SELECT COUNT(*)
    FROM Carrelli
    WHERE ordineEffettuato = '1' AND
        YEAR(dataOraAcquisto) = [annoParametro]
    GROUP BY IdUtente
)
```

4. Segmento significativo dell'applicazione

4.1 Frontend

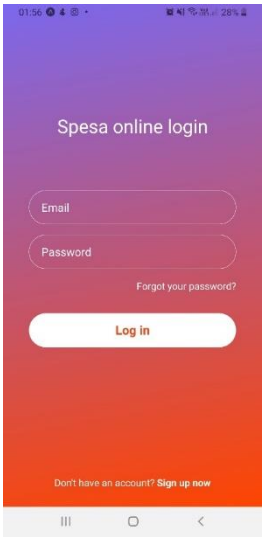


L'interfaccia utente, raggiungibile tramite sito web, deve essere accessibile da qualsiasi piattaforma, sia desktop che mobile e perciò è necessario che venga implementata tramite un **framework responsive**, come ad esempio **Bootstrap** o **JQuery**. I dati, prima di essere inviati al server, vengono controllati tramite codice **Javascript**, al fine di limitare il traffico verso il server.
Il seguente è un esempio di **form** html che funge da interfaccia utente tramite browser per l'operazione di inserimento di un prodotto (all'interno di un negozio):

```
<form method="post" action="inserimentoProdotto.php">
        <h1>Inserimento prodotto</h1>
        <div>
            <label for="nome">Nome prodotto:</label>
            <input type="text" name="nome" placeholder="nome" required>

            <label for="descrizione">
                  Descrizione prodotto (opzionale):
            </label>
            <input type="text" name="descrizione" placeholder="scrivi una
                  descrizione sul prodotto" required>

            <label for="prezzo">Prezzo:</label>
            <input type="number" name="prezzo" required>

            <label for="grandezze">Unità di misura:</label>
            <select id="grandezze">
                <option value="pz">pz</option>
                <option value="lt">lt</option>
                <option value="gr">gr</option>
            </select>
            <input type="submit" value="Inserisci prodotto">
        </div>
</form>
```

4.2 Middleware

L'applicazione per smartphone che fa uso di **RESTful web service** sviluppati tramite **JAX-FX**, viene implementata tramite React Native, un framework di Javascript utilizzato per lo sviluppo di applicazioni **cross-platform**. Il seguente è un frammento di codice che implementa una **Promise**, un oggetto di Javascript che permette di gestire il successo o fallimento di un'operazione asincrona; in questo caso si tratta di una richiesta di login al server. In caso di successo, l'utente verrà reindirizzato alla homepage, mentre in caso contrario gli sarà comunicato che le credenziali inserite sono errate e sarà invitato a ritentare. Si tratta dunque di un'operazione che fa da "tramite" tra frontend e backend:



```
Login = async (navigate) => {
            fetch('http://192.168.1.10:8080/ShoppingOnlineWS/
                resources/user?email=' + this.state.email + '&password=' +
                this.state.password, {
               method: 'get',
               headers: {
                 'Accept': 'application/json'
               },
             }).then(function(response) {
                 if (response.status === 200)
                     navigate('Home');

             }).catch((error)=>{
                 console.log("Api call error");
                 alert(error.message);
              });
}
```

4.3 Backend

Il server, al fine di connettersi al database, istanzia nella pagina "connector.php" un oggetto chiamato "**mysqli**", che apre una connessione verso il database per ogni richiesta (e la chiude in automatico al termine dell'operazione). Quest'operazione viene implementata nel seguente frammento di codice, che dovrà essere incluso in tutte le pagine .php che necessitano di connettersi al database per poter effettuare delle query.

```
<?php

    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "spesaonline";

    // Crea la connessione
    $conn = new mysqli($servername, $username, $password,
            $dbname);
    // Controlla se la connessione è andata a buon fine
    if ($conn->connect error)
        die("Connection failed: " . $conn->connect_error);

?>
```

Di seguito allego un'altra pagina .php, "visualizzaOrdini.php", che permette di **impaginare** in una tabella lo storico degli ordini di un utente. L'impaginazione consiste dunque nell'integrare all'interno della pagina html ciascun record risultante dalla query tramite un **ciclo di fetch**:

```php
<?php
    //controllo la sessione, per verificare che l'utente si sia prima
      autenticato
    require "checkSession.php";
    $IdUtente = $_SESSION["userID"];

    $query = $conn->prepare("SELECT * FROM Carrelli WHERE
            ordineEffettuato = '1' AND IdUtente = ?");
    $query->bind_param("i", $IdUtente);

    //effettuo la query al database, controllando che non ci siano stati
      errori di connessione con il db
    if ($query->execute() === TRUE) {
        //memorizzo i risultati della query da impaginare
        $result = $query->get_result();

        if ($result->num_rows == 0)
            echo "<h4>Non sono ordini effettuati</h4>";
        else {
            echo  "<table border='1'><tr><th>ID</th>
                  <th>stato ordine</th><th>data/ora acquisto</th>
                  <th>data/ora consegna</th>
                  <th>consegna a domicilio</th>
                  <th>richiesta fattura</th></tr>";

            //ciclo di fetch che va a impaginare ciascun record in una
            riga di una tabella
            while ($row = $result->fetch_assoc()) {
                $ID = $row['ID'];
                $dataOraAcquisto = $row['dataOraAcquisto'];
                $dataOraConsegna = $row['dataOraConsegna'];
                $consegnaDomicilio = $row['consegnaDomicilio'] == '1' ?
                   'SI' : 'NO';
                $fattura = $row['fattura'] == '1' ? 'SI' : 'NO';
                $stato = $row['statoOrdine'];
                $confermaCliente = $row['confermaCliente'];
                $azioneConferma = $confermaCliente == '1' ?
                  "<td>
                        <a href='gestisciOrdine.php?ID=$ID'>Gestisci
                        negoziazione ordine</a>
                  </td>" : "";

                echo
                  "<tr>
                        <td>$ID</td><td>$dataOraAcquisto</td>
                        <td>$dataOraConsegna</td>
                        <td>$consegnaDomicilio</td>
                        <td>$fattura</td><td>$stato</td>$azioneConferma
                   <tr>";
            }
            echo "</table>";
        }
    } else
        die("Impossibile connettersi al database");
?>
```