CLICKFOOD

Munfuletto Ilenia (0744613)

GENERAL DESCRIPTION:

Il Progetto di sistemi web richiesto consiste nell'implementazione di una versione base di un'applicazione per la gestione di ordini, la loro preparazione e consegna .L'app permette a chiunque la creazione di un ordine personalizzato, ed una volta creato, l'utente avrà a disposizione un codice per il ritiro/ricezione dell'ordine.

L'app deve anche permettere ad un cuoco loggato, di poter visualizzare gli ordini esistenti, contrassegnarli come "in lavorazione", come "domicilio" nel caso di consegna a domicilio o come "al banco", quest'ultimo ad indicarne la loro disponibilità al ritiro/consegna in negozio.

Un addetto consegna loggato invece, dopo aver visualizzato gli ordini "al banco", deve poter marcarli come "completato" una volta aver consegnato l'ordine al suo ordinante.

Un responsabile loggato, potrà avere acceso ad una pagina per il calcolo di statistiche relative all'attività, come il tempo medio di preparazione, di consegna, il numero di ordini complessivi o il numero di clienti registrati alla piattaforma.

Un cliente loggato, potrà invece ottenere un riepilogo degli ordini precedentemente effettuati, comprendente anche il suo codice di ritiro, ed effettuarne di nuovi.

Infine, un fattorino deve essere in grado di visualizzare gli ordini nello stato "domicilio" da dover consegnare. In caso di presa in carica dell'ordine, potrà marcarlo come "in consegna" e successivamente, come "consegnato".

SYSTEM ARCHITECTURE

Tecnologie utilizzate:

1. Classi Java servlet, per l'accettazione di richieste HTTP, per la costruzione delle relative pagine web di risposta e per il processo di validazione ed elaborazione dei dati inseriti dal client;

2. HTML e CSS per la realizzazione grafica dell'interfaccia utente, con framework Bootstrap 5.3.3;

3. JavaScript, in particolare JQuery 3.7.1, per associare gli eventi alle azioni dell'utente e per effettuare dei controlli di validazione, AJAX per effettuare le richieste HTTP, JSON per la gestione di oggetti scambiati tra client e server, Chart.js 2.9.4 per la creazione di grafici statistici e QRCode.js per la creazione di codice QR;

4. Apache Tomcat 10.1.2, quale middleware per il deployment della web application;

5. Database relazionale MySQL 8.0.28, gestito da MySQL Workbench, per la memorizzazione dei dati;

6. IntelliJ IDEA, come IDE per la scrittura del codice e la gestione delle risorse specificate precedentemente.

FUNCTIONAL REQUIREMENTS

Generali:

1. L'applicazione web sviluppata implementa un' app di gestione di ordini che permetterà ad un Cliente di accedere all'app, tramite apposito form di login (bottone "Login") presente nella navbar del sito (dopo essersi opportunamente iscritto con il form di iscrizione, col il bottone "Iscriviti"), e visualizzare gli ultimi ordini da lui effettuati, nella sezione "Area Riservata", ma anche di ordinare nella sezione "Ordina", in cui alcuni campi sono già precompilati per il cliente.
2. L'applicazione, effettuando il login come Cuoco, sempre tramite la navbar del sito, e la sua "Area Riservata", permetterà anche di gestire gli ordini "in preparazione", contrassegnandoli come tali, o se da consegnare a "domicilio" o "al banco", il tutto con appositi bottoni visualizzati sotto l'ordine.
3. L'applicazione, effettuando il login come Addetto, attraverso la navbar ("Area Riservata"), permetterà di cambiare stato all'ordine, da "al banco" a "completato".
4. Effettuando il login come Responsabile, quest'ultimo, sempre nella sua "Area Riservata" potrà visualizzare alcune statistiche di interesse, tramite grafici e valori significativi per l'attività.
5. Loggandosi come Fattorino, "Area Riservata", si potranno visualizzare gli ordini nello stato "domicilio" (cioè da dover consegnare), e si potrà far cambiare loro stato, passando a "in consegna" ed infine a "consegnato".
6. Chiunque invece, potrà accedere alla sezione pubblica "Ordina", collocata nella navbar, in cui poter comporre un ordine, specificando anche la modalità di pagamento e se il suddetto ordine, dovrà essere consegnato a domicilio o se ritirato fisicamente presso il negozio.

Client side:

1. Vista **index**: mostra la homepage del sito costituito da una navbar dalla quale è possibile loggarsi, registrarsi, navigare tra i contenuti della pagina ed accedere alla sezione di "Ordina". Inoltre, presenta un footer con link esterni per i social media.
2. Componente **Navbar** ("*Navbar.jsp*" nella cartella componenti): mostra le principali funzioni del sito, ossia "Home" per tornare alla index, "Ordina" per visualizzare la pagina per la creazione di un ordine, "Menù" come collegamento alla parte inferiore della index che mostra il menù. Sono poi presenti due bottoni, "Login" e "Iscriviti", rispettivamente per il login dell'utente e la registrazione del cliente, a seguito della quale verrà inviata una mail di benvenuto. La Navbar inoltre, presenta una sezione "Area Riservata" visibile ed accessibile solo in seguito ad un login, che reindirizza alla pagina personale in base al tipo di utente che si è loggato, mentre i bottoni di "Login" e "Iscriviti" vengono sostituiti dal bottone "Esci" che consente di effettuare il logout, invocando la servlet corrispondente.
3. Componente **Footer** ("*Footer.jsp*" nella cartella componenti): mostra il logo dell'app e presenta link per contatti e social.
4. Vista **ordina**: mostra la pagina per la creazione dell'ordine, costituita sempre dalla navbar, da un elenco di prodotti suddivisi in categorie e da un form per l'inserimento di informazioni relative alla consegna/ritiro, sulla scelta del metodo di pagamento e per l'inserimento di note opzionali per l'ordine. La pagina mostrerà il totale dell'ordine, sulla base dei prodotti selezionati e un bottone per l'invio dell'ordine alla fine della pagina. Se i campi inseriti sono validi verrà effettuata una richiesta post alla servlet corrispondente ed inviata un mail all'utente con il codice oppure segnalerà l'errore nell'input. In entrambi i casi, la risposta

visualizzata sarà un modale, che nel caso di buon esito della richiesta, comprenderà un QR Code del codice di ritiro.

5. Componente **Modal** ("*Modal.jsp*" nella cartella componenti): mostra il modale di notifica sull'esito dell'operazione corrente. Il modale di creazione dell'ordine, in caso positivo, mostra anche il QR Code del codice di ritiro, inviato anche per email.
6. Vista **errore**: mostra la pagina per gli errori 404 – Not Found o 500 – Internal Server Error, costituita sempre dalla navbar pubblica,dal footer, da una descrizione generica dell'errore verificatosi e da un bottone per tornare alla homepage dell'app.
7. Vista **Cuoco**: mostra la pagina riservata a seguito dell'accesso di un cuoco , costituita dalla navbar, compresa la sezione "Area Riservata" e dal footer. La pagina presenta una lista di ordini nello stato "da lavorare" o "in lavorazione", ognuno dei quali è identificato dal suo id, dall'elenco dei prodotti ordinati e la loro quantità. Sotto queste informazioni, sono specificate eventuali note per l'ordine, lo stato corrente dell'ordine (se ancora "da lavorare" o se già "in lavorazione") ed è presente anche un bottone per il cambio stato dell'ordine, passando a "in lavorazione" se era nello stato "da lavorare" oppure nello stato "al banco" o al "domicilio" se era nello stato "in lavorazione" in base al metodo di consegna dell'ordine. Il cambiamento di stato viene realizzato richiamando la servlet corrispondente. La pagina inoltre, esegue il reload automatico ogni 30s per aggiornare la lista degli ordini.
8. Vista **Addetto**: mostra la pagina riservata a seguito dell'accesso di un addetto, costituita dalla navbar con "Area Riservata" e dal footer. La pagina presenta una lista di ordini nello stato "al banco", ognuno dei quali è identificato dal suo id, dall'elenco dei prodotti ordinati e la loro quantità. Al di sotto di tali informazioni, ne sono specificate altre quali, il prezzo totale, la modalità di pagamento, l'indirizzo email dell'ordinante ed il QR Code del codice di ritiro da confrontare con quello fornito dal cliente. Anche in questo caso, è presente lo stato corrente dell'ordine ed un bottone per il cambio stato, che passerà dallo stato "al banco" a "consegnato". Il cambiamento di stato viene realizzato richiamando la servlet corrispondente. La pagina inoltre, esegue il reload automatico ogni 30s per aggiornare la lista degli ordini.
9. Vista **Responsabile**: mostra la pagina riservata a seguito dell'accesso di un responsabile, costituita dalla navbar e footer. La pagina presenta inoltre i grafici sul tempo medio preparazione e di consegna, le cui ascisse rappresentano gli stati degli ordini già completati o consegnati e le ordinate il tempo, espresso in secondi, che è stato richiesto per raggiungere l'i-esimo stato. La pagina inoltre fornisce dati relativi al tempo di attesa medio per la consegna dell'ordine, il totale degli ordini effettuati sulla piattaforma e il numero totale di clienti registrati.
10. Vista **Cliente**: mostra la pagina riservata a seguito dell'accesso di un cliente, costituita dalla navbar con "Area Riservata" e dal footer. La pagina presenta la lista di ordini effettuati dal cliente, ognuno dei quali è identificato dal suo id e dal riepilogo dell'ordine, ossia l'elenco dei prodotti ordinati con la loro quantità, il prezzo totale, l'indirizzo, il metodo di pagamento, la mail, la data ed il QR Code del codice di ritiro dell'ordine. Ancora più in basso è specificato lo stato corrente dell'ordine. Anche il cliente, può aver accesso alla sezione "Ordina" ma in questo caso le informazioni specifiche dell'utente verranno utilizzate per precompilare campi del form di ordinazione, quali il nome, il cognome e l'indirizzo mail, che rimarranno comunque modificabili dal cliente.
11. Vista **Fattorino**: mostra la pagina riservata a seguito dell'accesso di un fattorino , costituita dalla navbar, compresa la sezione "Area Riservata" e dal footer. La pagina presenta una lista di ordini nello stato "domicilio" o "in consegna", ognuno dei quali è identificato dal suo id, dall'elenco dei prodotti ordinati e la loro quantità. Sotto queste informazioni, sono specificati il prezzo totale, l'indirizzo, il metodo di pagamento, l'email dell'ordinante ed il QR Code del codice di ritiro. E' presente anche lo stato corrente dell'ordine (se ancora da consegnare a

"domicilio" o se già "in consegna") ed è presente anche un bottone per il cambio stato dell'ordine, passando a "in consegna" se era nello stato "domicilio" oppure nello stato "consegnato" se era nello stato "in consegna". Il cambiamento di stato viene realizzato richiamando la servlet corrispondente. La pagina inoltre, esegue il reload automatico ogni 30s per aggiornare la lista degli ordini.

12. Vista **recuperoPassword**: mostra la pagina pubblica per il recupero della password, costituita dalla navbar,il footer, un campo input per l'inserimento della mail di cui non si conosce più la password ed un bottone sottostante per l'invio di un link di recupero alla mail fornita. Se quest'ultima non esiste o non è valida, si verrà inoltrati alla pagina di index, altrimenti verrà inviata effettivamente la mail contenente un link diretto alla vista di "*cambiaPassword.jsp*".
13. Vista **cambiaPassword**: mostra la pagina privata per il cambio password con due campi input per l'inserimento della nuova password ed un bottone per richiamare la servlet corrispondente ed effettuare la modifica.
14. Componente head delle pagine: contiene tutti i tag necessari a importare stili CSS, Bootstrap e personalizzati compresi nel file ("*style.css*") e le librerie JavaScript di jQuery, Chart.js e QRCode.js .
15. Validazione client-side (tramite attributi HTML e jQuery).
16. Tecnologie:
    - HTML
    - CSS
    - JavaScript
    - jQuery
    - AJAX
    - JSON
    - Framework Bootstrap per stili e script dell'interfaccia
    - Chart.js per la generazione di grafici
    - QRCode.js per la generazione degli stessi

Server side:

Il model (*il package '**model**'*) sarà suddiviso nelle seguenti classi Java Bean:

- **Utente**, racchiude le informazioni di un utente generico dell'applicazione.
- **Cliente**, estende Utente e aggiunge informazioni anagrafiche sul cliente.
- **Ordine**, racchiude le informazioni più generali di un ordine.
- **DettagliOrdine**, racchiude le informazioni specifiche di un prodotto in un ordine.
- **Prodotto**, racchiude le informazioni di un generico prodotto acquistabile dell'utente.
- **TempisticheOrdine**, racchiude il timestamp in cui un ordine entra in un certo stato.

Le utils (*il package '**utils**'*) sono varie classi Java ausiliarie utilizzate per il funzionamento dell'applicazione, tra cui quelle per la connessione al database e la cifratura delle password. Tra queste abbiamo:

1. Classe **Connessione**: inizializza la connessione con il database tramite parametri contestuali specificati all'interno del file *context.xml*, tra cui la classe del connettore JDBC com.mysql.cj.jdbc.Driver (8.0.28) importata tramite dipendenza Maven.
2. Classe **Query**: specifica le query da eseguire ed i metodi per la gestione dei risultati.

3. Classe **SHA256**: utilizzata per la cifratura delle password degli utenti e la generazione del codice di ritiro (come digest della stringa "ORDER-id_ordine").
4. Classe **EmailSender**: utilizzata per l'invio delle mail di registrazione, creazione dell'ordine e recupero della password. Sfrutta le classi dei package javax.mail (1.6.2) e javax.activation (1.1.1) importate tramite dipendenze Maven.

Le servlet (*il package '**control**'*) gestiscono le richieste dell'utente, ponendosi da ponte tra la logica di business e le viste. Si distinguono tra:

1. **LoginServlet**: controller adibito alla gestione del login degli utenti. In particolare:
   - Metodo *doGet*: reindirizza alla homepage del sito, cioè l'index.
   - Metodo *doPost*: gestisce la richiesta di login, effettuando ulteriori validazioni sull'input. In caso positivo, controlla l'effettiva esistenza dell'utente intenzionato a loggarsi nel database con la rispettiva password. Segnala possibili errori nel caso non esistesse un utente con quella mail e password, reindirizzando all'index, oppure aggiunge in sessione l'utente, reindirizzandolo verso la sua pagina riservata in base alla tipologia di utente. Più precisamente, viene invocata la servlet corrispondente al tipo, che si occuperà del reindirizzamento. In caso invece, di altri errori imprevisti viene settato uno stato 500 e mostrata la pagina di errore.
2. **LogoutServlet**: controller adibito alla gestion del logout degli utenti. In particolare:
   - Metodo *doGet*: invalida la sessione e reindirizza all'index.
   - Metodo *doPost*: richiama il metodo doGet().
3. **RegistrazioneServlet**: controller adibito alla gestione della registrazione degli utenti sul sito.
   - Metodo *doPost*: gestisce la richiesta di iscrizione, validando i parametri della richiesta ed effettuando opportuni controlli. In un primo momento, verifica se l'email utilizzata non sia già registrata, se si, setta un codice di errore e un testo di errore da mostrare in un modale. Altrimenti, viene effettivamente inserito il nuovo cliente nel database, inviata una mail di conferma all'utente e settato un codice di successo e corrispondente testo da mostrare in seguito sempre in un modale. In caso di errori dovuti alla query, verrà settato un differente codice e testo di errore per il modale, oppure in caso di errori generici, settato uno stato 500 e mostrata la pagina di errore.
4. **OrdinaServlet**: controller adibito alla gestione e creazione degli ordini degli utenti. Si distingue tra:
   - Metodo *doGet*: si occupa di recuperare i prodotti necessari per popolare la pagina della creazione dell'ordine. Una volta aver ottenuto questi dati, li inserisce in sessione e reindirizza alla pagina di ordinazione che si occuperà di mostrare tali dati.
   - Metodo *doPost*: si occupa dell'inserimento dell'ordine nel database. Processa e valida i parametri in richiesta, settando un stato di errore 500 e reindirizzando all'index nel caso in cui anche uno solo di questi parametri non rispetti il proprio pattern. Altrimenti, una volta aver ottenuto tramite query, il valore dell'id_Ordine dell'ultimo ordine presente nel database, aggiunge l'ordine corrente incrementando tale valore di 1. L'inserimento consiste nella chiamata ad un metodo nella classe Query, che si occuperà prima dell'inserimento dell'ordine e successivamente dell'inserimento dei prodotti che ne fanno parte. In caso di esito positivo di tutte queste operazioni, viene inviata una mail all'utente con il QR Code del suo codice di ritiro e settati in sessione un codice, testo e lo stesso QR Code per mostrare inoltre un modale all'ordinante. Ulteriore operazione eseguita, è l'inserimento di un timestamp con lo stato iniziale "da lavorare" dell'ordine. Nel caso l'operazione di inserimento dell'ordine fallisse,

verrà settato un codice e testo di errore da mostrare tramite modale, invece in caso di errori generici, sarà settato lo stato 500 e reindirizzato l'utente alla pagina di errore.

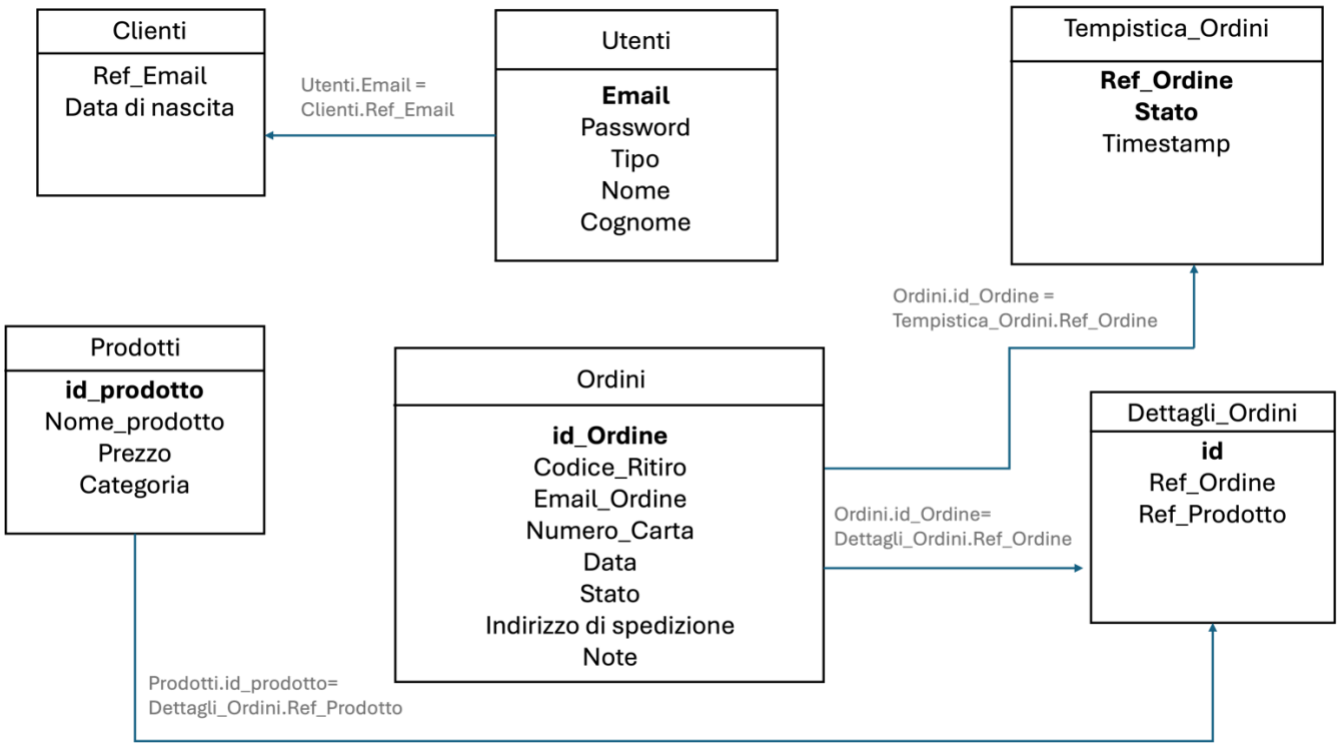
5. **CuocoServlet**: controller adibita alla gestione delle operazioni riguardanti un generico cuoco loggato. Presenta:
   - Metodo *doGet*: si occupa di controllare l'effettiva presenza di un utente di tipo Cuoco in sessione, se no, reindirizza all'index. Se in sessione è presente un cuoco, il metodo recupera gli ordini da mostrare al cuoco, cioè gli ordini in stato "da lavorare" o "in lavorazione", e dopo averli messi in sessione, reindirizza alla pagina privata del cuoco, dove questi ordini verranno mostrati.
   - Metodo *doPost*: si occupa di gestire la richiesta di cambiamento stato del cuoco, settando il nuovo stato a "in lavorazione" o "domicilio" in base allo stato corrente dell'ordine e in base alla modalità di consegna dell'ordine. Il metodo invoca un metodo della classe Query, passando l'id e il nuovo stato dell'ordine come parametri. In caso di errori viene impostato un testo e un codice di errore da mostrare in un modale al cuoco.
6. **AddettoServlet**: controller adibita alla gestione delle operazioni riguardanti un addetto loggato. Presenta:
   - Metodo *doGet*: controlla l'effettiva presenza di un utente di tipo Addetto in sessione, se no, reindirizza all'index. Se in sessione è presente un addetto, il metodo recupera gli ordini da mostrargli, cioè gli ordini nello stato "al banco", e dopo averli messi in sessione, reindirizza alla pagina riservata dell'addetto dove questi ordini verranno mostrati.
   - Metodo doPost: si occupa di gestire la richiesta di cambiamento stato dell'addetto, settando il nuovo stato a "completato". Il metodo invoca un metodo della classe Query, passando l'id e il nuovo stato dell'ordine come parametri. In caso di errori viene impostato un testo e un codice di errore da mostrare in un modale all'addetto.
7. **ResponsabileServlet**: controller adibita alla preparazione delle informazioni per la dashboard del responsabile.
   - Metodo *doGet*: controlla l'effettiva presenza di un utente di tipo Responsabile in sessione, se no, reindirizza all'index. Se in sessione è presente un responsabile, il metodo recupera varie informazioni statistiche per popolare i grafici e le sezioni della sua area riservata. Una volta aver ottenuto tali dati, li aggiunge in sessione e reindirizza alla pagina riservata del responsabile.
   - Metodo *doPost*: richiama il metodo doGet().
8. **ClienteServlet**: controller adibita a mostrare all'utente gli ordini da lui effettuati.
   - Metodo *doGet*: controlla l'effettiva presenza di un utente di tipo Cliente in sessione, se no, reindirizza all'index. Se in sessione è presente un cliente, il metodo recupera i suoi ordini dal database, li salva in sessione e lo reindirizza alla sua pagina riservata, dove i suoi ordini verranno mostrati.
   - Metodo *doPost:* richiama il metodo doGet().
9. **FattorinoServlet**: controller adibita alla gestione delle operazioni riguardanti un fattorino loggato. Si distingue in :
   - Metodo *doGet*: controlla l'effettiva presenza di un utente di tipo Fattorino in sessione, se no, reindirizza all'index. Se in sessione è presente un fattorino, il metodo recupera gli ordini da mostrargli, cioè gli ordini in stato "domicilio" o "in consegna, e dopo averli messi in sessione, reindirizza alla pagina privata del fattorino, dove questi ordini verranno mostrati.
   - Metodo *doPost*: si occupa di gestire la richiesta di cambiamento stato del fattorino, settando il nuovo stato a "in consegna" o "consegnato" in base allo stato corrente

dell'ordine. Il metodo invoca un metodo della classe Query, passando l'id e il nuovo stato dell'ordine come parametri. In caso di errori viene impostato un testo e un codice di errore da mostrare in un modale al fattorino.

10. **RecuperoPasswordServlet**: controller adibita alla gestione delle procedure per il recupero e ripristino della password. Si differenzia tra:
    - Metodo *doGet*: effettua una validazione sulla mail inserita, invocando un metodo della classe Query per la ricerca di un utente tramite email. Se il metodo trova l'utente, allora viene controllata la presenza o meno del parametro "codice=recupero", presente nella url inviato all'utente tramite mail. Se non esiste, viene effettivamente inviata l'email con il link e impostato un codice e testo da mostrare in un modale, per poi infine reindirizzare all'index. Altrimenti, il metodo reindirizza al cambiaPassowrd.jsp. In caso di qualunque tipo di errore, come la non presenza dell'utente nel database, si viene reindirizzati all'index.
    - Metodo *doPost*: valida i parametri della richiesta HTTP ed effettua il cambio password. In caso di successo, viene impostato un codice e testo per un modale che suggerisce l'esito positivo dell'operazione, altrimenti si visualizzerà un codice e testo di errore, sempre in un modale.
11. Tecnologie:
    - Java Servlet
    - JSP
    - JavaMail API
    - Libreria Java JSON
    - Libreria Java QR Code

DATA MODEL

Schema logico per il database.



N.B : in **grassetto** le chiavi primarie di ogni tabella.