Report

1 Django

1.1 Introduzione

Django è un Web Framework di Python ad alto livello, open source. Si occupa principalmente del problema dello sviluppo web, permettendo all'utente di focalizzarsi sulla scrittura dell'applicazione. Segue il paradigma Model-Template-View, che segue il pattern MVC (Model-View-Controller). Questo pattern separa la rappresentazione del modello dei dati, dell'interfaccia utente e della logica applicativa, detta anche logica di business.

1.2 Caratteristiche

Le carattestiche principali del Web Framework Django sono:

- Veloce: è stato progettato per aiutare gli sviluppatori ad implementare applicazioni nel modo più veloce possibile.
- Completo: include molte funzionalità extra che si possono usare per gestire i task di sviluppo Web più comuni, ad esempio: autenticazione dell'utente, amministrazione dei contenuti e molte altre.
- Sicuro: aiuta gli sviluppatori ad evitare molti problemi comuni di sicurezza, come l'SQL Injection.
- Scalabile.
- Versatile.

Alcuni esempi di siti che utilizzano Django sono: Instagram, Mozilla, Pinterest e National Geographic.

1.3 Tutorial

Per sviluppare l'applicazione, è stato necessario, inizialmente, seguire un tutorial specifico, consultabile al link.

1.3.1 Request e Response

Django utilizza gli oggetti Request e Response per passare lo stato del sistema all'interno dell'applicazione. Quando viene richiesta una pagina, Django crea un oggetto HttpRequest che contiene i metadati della richiesta. In seguito, viene caricata la View appropriata, passando HttpRequest come primo argomento della funzione view. Ogni View è responsabile del ritorno di un oggetto HttpResponse.

Gli oggetti HttpRequest e HttpResponse sono definiti nel modulo django.http.

Gli oggetti HttpRequest sono creati automaticamente da Django, mentre gli oggetti HttpResponse sono responsabilità dello sviluppatore: ogni vista che viene scritta è responsabile per istanziare, popolare e ritornare un oggetto HttpResponse.

1.3.2 Model

Un modello è una sorgente di informazione riguardo i dati e contiene i campi e le funzioni essenziali dei dati che si stanno salvando. In genere, ogni modello si mappa in una singola tabella del database.

Ogni modello è una classe di Python che è sottoclasse di django.db.models.Model e ogni attributo del modello rappresenta un campo del database. Django fornisce API di accesso al database generate automaticamente.

1.3.2.1 QuerySet

Dopo aver creato i data model, Django fornisce automaticamente le API di astrazione del database che consentono la creazione, recupero, aggiornamento e cancellazione degli oggetti.

Una classe di modello rappresenta una tabella del database e un'istanza di quella classe rappresenta un particolare record nella tabella del database. Per creare un oggetto, è necessario istanziarlo usando argomenti chiave della classe di modello e poi usare il metodo *save()* per memorizzarlo nel database. Questo metodo si riferisce all'istruzione SQL di INSERT.

Per memorizzare i cambiamenti di un oggetto che è già presente nel database viene usato ancora il metodo *save()*. Questo metodo si riferisce all'istruzione SQL di UPDATE.

Per recuperare gli oggetti dal database, è necessario costruire un QuerySet sulla classe di modello. Un QuerySet rappresenta una collezione di oggetti provenienti dal database. Può avere nessuno, uno o tanti filtri. I filtri restringono i risultati della query basandosi su determinati parametri. Nella notazione SQL, un QuerySet si riferisce all'istruzione SELECT e un filtro è una clausola limitante, come WHERE o LIMIT.

Per eliminare un oggetto e ritornare il numero di oggetti eliminati, si utilizza il metodo *delete()*. Ogni QuerySet ha questo metodo che elimina tutti i membri di quel particolare QuerySet. Quando Django elimina un oggetto, da default simula il comportamento del vincolo SQL *ON DELETE CASCADE*, ovvero ogni oggetto che ha le chiavi esterne che puntano all'oggetto che dev'essere eliminato, sarà eliminato assieme ad esso. Questo metodo si riferisce all'istruzione SQL di DELETE.

1.3.2.2 RawQuerySet

Django fornisce un altro modo per l'esecuzione di operazioni di tipo SELECT sul database, ovvero utilizzando raw queries.

Il metodo *raw()* può essere usato a questo scopo. Prende come parametro la query SQL, la esegue e ritorna un'istanza di django.db.models.query.RawQuerySet. Questa istanza può iterata come una normale QuerySet per fornire istanze dell'oggetto.

Per le operazioni di UPDATE, INSERT o DELETE si può accedere al database direttamente, aggirando completamente il livello model. L'oggetto django.db.connection rappresenta la connessione di default al database. Per utilizzare la connessione al database, viene usato il metodo *cursor()* per ottenere un oggetto cursore. Poi, si richiama il metodo *execute(sql, [params])* per eseguire la query SQL.

1.3.2.3 Migrations

Le migrations sono il modo in cui Django propaga i cambiamenti che vengono fatti ai modelli (come l'aggiunta di un campo, l'eliminazione di un modello, ecc…) nello schema del database. Sono progettate per essere quasi del tutto automatiche, ma è necessario che lo sviluppatore decida quando eseguire le migrations e conosca i problemi comuni che potrebbe incontrare.

Le migrations sono supportate su tutti i backend con cui Django viene fornito e anche su backend di terzi se compatibili. Alcuni database sono più capaci di altri quando si tratta di migrazioni di schema, ad esempio: PostgreSQL, MySQL e SQLite.

1.3.3 View

Django usa il concetto di views per incapsulare la logica responsabile per processare una richiesta dell'utente e per ritornare la risposta.

1.3.3.1 URL dispatcher

Per progettare gli URL per un'applicazione, bisogna creare un modulo in Python chiamato URLconf (URL configuration). Questo modulo è un mapping tra le espressioni dei path URL e le funzioni in Python (corrispondenti alle views). Questo mapping può essere costruito dinamicamente.

Ogni espressione negli urlpattern è compilata la prima volta in cui avviene l'accesso e questo rende il sistema molto veloce.

1.3.3.2 View Function

Una view function è una funzione Python che prende una Web Request e ritorna una Web Response. La response può essere un contenuto HTML di una pagina web, un redirect, un errore 404, un documento XML, un'immagine, o qualsiasi altra cosa.

La funzione contiene qualsiasi logica arbitraria che sia necessaria per ritornare quella response. Per convenzione, le views vengono inserite in un file chiamato views.py, nella cartella di progetto.

1.3.3.3 Shortcut Functions

Il package django.shortcuts raccoglie le funzioni di supporto e le classi che attraversano livelli multipli dell'MVC.

La funzione *render()* combina un dato template con un dato context dictionary e ritorna un oggetto HttpResponse con il testo renderizzato.

Gli argomenti richiesti sono:

- request: oggetto utilizzato per generare questa response.
- template_name: nome intero del template da usare o una sequenza di nomi di template.

La funzione *redirect()* ritorna un HttpResponseRedirect all'URL appropriato agli argomenti passati, che potrebbero essere: un model, un nome di view, un URL assoluto o relativo.

1.3.4 Template

Il livello template fornisce una sintassi designer-friendly per effettuare il render dell'informazione che dev'essere mostrato all'utente.

Un template contiene sia le parti statiche dell'output HTML desiderato sia una sintassi speciale che descrive come inserire contenuti dinamici.

2 IDE

Per lo sviluppo dell'applicazione è stato utilizzato PyCharm fornito da JetBrains. È un ambiente di sviluppo che comprende un editor e vari tool di analisi, test e debug del codice.

Le caratteristiche principali sono:

- Assistenza e analisi della codifica.
- Navigazione di progetto e codice.
- Refactoring.
- Supporto nativo per Web Framework.
- Debugger integrato.
- Unit test integrato con copertura del codice riga per riga.
- Controllo di versione integrato.

3 Scenario

Lo scenario prevede lo sviluppo di una web application per la gestione di un cocktail bar.

Il sistema fornisce una fase di registrazione e una fase di autenticazione.

La fase di registrazione permette ad un utente generico di registrarsi nel sistema fornendo Nome, Cognome, Email, Telefono, Password.

La fase di autenticazione avviene tramite Email e Password e identifica due tipi di utente: Barista e Cliente.

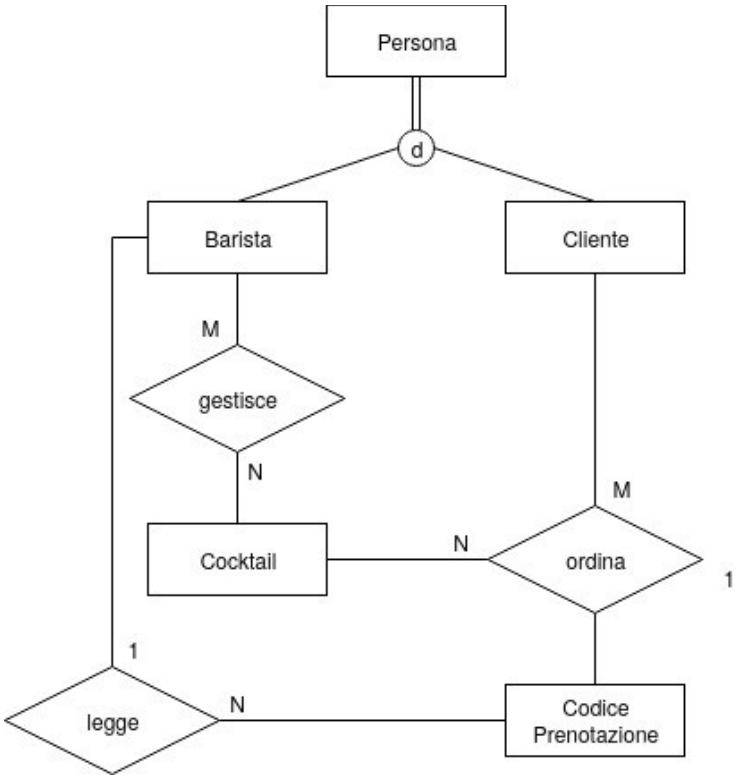
Il Barista deve essere in grado di:

- Inserire un nuovo cocktail nel menù, specificando Nome, Ingredienti e Prezzo.
- Modificare un cocktail già esistente, specificando i nuovi Ingredienti, il nuovo Prezzo o entrambi.
- Eliminare un cocktail già esistente.
- Controllare i Codici di Prenotazione, una volta che l'ordine viene preparato.

Il Cliente deve essere in grado di:

- Fare l'ordinazione, dopo la cui conferma viene rilasciato un Codice di Prenotazione che verrà visualizzato dal Barista in fase di preparazione.
- Visualizzare lo storico degli ordini.

4 Diagramma EER



In figura è mostrato il diagramma EER relativo allo scenario descritto nel capitolo precedente.

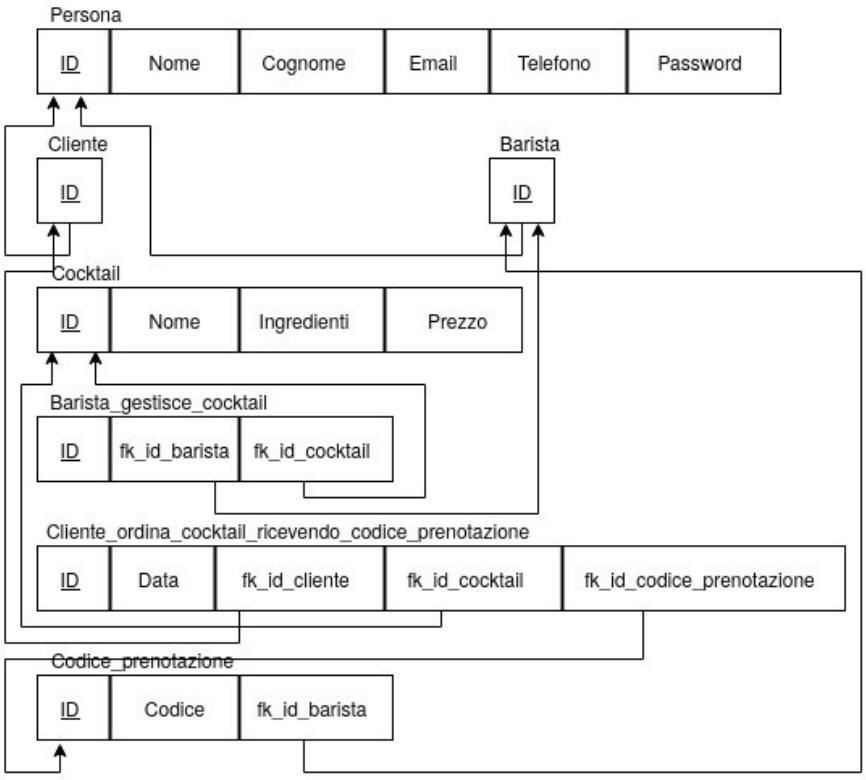
Le entità Barista e Cliente sono specializzazioni di tipo disjoint dell'entità Persona.

La relazione Barista gestisce Cocktail ha cardinalità M:N, ovvero un barista può gestire N cocktail e un cocktail può essere gestito da M baristi.

La relazione Barista legge Codice Prenotazione ha cardinalità 1:N, ovvero un barista può leggere N codici di prenotazione e un codice prenotazione è letto da un barista.

La relazione Cliente ordina Cocktail ricevendo Codice Prenotazione è ternaria, ovvero il cliente può ordinare N cocktail ricevendo un codice di prenotazione e un cocktail può essere ordinato da M clienti e essere associato ad un codice di prenotazione.

5 Mapping Relazionale



In figura è mostrato il mapping relazionale relativo al diagramma EER descritto nel capitolo precedente.

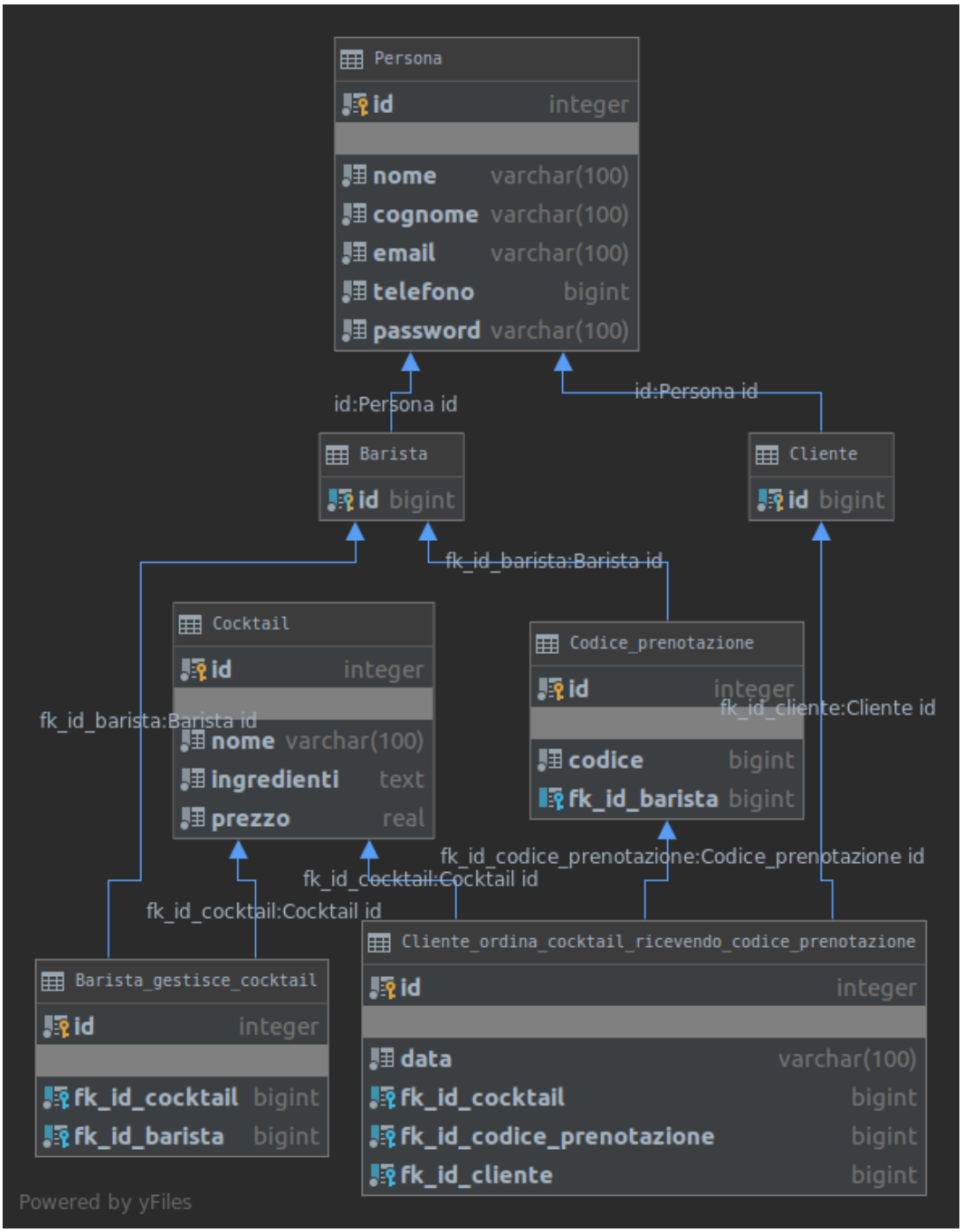
Le chiavi primarie ID delle entità Cliente e Barista sono anche chiavi esterne, relative all'ID (chiave primaria) dell'entità Persona.

La relazione Barista_gestisce_cocktail ha cardinalità N:M, quindi viene tradotta in una tabella che contiene le chiavi primarie delle entità Barista e Cocktail come chiavi esterne.

La relazione Cliente_ordina_cocktail_ricevendo_codice_prenotazione è ternaria e viene tradotta in una tabella che contiene le chiavi primarie delle entità Cliente, Cocktail e Codice_prenotazione, oltre all'attributo Data.

La relazione Barista legge Codice_prenotazione ha cardinalità 1:N e viene mappata all'interno della tabella Codice_prenotazione che contiene la chiave primaria dell'entità Barista come chiave esterna.

6 Descrizione Database

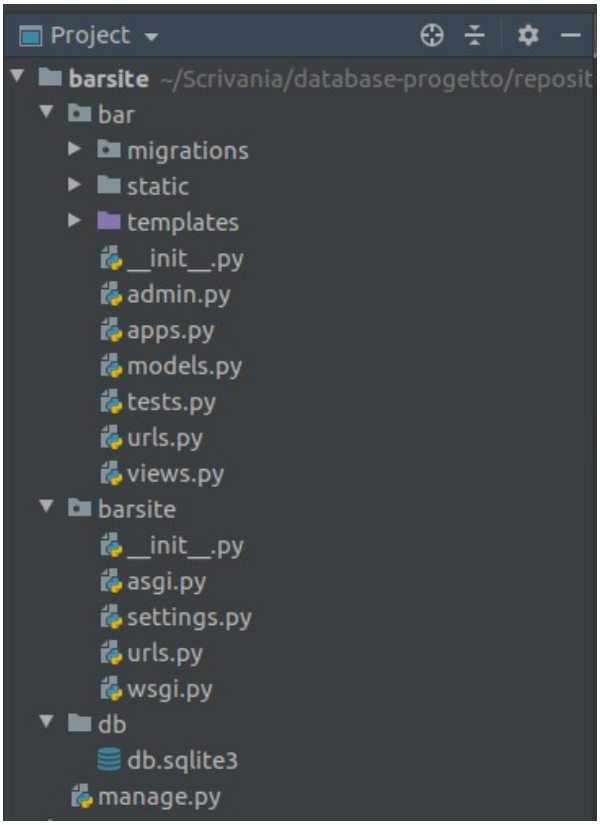


In figura, viene riportato il diagramma delle classi del database estratto grazie alle funzionalità dell'IDE utilizzato.

Vengono messi in risalto gli attributi e i relativi tipi di ogni tabella e i vincoli di integrità referenziale.

7 Descrizione Progetto

7.1 Struttura



In figura, viene riportata la struttura del progetto *barsite*. All'interno della cartella di progetto, sono presenti le directory:

- *bar*: è l'applicazione creata per la gestione del bar. Al suo interno, contiene le seguenti directory e files:
  - migrations: directory contenente i file che aggiornano la struttura del database, se richiesto.
  - static: directory contenente la libreria bootstrap, le immagini e le gif utilizzate all'interno del sito, e file .css di tipo stylesheet che specificano il formato di rappresentazione delle pagine .html.
  - template: directory contenente i file .html visualizzati all'interno dell'applicazione.
  - __init__.py: attraverso il quale Python comprende che l'app *bar* è un package Python.
  - admin.py: in cui vengono registrati i modelli dell'app con l'applicazione admin di Django (non utilizzato in questo progetto).
  - apps.py: file di configurazione comune a tutte le applicazioni Django.
  - models.py: in cui vengono dichiarati i modelli.
  - tests.py: contenente le prodedure di test utilizzate durante il testing dell'applicazione (non utilizzato in questo progetto).
  - views.py: in cui vengono dichiarate le views.

    - urls.py: contenente le dichiarazioni URL dell'applicazione.
- *barsite*: è l'applicazione che Django crea in automatico dopo la creazione del progetto. Al suo interno, contiene i seguenti files:
    - __init__.py, attraverso il quale Python comprende che si tratta di un package Python.
    - asgi.py: entry-point che contiene i web server per lanciare il progetto.
    - settings.py: configurazione di progetto.
    - urls.py: contenente le dichiarazioni URL del progetto.
    - wsgi.py: entry-point che contiene i web server per lanciare il progetto.
- *db*: all'interno è presente il database db.sqlite3, utilizzato dall'applicazione per eseguire le operazioni CRUD e creato quando viene lanciato il comando migrate.

e lo script *manage.py*, che è un'utility a linea di comando per eseguire i comandi di Django all'interno del progetto.

7.2 Funzionalità

7.2.1 Homepage

L'Homepage dell'applicazione web fornisce un elenco dei cocktail presenti nel menù, a scopo di consultazione.

In questa schermata l'utente ha la possibilità di effettuare:

- la registrazione al sito, prevista per i soli clienti che vogliono interagire con l'applicazione. L'utente viene quindi inserito nelle tabelle del database Persona e Cliente;
- il login al sito, previsto sia per i clienti sia per i baristi. La funzione di login permette l'individuazione del tipo di utente effettuando una SELECT sulle tabelle Cliente e Barista, controllando che l'id assegnato faccia riferimento ad una o all'altra tabella. La funzione di logout, presente in ogni pagina dopo l'autenticazione, consente di effettuare un redirect sulla Homepage del sito.

7.2.2 Cliente

Dopo l'autenticazione, il Cliente visualizza la propria area riservata, da cui può scegliere di:

- effettuare un ordine, scegliendo uno o più cocktail dal menù. Alla fine, viene rilasciato un codice di prenotazione di quattro cifre, generato in maniera casuale, relativo all'ordinazione. In questa fase, viene inserito un record nella tabella Cliente_ordina_cocktail_ricevendo_codice_prenotazione;
- visualizzare lo storico delle sue ordinazioni. Viene, quindi, eseguita una SELECT sulla tabella Cliente_ordina_cocktail_ricevendo_codice_prenotazione, filtrando i record tramite l'id del Cliente. Una volta ottenuti i risultati richiesti, viene effettuata una nuova SELECT sulla tabella Cocktail per risalire al nome dei cocktail ordinati tramite l'id corrispondente.

7.2.3 Barista

Dopo l'autenticazione, il Barista visualizza la propria area riservata, da cui può scegliere di:

- inserire i cocktail nel menù, definendo nome, ingredienti e prezzo. In questa fase, viene effettuato un INSERT nella tabella Cocktail e un INSERT nella tabella Barista_gestisce_cocktail, specificando l’azione ‘Inserimento’;
- modificare i cocktail, aggiornando gli ingredienti, il prezzo o entrambi. In questa fase, viene effettuato un UPDATE sul record corrispondente nella tabella Cocktail e un UPDATE nella tabella Barista_gestisce_cocktail, modificando l’azione ‘Inserimento’ in ‘Modifica’;
- eliminare i cocktail, eseguendo un DELETE del record corrispondente nella tabella Cocktail;
- controllare i codici di prenotazione, per poter preparare i cocktail corrispondenti. In questo caso, viene effettuata una SELECT sulla tabella Cliente_ordina_cocktail_ricevendo_codice_prenotazione, filtrando i record tramite l’id del Codice_prenotazione. Una volta ottenuti i risultati richiesti, viene effettuata una nuova SELECT sulla tabella Cocktail per risalire al nome e agli ingredienti dei cocktail ordinati tramite l’id corrispondente.