



**Università  
degli Studi  
di Ferrara**

## **DIPARTIMENTO DI INGEGNERIA**

*Corso di Laurea in Ingegneria Informatica e dell'Automazione*

### ***Progetto di Ingegneria del Software Avanzata***

*Professor Alberti Marco  
Professor Azzolini Damiano*

Alessandro Bertelli

*Anno Accademico 2021/22*

# Contents

1. Introduzione
2. Descrizione del database
  - Analisi dei requisiti con descrizione del mini-mondo
  - Progettazione dello schema ER/EER
  - Schema Relazionale
3. Requisiti di sistema
  - Controllo di versione: *Git/GitHub/GitKraken*
  - Gestione delle dipendenze: *conda*
  - Suite di test automatizzati: *pytest-flask*
4. Statechart

# Introduzione

Lo sviluppo di questo progetto è finalizzato al soddisfacimento dei requisiti richiesti per quanto concerne i corsi di Ingegneria del Software Avanzata e di Tecnologie per le Basi di Dati.

I requisiti specifici richiesti per Tecnologie per le basi di dati sono:

- realizzazione di un'applicazione che utilizza un database,
- utilizzo di linguaggio di programmazione a scelta,
- presentazione dello schema logico del database,
- il database deve contenere almeno 5 tabelle,
- l'applicazione deve fare operazioni di INSERT, UPDATE, DELETE e almeno un JOIN,
- il DBMS deve essere SQL Server, Oracle, DB2 o postgres (non si può utilizzare mySQL o Access).

L'applicazione da me sviluppata è una web app Flask che si interfaccia con un database postgres per gestire le prenotazioni di una pizzeria. Tramite essa infatti, un cliente potrà visualizzare il menu della pizzeria, effettuare le proprie prenotazioni, modificarle o eliminarle. Ho utilizzato psycopg2 per realizzare la connessione tra la web app Flask (sviluppata in python con l'utilizzo di html, css e javascript lato web) e il database postgres. Inoltre ho utilizzato DBeaver come strumento di amministrazione database.

Per quanto riguarda Ingegneria del Software Avanzata, il progetto ha lo scopo di verificare la capacità di condurre un progetto software utilizzando le tecniche e gli strumenti proposti nel corso, con conseguenti sviluppo e discussione del progetto.

Per quanto riguarda l'integrazione di questi requisiti nel mio progetto ho quindi utilizzato:

- **Git, GitHub, GitKraken:** per il controllo di versione,
- **conda:** package manager open source (utilizzato per la gestione delle dipendenze),
- **pytest-flask:** framework di test per la creazione di una suite di test automatizzati.

## Descrizione del database

### Analisi dei requisiti con descrizione del mini-mondo

La base di dati PIZZERIA tiene traccia di tutte le prenotazioni di una pizzeria per asporto, dei menu delle pizze e delle bevande, dei clienti e dei coupon (buoni sconto) messi a disposizione dalla pizzeria.

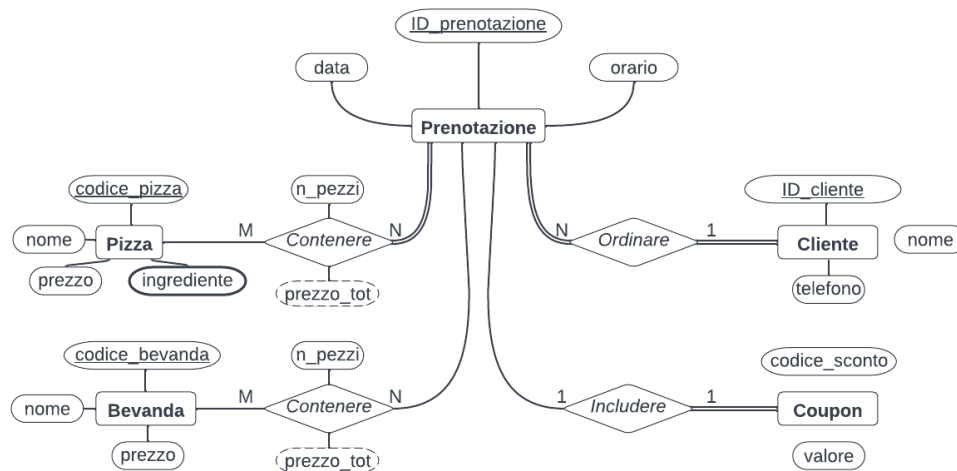
Ogni prenotazione ha un codice univoco, un cliente associato, un orario di riferimento e la lista dei codici dei prodotti (pizze e bevande) desiderati. I prodotti sono di due tipi: pizze o bevande.

Ogni pizza ha un codice univoco, un nome, (la lista di ingredienti) e un prezzo. Ogni bevanda ha un codice univoco, un nome e un prezzo. Ogni cliente è identificato da un ID cliente univoco che sarà associato al nome e, in modo facoltativo, al numero di telefono del cliente. Inoltre la pizzeria in particolari occasioni rilascia dei coupon a qualche cliente permettendogli di avere diversi tipi di sconto.

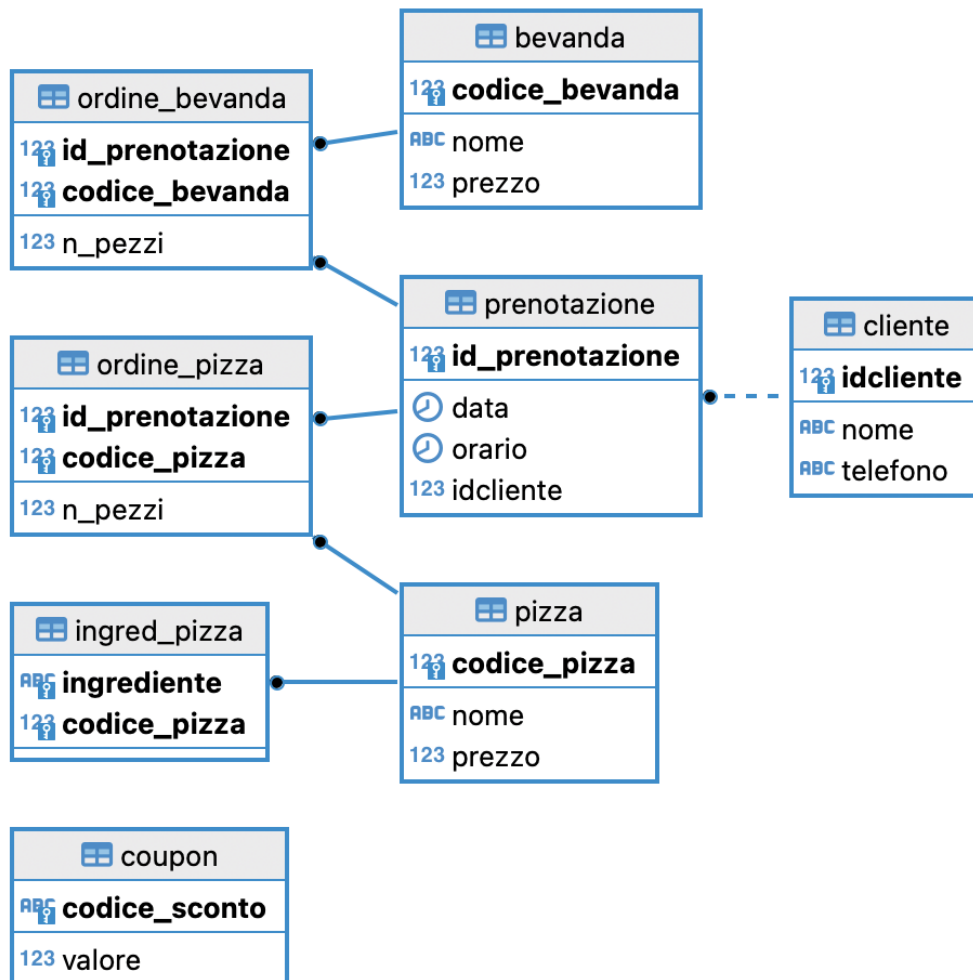
Ogni coupon ha un codice univoco identificativo e un valore in euro pari al valore dello sconto applicabile (chiaramente dopo essere stato utilizzato viene eliminato dal database, i coupon non sono cumulabili).

### Progettazione dello schema ER/EER

#### Schema ER non normalizzato



Schema ER normalizzato - *autogenerato da DBeaver*



## Schema Relazionale

Normalizzazione in Terza Forma Normale (3NF)

### PRENOTAZIONE

|                        |      |        |                  |
|------------------------|------|--------|------------------|
| <u>ID_PRENOTAZIONE</u> | DATA | ORARIO | <u>IDCLIENTE</u> |
|------------------------|------|--------|------------------|

### CLIENTE

|                  |      |          |
|------------------|------|----------|
| <u>IDCLIENTE</u> | NOME | TELEFONO |
|------------------|------|----------|

### COUPON

|                      |        |
|----------------------|--------|
| <u>CODICE_SCONTO</u> | VALORE |
|----------------------|--------|

### PIZZA

|                     |      |        |
|---------------------|------|--------|
| <u>CODICE_PIZZA</u> | NOME | PREZZO |
|---------------------|------|--------|

### ORDINE\_PIZZA

|                        |                     |         |
|------------------------|---------------------|---------|
| <u>ID_PRENOTAZIONE</u> | <u>CODICE_PIZZA</u> | N_PEZZI |
|------------------------|---------------------|---------|

### INGRED\_PIZZA

|                    |                     |
|--------------------|---------------------|
| <u>INGREDIENTE</u> | <u>CODICE_PIZZA</u> |
|--------------------|---------------------|

### BEVANDA

|                       |      |        |
|-----------------------|------|--------|
| <u>CODICE_BEVANDA</u> | NOME | PREZZO |
|-----------------------|------|--------|

### ORDINE\_BEVANDA

|                        |                       |         |
|------------------------|-----------------------|---------|
| <u>ID_PRENOTAZIONE</u> | <u>CODICE_BEVANDA</u> | N_PEZZI |
|------------------------|-----------------------|---------|

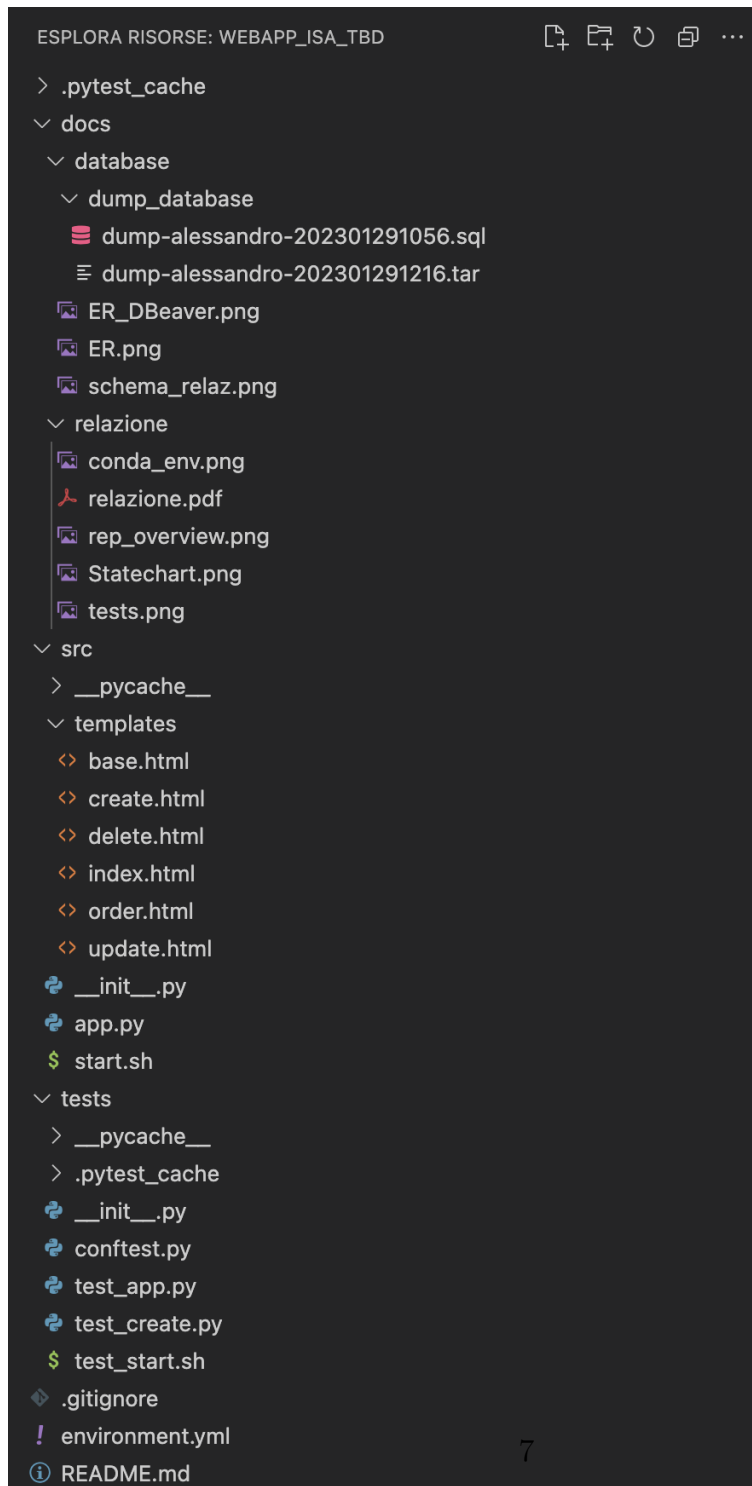
### LEGENDA:

- chiave primaria
- chiave esterna

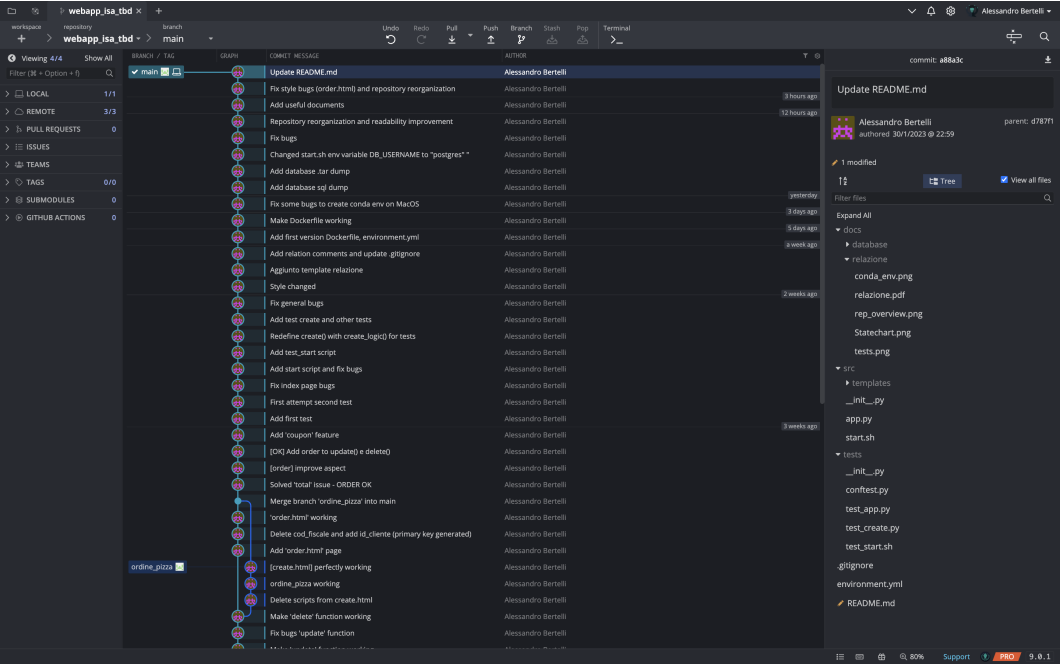
# Requisiti di sistema

Controllo di versione: *Git/GitKraken*

Organizzazione repository



# GUI GitKraken





## Gestione delle dipendenze: *conda*

```
! environment.yml ×
! environment.yml
1  name: proj_env
2  channels:
3  - defaults
4  dependencies:
5  - bzip2=1.0.8
6  - ca-certificates=2022.10.11
7  - certifi=2022.9.24
8  - libffi=3.4.2
9  - libuuid=1.41.5
10 - ncurses=6.3
11 - openssl=1.1.1s
12 - pip=22.2.2
13 - python=3.10.8
14 - readline=8.2
15 - setuptools=65.5.0
16 - sqlite=3.40.0
17 - tk=8.6.12
18 - tzdata=2022g
19 - wheel=0.37.1
20 - xz=5.2.8
21 - zlib=1.2.13
22 - pip:
23   - attrs==22.2.0
24   - charset-normalizer==3.0.1
25   - click==8.1.3
26   - exceptiongroup==1.1.0
27   - flask==2.2.2
28   - flask-testing==0.8.1
29   - idna==3.4
30   - iniconfig==2.0.0
31   - itsdangerous==2.1.2
32   - jinja2==3.1.2
33   - markupsafe==2.1.1
34   - packaging==23.0
35   - pluggy==1.0.0
36   - psycpg2-binary==2.9.5
37   - pytest==7.2.0
38   - pytest-flask==1.2.0
39   - requests==2.28.2
40   - tomli==2.0.1
41   - urllib3==1.26.14
42   - werkzeug==2.2.2
43 prefix: /home/alessandro/anaconda3/envs/proj_env
```

## Suite di test automatizzati: *pytest-flask*

Per sviluppare una suite di test automatizzati ho utilizzato la libreria *pytest-flask*. Come si può notare dalla struttura della repository, tutti i file di test sono, come indicato dalle best practice, organizzati nella cartella *tests* in questo modo:

- **conftest.py**: configurazione iniziale che permette, tramite database URI, la connessione al database per l'esecuzione dei test,
- **test\_app.py**: test che verifica la corretta connessione al database,
- **test\_create.py**: test che verifica il corretto funzionamento della funzione `create()`. Ho impostato dei dati di test e il test client per testare la richiesta HTTP POST al fine di controllare che le prenotazioni vengano inserite correttamente all'interno del database,
- **test\_start.sh**: script bash per avviare automaticamente i test inserendo le variabili globali d'ambiente per connettersi al database.

```
(proj_env) alessandro@airidiAlessandro ~ % cd Documents/webapp_isa_tbd/tests
(proj_env) alessandro@airidiAlessandro tests % ./test_start.sh
===== test session starts =====
platform darwin -- Python 3.10.8, pytest-7.2.0, pluggy-1.0.0
rootdir: /Users/alessandro/Documents/webapp_isa_tbd/tests
plugins: flask-1.2.0
collected 1 item

test_create.py . [100%]
===== 1 passed in 0.85s =====
===== test session starts =====
platform darwin -- Python 3.10.8, pytest-7.2.0, pluggy-1.0.0
rootdir: /Users/alessandro/Documents/webapp_isa_tbd/tests
plugins: flask-1.2.0
collected 1 item

test_app.py . [100%]
===== 1 passed in 0.01s =====
```

## Statechart

