Utilizzo di un database in un’applicazione reale

Luca Gagliardelli

Basi di Dati - Mantova

Introduzione

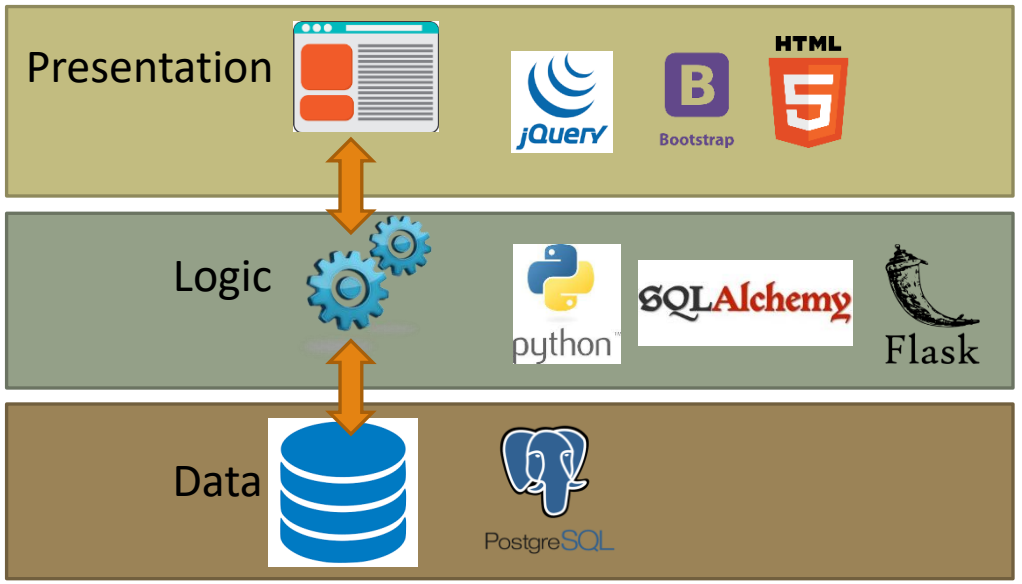
La maggior parte delle applicazioni che usiamo quotidianamente fanno uso di un database per la memorizzazione dei dati.

In questa lezione cercheremo di mettere in pratica gli argomenti trattati durante il corso al fine di realizzare un database per un'applicazione che può essere utilizzata nel mondo reale.

Realizzeremo una tipica web application che utilizza un database per la memorizzazione dei dati.

Struttura dell'applicazione

L'applicazione che realizzeremo avrà una classica struttura a tre livelli ed utilizzerà diversi framework



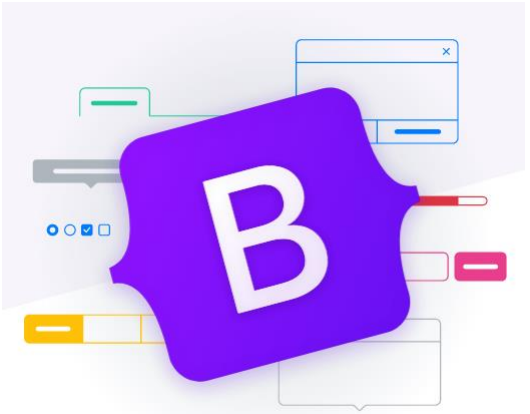
Tecnologie utilizzate

Boostrap

È un framework per la creazione di interfacce grafiche per HTML. È scritto in CSS e JavaScript.

Contiene diversi componenti già pronti che possono essere utilizzati per creare interfacce avanzate in modo semplice.

Sito web di riferimento: https://getbootstrap.com

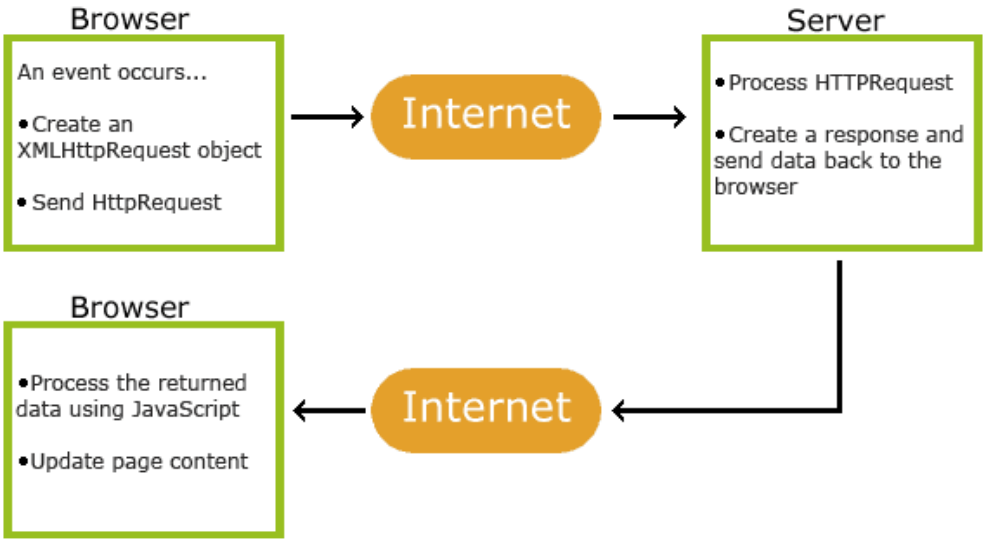


jQuery

È una libreria di JavaScript che contiene diverse utility che possono essere utilizzate per manipolare l'HTML in modo semplice ed efficace.

Contiene anche funzionalità che consentono di gestire in modo semplice chiamate AJAX (**A**synchronous **J**avaScript **A**nd **X**ML), una tecnologia che consente di effettuare richieste a servizi web per ottenere/inviare dati.

Sito web: http://jquery.com



Flask

Flask è un web server scritto in Python, detto micro-framework perché contiene un core minimale per la creazione di servizi web, ma può essere esteso con varie estensioni.

Può essere utilizzato per creare delle API (Application Programming Interface, indica un insieme di procedure per espletare un compito) che possono essere richiamate utilizzando JavaScript (es. con jQuery) per recuperare i dati da un database.

Sito di riferimento: https://flask.palletsprojects.com

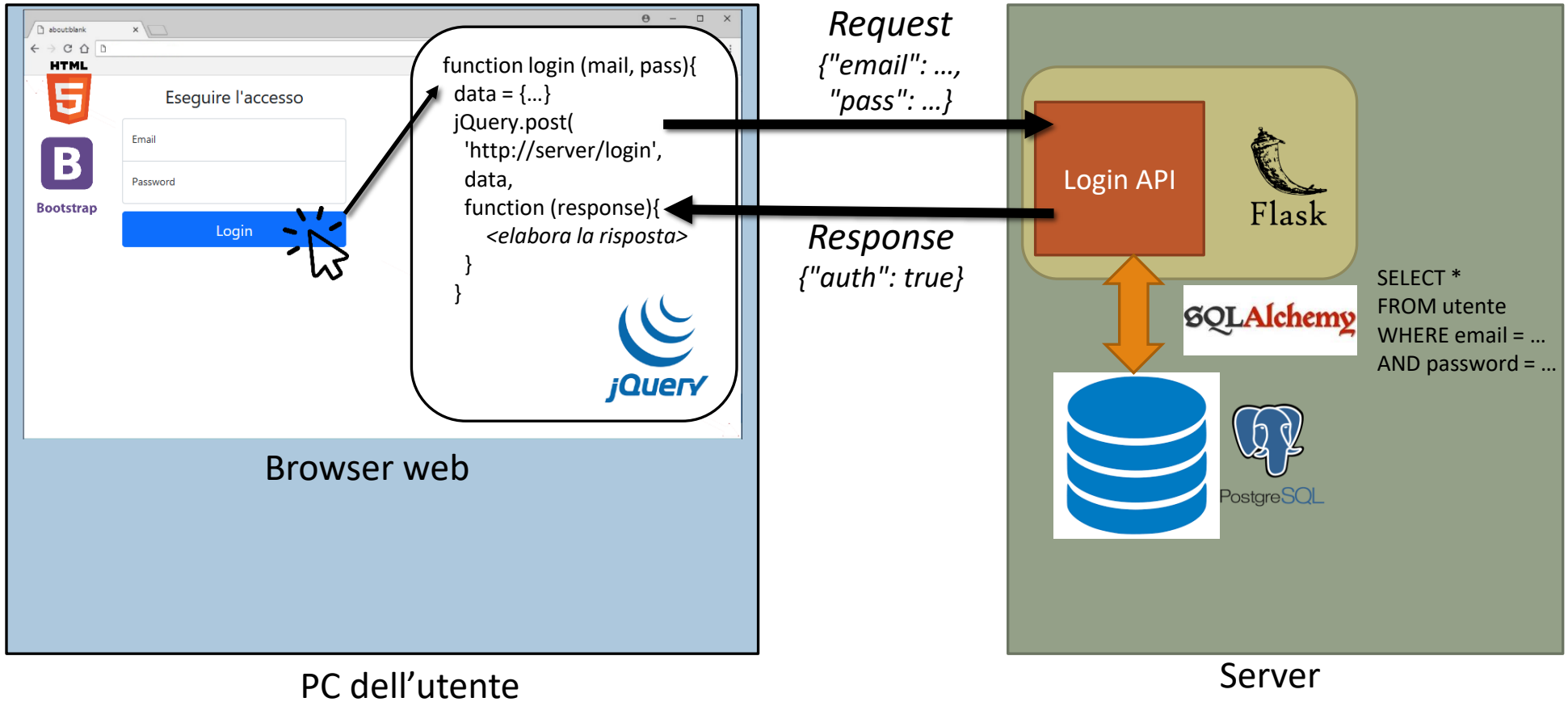
SQLAlchemy

È una libreria di Python che consente di interfacciarsi con vari DBMS per eseguire interrogazioni SQL.

Consente di interrogare un database in modo diretto (RAW) oppure tramite tecnologia Object Relational Mapping (ORM).

Sito web di riferimento: https://www.sqlalchemy.org

Esempio di funzionamento



SQLAlchemy: approfondimento

Installare le librerie SQLAlchemy e Jupyter (è necessario avere Python installato, ad esempio si può installare *Anaconda*)

pip install sqlalchemy
pip install jupyter

Scaricare il notebook
https://github.com/Gaglia88/basi_dati/blob/master/Notebooks/SQLAlchemy.ipynb

Aprire il terminale di Python, spostarsi nella directory in cui si è scaricato il notebook ed eseguire il comando

jupyter notebook

Una volta avviato Jupyter si può aprire il notebook ed eseguirlo.

Servizio ticket

PROGETTAZIONE DEL DATABASE

Richiesta

Si vuole creare un sistema informativo per l'apertura di ticket per un sistema informatico. Al sistema hanno accesso degli utenti di cui si riporta un'email identificativa, la password, il nome e il cognome. Gli utenti si suddividono in tre tipologie:

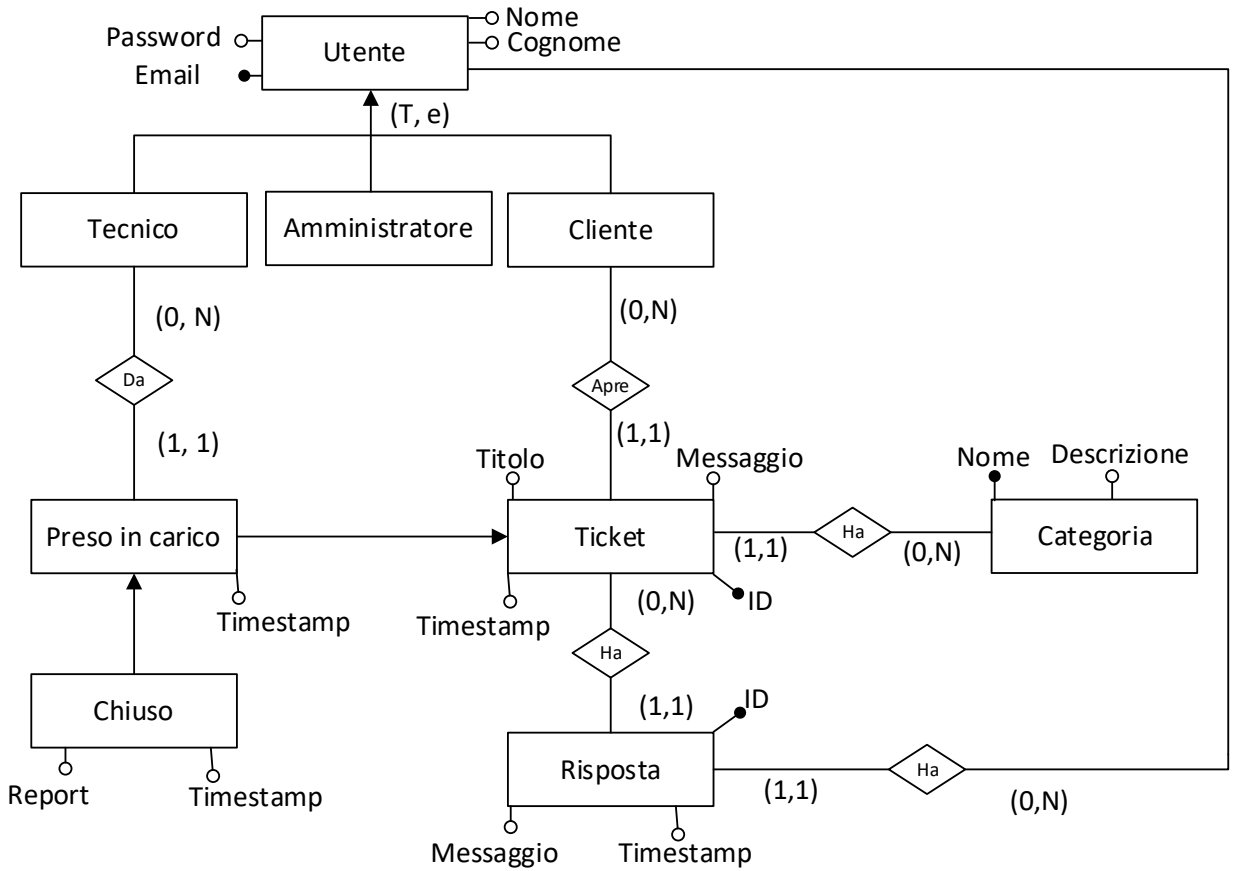
- Amministratori: possono gestire gli utenti iscritti al sistema e le categorie a cui i ticket appartengono;
- Clienti: possono aprire nuovi ticket e inviare note aggiuntive per un ticket segnalato;
- Tecnici: possono leggere i ticket aperti dagli utenti, inviare risposte e cambiare lo stato del ticket.

Un cliente può aprire un ticket, ogni ticket ha un codice identificativo, una data e un'ora di inserimento, un titolo, un messaggio ed appartiene a una categoria. La categoria ha un nome univoco e una descrizione.

I tecnici vedono i ticket aperti dai clienti, li leggono e possono prenderli in carico, quando un tecnico prende in carico un ticket viene memorizzata la data e l'ora di presa in carico. Un tecnico può rispondere ad un ticket inviando un messaggio in cui può chiedere chiarimenti ad un cliente, anche il cliente a sua volta può rispondere. Per ogni messaggio inviato ad un ticket si memorizzano data e ora di inserimento.

Quando il tecnico ha completato un ticket, lo segna come completato. Per un ticket completato si memorizzano data e ora di completamento e un report che descrive cosa è stato fatto per risolverlo. Il ticket può essere chiuso solo dal tecnico che lo prende in carico.

Modello ER


Password
Email
Utente
Nome
Cognome
(T, e)
Tecnico
Amministratore
Cliente
(0, N)
(0,N)
Da
Apre
(1, 1)
(1,1)
Titolo
Messaggio
Nome
Descrizione
Preso in carico
Ticket
Ha
Categoria
(1,1)
(0,N)
Timestamp
Timestamp
(0,N)
ID
Ha
Chiuso
(1,1)
ID
Risposta
Ha
Report
Timestamp
(1,1)
(0,N)
Messaggio
Timestamp

Traduzione in relazionale

Per semplificare lo schema eseguiamo un collasso verso l'alto di tutte le gerarchie/specializzazioni

Utente (Email, Password, Nome, Cognome, Ruolo)
  Dominio Ruolo: {amministratore, cliente, tecnico}

Categoria (Nome, Descrizione)

Ticket (id, titolo, messaggio, timestamp, timestamp_carico, report, timestamp_chiusura, email_cliente, email_tecnico, stato, nome)
  Dominio stato: {aperto, in_carico, chiuso}
  FK: nome **REFERENCES** categoria **NOT NULL**
  FK: email_cliente **REFERENCES** utente **NOT NULL**
  FK: email_tecnico **REFERENCES** tecnico

Risposta(Id, ticket_id, timestamp, messaggio, email)
  FK: ticket_id **REFERENCES** ticket **NOT NULL**
  FK: email **REFERENCES** utente **NOT NULL**

Considerazioni

Nei database che vengono utilizzati in applicazioni reali, solitamente, si:

- Aggiungere un **identificatore incrementale di tipo intero** ad ogni tabella, cercando di evitare chiavi primarie composte o di tipo stringa. Queste creerebbero problemi nella dimensione del database dovendo riportare molti dati nelle tabelle che vi fanno riferimento. Le **chiavi originali verranno trasformate in chiavi alternative** (AK);
- Mantenere tutto lo **schema** in **minuscolo** (nomi tabelle, attributi);
- I **nomi** delle **tabelle** andrebbero tenuti al **singolare**, in questo modo si riesce meglio ad identificare cosa contiene un singolo record;
- Utilizzare una **notazione snake_case** per definire i nomi delle tabelle e degli attributi.
- Dare **nomi significativi alle associazioni tradotte**: quando si traduce un'associazione molti-a-molti è bene rinominare la relazione (tabella) risultante con i nomi delle due entità che associa separate da un underscore (es. *entità1_entità2*).

Schema relazionale modificato

utente (uid, email, password, nome, cognome, ruolo)
  AK: email
  Dominio ruolo: {amministratore, cliente, tecnico}

categoria (cid, nome, descrizione)

ticket (tid, titolo, messaggio, timestamp, timestamp_carico, report_chiusura, timestamp_chiusura, uid_cliente, uid_tecnico, stato, cid)
  Dominio stato: {aperto, in_carico, chiuso}
  FK: cid REFERENCES categoria NOT NULL
  FK: uid_cliente REFERENCES utente NOT NULL
  FK: uid_tecnico REFERENCES tecnico

risposta(rid, tid, timestamp, messaggio, uid)
  FK: tid REFERENCES ticket NOT NULL
  FK: uid REFERENCES utente NOT NULL

Esercizio

Tradurre in SQL lo schema relazionale presentato nella slide precedente.

Suggerimenti

- Utilizzare un vincolo CHECK per verificare che il ruolo possa assumere solo i valori elencati;
- Gli identificatori incrementali in PostgreSQL sono definiti dal tipo SERIAL;

Applicazione

Collegarsi a https://github.com/Gaglia88/basi_dati/tree/master/Applicazione e seguire le istruzioni riportate.

Alla stessa pagina è possibile trovare la soluzione dell'esercizio precedente.