efregn

Tesina del progetto del corso «Tecnologie Web» A.A. 2023-2024

Cocciardi Daniele Silvestro – MATR. N. 142029

Sito web di un ristorante di specialità pugliesi.

Sommario

1 Presentazione generale del progetto

1.1 Traccia

«Efregn» è il sito web del fittizio ristorante di specialità pugliesi con lo stesso nome. Il sito consente agli utenti di esplorare i vari piatti presenti nel menù del ristorante e, previa registrazione, di creare una vera e propria ordinazione da spedire direttamente in cucina, accelerando così i tempi per la consumazione. I manager del ristorante possono, tramite un'area a loro riservata, modificare il menù aggiungendo nuovi piatti.

1.2 Funzionalità

Vengono di seguito elencate le principali funzionalità del sito web.

- **Esplorazione dei piatti:** Gli utenti (sia utenti registrati che guest) possono visualizzare tutti i piatti presenti nel menù del ristorante. I piatti sono ordinati di default per prezzo crescente, ma è possibile cambiare l'ordinamento (scegliendo tra: prezzo crescente, prezzo decrescente e nome). È anche possibile filtrare i piatti per categoria ("classico" identifica un piatto senza una categoria specifica, "vegetariano" identifica un piatto senza ingredienti di origine animale, "vegano" identifica un piatto senza ingredienti di origine né derivazione animale, "piccante" identifica un piatto in cui sono presenti spezie piccanti). È possibile effettuare una ricerca globale direttamente dalla navbar per trovare piatti o ingredienti richiesti.
- **Sistema di ordinazione:** Previa registrazione al sito, gli utenti (non di tipo guest), possono creare una nuova ordinazione e aggiungere i piatti desiderati. Una volta completata, è possibile modificare (ossia rimuovere alcuni piatti) o confermare l'ordinazione per inviarla alla cucina. Tale sistema funziona in maniera analoga al "carrello" dei comuni siti di e-commerce. Per ogni ordinazione l'utente accumula anche dei punti fedeltà, calcolati automaticamente dal sito in fase di conferma (50 punti per ogni euro speso). Tali punti fedeltà consentono all'utente di ottenere uno *sconto di 5 euro al raggiungimento dei 5000 punti*.
- **Profili "Manager" e tab per l'amministrazione:** «Efregn» consente agli utenti facenti parte dello staff di registrarsi al sito con un ac-

count appartenente al gruppo “Manager”, inserendo nel form di registrazione una mail il cui dominio termina in *@efregnstaff.com*. Il sito, dopo l’inserimento della suddetta mail aziendale, assegnerà automaticamente all’utente il gruppo “Manager”. Gli account Manager possono, tramite una scheda riservata dal nome “Amministrazione”, aggiungere nuovi piatti al menù del ristorante, inserendo per ogni piatto: il nome, gli ingredienti, il prezzo, la o le categoria/e (“classico”, “vegetariano”, “vegano”, “piccante”) e un’immagine che lo rappresenta. Gli account di questa tipologia possono, come quelli degli utenti normali, effettuare ordinazioni da inviare direttamente in cucina e beneficiano di uno sconto dipendenti pari al 15% del totale, applicato automaticamente dal sito in fase di conferma. Gli account manager sono *esclusi dalla raccolta punti*.

- **Piatti consigliati:** Gli utenti che effettuano l’accesso possono visualizzare sulla homepage – e aggiungere all’ordinazione direttamente da lì – dei piatti consigliati, in base alle precedenti ordinazioni e alle categorie che hanno consumato più di frequente.

1.3 Use Case UML Diagram

Di seguito il diagramma:

1.4 Git

Per l'hosting remoto ho scelto di utilizzare la piattaforma *GitHub*. Il repository si trova all'indirizzo: https://github.com/dcocciardi/efregn.

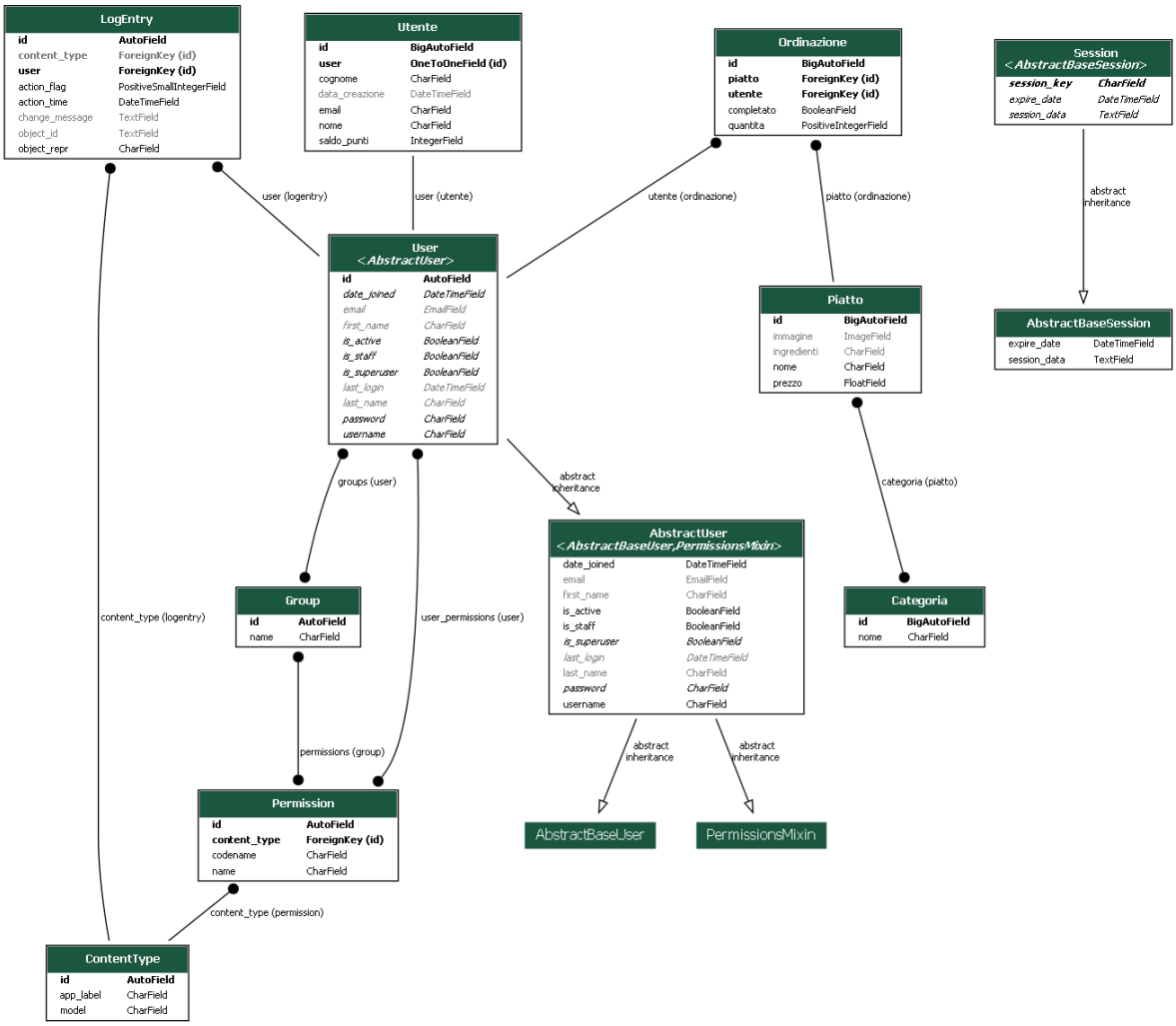
2 Database

2.1 Database adoperato

Per Efregn ho scelto di adoperare il database SQLite fornito di default da Django, ottimale per lo sviluppo e il testing dell'applicazione. Tuttavia, per un ambiente di produzione, è consigliabile un'opzione che garantisca maggiori prestazioni ed affidabilità, come ad esempio un DBMS del tipo di MySQL. In quel caso, è necessario modificare le impostazioni in *settings.py*.

2.2 Diagramma delle classi

Di seguito il diagramma delle classi, generato con *graph_models:*



2.3 Dettagli aggiuntivi

2.3.1 La classe Utente

La prima macro-entità su cui ho ritenuto essenziale che si basasse Efregn è l'utente, ossia la persona registrata al programma fedeltà. Di tale persona, il sito salva alcuni campi fondamentali, tra cui il nome. Questo campo viene poi usato in diverse pagine per creare un'esperienza più "personalizzata" (fig. a). Questa classe rappresenta dunque l'entità alla quale è principalmente rivolta la piattaforma, ossia i clienti fidelizzati del locale.

Figura a: Dettaglio della schermata di ordinazione in cui viene usato il nome utente.

Per consentire a questa classe di gestire al meglio anche gli account creati dallo staff, ho pensato di sfruttare la classe Group inclusa in Django. Tale classe consente di associare ad ogni utente un gruppo, ognuno con i propri permessi e caratteristiche. Questa scelta mi ha aiutato non poco nello sviluppo, soprattutto per rendere alcune pagine visualizzabili soltanto dai manager, oppure per gestire il funzionamento del programma fedeltà.

2.3.2 Pagina di Amministrazione

Volevo sincerarmi che solamente gli account appartenenti al gruppo Manager fossero in grado di modificare il menù. Per questo, mi sono avvalso di un *decoratore* che ho chiamato "utenti_autorizzati". Tale decoratore restringe dunque la visibilità a determinati "ruoli" (grup-

pi). Mi è stato sufficiente usare il decoratore con la specifica (*ruoli_autorizzati=['Manager'])* per ottenere l'effetto desiderato.

Figura b: Pagina visualizzata da un utente non autorizzato a modificare il menù.

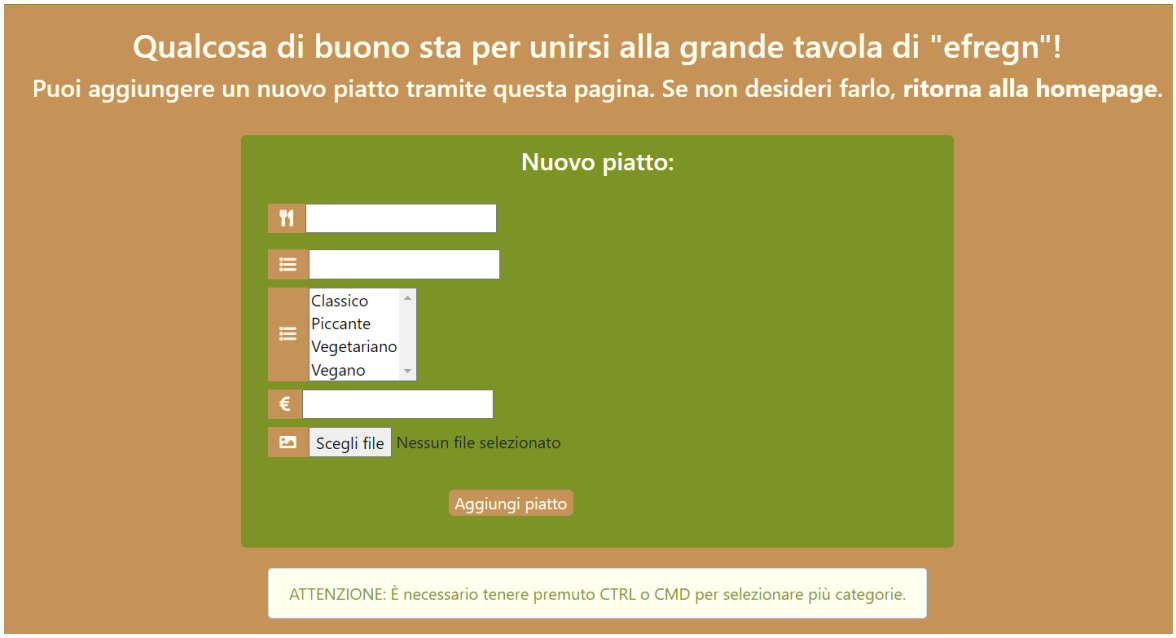


Figura c: Pagina visualizzata da un Manager.

Tutti i decoratori che ho utilizzato nel progetto sono presenti in *accounts/decorators.py*

2.3.3 L'ordinazione

Ordinazione è la classe che consente alla piattaforma di svolgere il suo compito al meglio. Gli utenti, una volta effettuato l'accesso dalla schermata di login, possono aggiungere i piatti che vogliono mangiare alla loro ordinazione. Il sito calcola automaticamente il totale dell'ordinazione, i punti accumulati e gli sconti erogati (15% del totale se l'account appartiene al gruppo Manager; 5 euro se l'account ha un saldo punti maggiore o uguale a 5000 e il totale è maggiore di 5 euro). L'ordinazione prende anche in considerazione le categorie dei piatti che sono stati aggiunti per suggerirne di simili (vedi paragrafo 4.2 Recommendation System).

Figura d: Confronto tra ordinazioni in cui vengono applicate scontistiche diverse.

3 Logica backend

Le cartelle del progetto sono organizzate come segue. Quelle principali vengono dettagliate di fianco:

```
efregn
├──accounts
│   ├──migrations
│   └──templates
│       └──accounts
├──efregn
├──images
│   └──piatti
└──static
    ├──css
    ├──images
    └──javascript
```

- **Accounts** contiene il codice principale;
- **Migrations** contiene le migrazioni generate durante lo sviluppo;
- **Templates** contiene i template HTML;
- **Efregn** è la cartella creata di default da *startproject*;
- **Images/piatti** contiene le immagini dei piatti, anche quelle caricate dagli utenti del gruppo Manager;
- **Static/images** contiene le immagini di sfondo usate in alcune pagine e il logo.

Alcune cartelle, ad esempio static/javascript, sono vuote e non contengono alcun elemento. Questo perché, in fase di sviluppo, avevo cercato di separare gli elementi quanto più possibile, ma sono incappato in alcuni problemi di esecuzione e ho preferito "trasferire" il codice direttamente nelle pagine HTML.

3.1 I packages

L'unica dipendenza presente nel Pipfile risulta essere Django:

```
[packages]
django = "*"
```

4 Logica frontend

Per il frontend del progetto ho usato Bootstrap nella versione 4.4.1, jQuery nella versione 3.4.1 e FontAwesome nella versione 5.6.1. Inoltre, per gestire il corretto posizionamento dei menù dropdown inizialmente previsti nella navbar e successivamente relegati alla sola versione mobile, ho utilizzato anche la libreria Popper.js nella versione 1.16.0.

4.1 Navbar e ricerca globale

Per rendere migliore l'esperienza utente, ho inserito all'interno della navbar una casella di ricerca globale. Tramite questa casella l'utente può digitare gli ingredienti, le categorie e i nomi dei piatti presenti del menù. Se l'utente ha effettuato l'accesso, sarà in grado di aggiungere all'ordinazione i piatti che corrispondono ai criteri di ricerca immessi direttamente dalla pagina dei risultati.

4.1.2 Template coinvolti

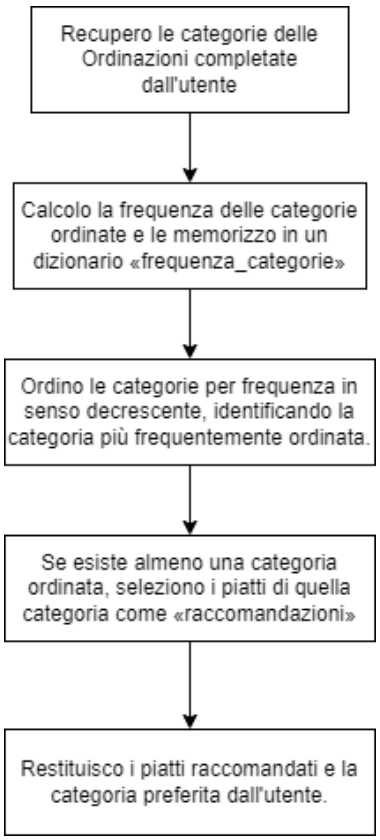
Il campo di ricerca è posizionato nella navbar, il cui template si trova in *templates/navbar.html*. I risultati di ricerca vengono visualizzati nella pagina cerca_piatto, il cui template si trova in *templates/cerca_piatto.html*.

4.1.3 View "cerca_piatto"

La view che si occupa di gestire la ricerca si chiama cerca_piatto, ed è presente nel file *accounts/views.py*. La view accetta solo richieste GET. La query string della richiesta viene utilizzata per recuperare il valore del parametro *q*, che rappresenta la stringa di ricerca. Suc-

cessivamente, la view esegue una ricerca nel modello *Piatto* filtrando gli oggetti che hanno il campo *nome* o *ingredienti* che contengono la stringa di ricerca (case-insensitive). Infine, i risultati della query vengono passati al template *cerca_piatto.html* insieme alla stringa di ricerca. Il template utilizza quindi questi dati per visualizzare i risultati.

4.2 Recommendation System



Per personalizzare ulteriormente l'esperienza utente, ho implementato un recommendation system che potesse consigliare all'utente alcuni piatti da aggiungere all'ordinazione.

Il sistema viene gestito dalla view *genera_raccomandazioni* (disponibile nel file *accounts/views.py*). Tale view raccomanda dei piatti in base alle categorie precedentemente ordinate da un utente, come da schema a lato.

Il template HTML della home gestisce i risultati passati dalla view e li visualizza in fondo alla pagina, con uno stile grafico leggermente diverso rispetto ai normali piatti del menù. La categoria più amata dall'utente viene anche esplicitata (fig. e)

Figura e: Dettaglio della homepage con le raccomandazioni. La categoria è esplicitata in alto.

5 Sfide riscontrate

Durante lo sviluppo di questo progetto, rivelatosi appassionante e molto interessante, non sono state poche le sfide che ho affrontato. Era la prima volta che mettevo le mani su un progetto Python e non avevo minimamente idea di come funzionasse il linguaggio. A mano a mano che continuavo il progetto, e che seguivo le lezioni, tutto però è sembrato farsi più chiaro. Alla fine, solamente due sono state le sfide maggiori.

5.1 Sfida: gestione dell'ordinazione

Ammetto che creare il meccanismo dell'ordinazione ha creato non pochi problemi. Ho lasciato volontariamente quella parte per ultima, così da pensarci quanto più tardi possibile, ma alla fine è stato necessario implementarla. I problemi riscontrati maggiormente sono stati in merito alla rimozione dei piatti (risolta aggiungendo un button apposito) e al meccanismo di calcolo sconti e accumulo punti. Mi veniva estremamente complicato calcolare tutto in una unica view e passare poi i risultati al template, così, quasi sul finire del progetto, ho deciso di "decentralizzare" il calcolo dei dati di ogni singola ordinazione in diverse view separate. Al momento in cui scrivo questa tesina, l'ordinazione è una delle parti più complesse della piattaforma ed è gestita dalle view: *calcola_dati_ordinazione*, *visualizza_ordinazione* e *invia_ordinazione*. Per il corretto funzionamento del sistema, sono essenziali anche le view: *aggiungi_al_carrello* e *rimuovi_piatto*. Tutte queste view sono disponibili in *accounts/views.py*. La scelta di usare una view che calcolasse totale, sconti, scontistiche riservate ai Manager e punti accumulati è stata illuminante e mi ha permesso di rendere più snello il codice.

5.2 Sfida: pagina "Amministrazione"

Inizialmente avevo pensato di includere la pagina di amministrazione con una voce nella navbar che comparisse solo se il gruppo dell'utente che aveva effettuato l'accesso fosse stato "Manager". Per qualche motivo, però, non sono riuscito a realizzare questo e la dicitura "Amministrazione" compariva nella navbar di qualsiasi utente. Per risolvere il problema, quindi, ho pensato ad un approccio leggermene diverso: mostrare sempre la pagina nella navbar ma, tramite l'uso di un decoratore, reindirizzare l'utente non autorizzato ad una pagina dal nome *non_autorizzato.html*. Le informazioni in merito al decoratore utilizzato e alla pagina in sé sono al paragrafo 2.3.2 Pagina di Amministrazione.

6 Test

Ho effettuato test su:

- Creazione di un nuovo utente semplice;
- Creazione di un nuovo utente Manager, per controllare il corretto inserimento dell'utente nel gruppo Manager a partire dalla sola mail aziendale;
- Creazione di una nuova ordinazione da parte di un utente semplice, aggiungendo al carrello un piatto di tipo "caffè", per controllare la corretta aggiunta del piatto all'ordinazione;
- Invio dell'ordinazione in cucina.

L'esito di tali test è stato positivo:

```
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....
----------------------------------------------------------------------
Ran 4 tests in 4.680s

OK
Destroying test database for alias 'default'...
```

I test sono visualizzabili al file *accounts/tests.py*.

7 Screenshots

Gli screenshot, compresi quelli utilizzati in questa tesina, sono disponibili nel repository (https://github.com/dcocciardi/efregn). Più precisamente, in docs/screenshot.

8 Link utili

https://fontawesome.com/docs/web/use-with/python-django

https://popper.js.org/docs/v1/

https://jquery.com/

https://getbootstrap.com/docs/4.4/getting-started/introduction/