

Relazione progetto di Linguaggi Dinamici

Realizzazione di un'applicazione Web sviluppata con framework Django

Sito per la spedizione di pasti, come intermediario tra clienti
e ristoranti: Trust Eat



Sviluppata da:

Marco Piccinni

Daniele Tortoli

Corso di Informatica Triennale, Unimore A.A. 2018/2019

Indice:

- [Prerequisiti e informazioni generali](#)
- [Funzionalità](#)
- [Permessi](#)
- [Schema del Database e implementazione](#)
- [Organizzazione del codice](#)
- [Utilizzo dell'applicazione](#)
- [JavaScript](#)
- [Test](#)

Prerequisiti e informazioni generali

Per la realizzazione dell'applicazione web sono stati utilizzati i seguenti software:

- **Django** v2.1.5
- **PostgreSQL** v10
- **pgAdmin4** v3.5
- **Python** 3.7

Inoltre è stato necessario l'utilizzo dei seguenti componenti aggiuntivi per python (installabili tramite il comando *pip*):

- **django-crispy-forms**==1.7.2
- **psycopg2-binary**==2.7.6.1
- **social-auth-app-django**==3.1.0
- **Pillow**==5.4.1
- **googlemaps**==3.0.2

La maggior parte del codice è scritta in Python 3 (v3.7) mentre per le pagine web è stato usato HTML, CSS e Javascript.

Per lo sviluppo delle componenti grafiche delle pagine ci si è appoggiati alle librerie **Bootstrap** v4.1 (<https://getbootstrap.com/>)

Funzionalità

L'applicazione, chiamata Trust Eat, è una piattaforma per ordinare cibo da asporto da locali registrati per essere consegnato alla propria abitazione.

Il sito è pensato per poter essere utilizzato sia da utenti registrati, sia da utenti anonimi, questi ultimi con funzionalità ridotte.

La registrazione è completabile tramite il sito, oppure tramite social (Google e Facebook).

Gli utenti non registrati possono visualizzare i locali registrati (con la possibilità di utilizzare la pagina di ricerca), leggerne i dettagli e l'elenco di prodotti e menù offerti. Per ogni locale possono inoltre visualizzare il voto lasciato da altri utenti e le relative recensioni, ma non possono aggiungerne.

Gli utenti registrati possono, in aggiunta a quanto è possibile fare come utente anonimo, effettuare e gestire gli ordini (ovvero annullarli, finché il locale non accetta o rifiuta l'incarico), e osservarne lo storico.

Per confermare l'ordine è necessario impostare l'orario di preferenza a cui si desidera ricevere l'ordine e specificare la modalità di pagamento: alla consegna o con carta di credito valida (aggiunta in fase di registrazione o successivamente nella pagina personale). L'ordine è possibile richiederlo entro la giornata corrente. Infine è possibile lasciare una valutazione al locale includendo, se si desidera, una descrizione.

Un utente può registrarsi anche come commerciante, in tal caso può aggiungere e gestire locali creati da lui o da altri, nel caso questi gli abbiano concesso l'autorizzazione.

La gestione riguarda le informazioni del locale (*nome, descrizione, tag, posizione, recapiti, foto, prezzo di spedizione, altri proprietari*), di prodotti e menù (*nome, foto, descrizione, prezzo e, nel caso dei menù, i prodotti associati*).

Inoltre è possibile *accettare o rifiutare* gli ordini in attesa, avendo a disposizione ulteriori dati come la distanza effettiva e il tempo di consegna previsto, visualizzare su una mappa gli ordini *in consegna* e osservare lo storico degli ordini da consegnare, consegnati e rifiutati.

Per ogni recensione ricevuta su un locale è possibile lasciare una risposta.

La ricerca viene effettuata nel seguente modo: bisogna prima di tutto specificare la località (nel caso l'utente sia registrato la località di default è quella del proprio indirizzo inserito nei dati personali), in aggiunta si può andare a restringere il campo con dei filtri in base alla tipologia del locale (*tag*).

I locali mostrati in seguito alla ricerca sono organizzati in ordine decrescente di voto e, in caso di parità, avrà la precedenza il locale con maggior numero di recensioni (in quanto considerato un indice di maggiore affidabilità).

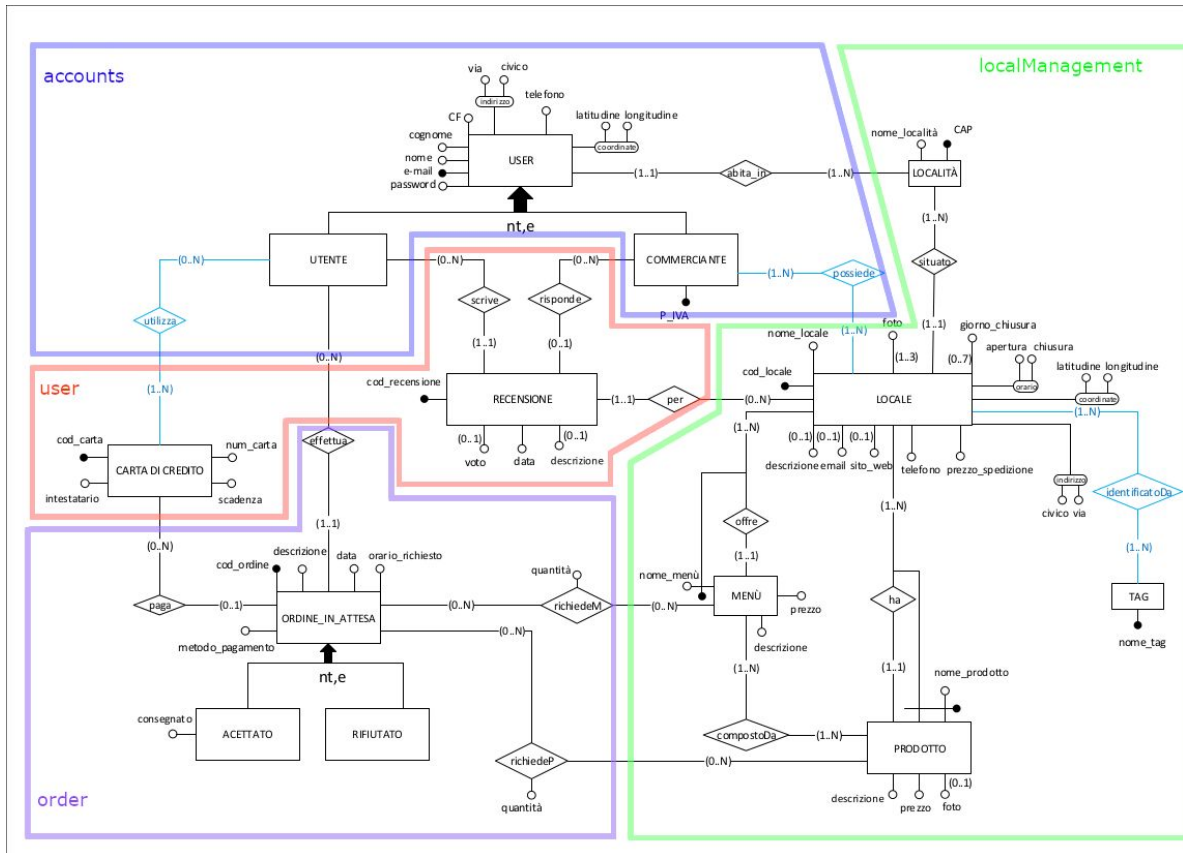
La pagina di amministrazione permette di creare e modificare dati all'interno del database, solo tramite questa pagina è possibile aggiungere località e tag (per evitarne un uso scorretto).

Permessi

Funzionalità	Anonimo	Utente	Commerciante
Ricerca	✓	✓	✓
Registrazione	✓	✗	✗
Login	✗	✓	✓
Profilo personale	✗	Lista ordini	Lista locali
Modifica dati personali	✗	Carte di credito	Partita IVA
Locale	✓	✓	✓
Gestione locale	✗	✗	✓
Ordinazione	✗	✓	✗
Gestione ordini locale	✗	✗	✓
Valutazione	✗	✓	✗
Modifica valutazione	✗	✓	✗
Risposta	✗	✗	✓
Modifica risposta	✗	✗	✓
Visualizzazione recensioni	✓	✓	✓
Aggiunta carta di credito	✗	✓	✗
Visualizzazione riepilogo ordine	✗	✓	Senza carta di credito

Schema del Database e implementazione

Schema ER con suddivisione delle applicazioni



In **azzurro** sono segnate le relazioni di tipo Multi-a-Multi che sono create e gestite automaticamente da Django.

Le entità sono racchiuse, secondo l'[organizzazione del codice](#), nelle relative applicazioni.

The diagram illustrates the following models and their attributes:

- django.contrib.contenttypes.models.ContentType**: app_label (CharField), model (CharField), objects (ContentTypeManager), permission_set (QuerySet).
- django.contrib.auth.models.Permission**: codename (CharField), content_type (ForeignKey), group_set (QuerySet), name (CharField), objects (PermissionManager).
- django.contrib.auth.models.Group**: name (CharField), objects (GroupManager), permissions (ManyToManyField), user_set (QuerySet).
- accounts.models.User**: cap (CharField), civico (CharField), email (EmailField), is_commerciante (BooleanField), is_utente (BooleanField), latitude (FloatField), longitude (FloatField), telefono (CharField), utente_user (QuerySet), via (CharField).
- accounts.models.Commerciante**: p_iva (CharField), possiede_locale (ManyToManyField), recensione_set (QuerySet), user (OneToOneField).
- user.models.CartaDiCredito**: carta_di_credito (CharField), cod_carta (AutoField), intestatario (CharField), numero_carta (CharField), scadenza (DateField).
- user.models.Utente**: carta_di_credito (ManyToManyField, through = localManagement.models.CompostoDa), recensione_set (QuerySet), user (OneToOneField).
- user.models.Recensione**: cod_locale (ForeignKey), cod_recensione (AutoField), date (DateField), descrizione (CharField), email (ForeignKey), p_iva (ForeignKey), voto (SmallIntegerField).
- localManagement.models.Tag**: locale_tag (QuerySet), nome_tag (CharField).
- localManagement.models.Localita**: cap (CharField), locale_set (QuerySet), nome_locale (CharField).
- localManagement.models.Locale**: cap (ForeignKey), cod_locale (AutoField), descrizione (CharField), email (EmailField), latitude (FloatField), longitude (FloatField), nome_locale (CharField), num_civico (CharField), orario_apertura (TimeField), orario_chiusura (TimeField), prezzo_di_spedizione (FloatField), richiestep_set (QuerySet), sito_web (CharField), tag (ManyToManyField), telefono (CharField), via (CharField).
- localManagement.models.Prodotto**: cod_locale (ForeignKey), descrizione_prodotto (CharField), foto_prodotto (ImageField), nome_prodotto (CharField), prezzo (DecimalField), type_nome_prodotto (QuerySet).
- localManagement.models.Chiusura**: cod_locale (ForeignKey), giorno_chiusura (CharField).
- localManagement.models.FotoLocale**: cod_locale (ForeignKey), foto_locale (ImageField).
- localManagement.models.Menu**: cod_locale (ForeignKey), composto_da_prodotto (ManyToManyField), descrizione_menu (CharField), nome_menu (CharField), prezzo (DecimalField), type_nome_menu_1 (QuerySet).
- localManagement.models.CompostoDa**: cod_locale (ForeignKey), nome_menu (ForeignKey), nome_prodotto (ForeignKey).
- order.models.OrdineInAttesa**: accettato (BooleanField), carta_di_credito (ForeignKey), cod_carta (AutoField), cod_ordine (AutoField), consegnato (BooleanField), data (DateTimeField), descrizione (CharField), email (ForeignKey), menus (ManyToManyField), metodo_pagamento (CharField), orario_richiesto (TimeField), prodotti (ManyToManyField), richiestep_set (QuerySet).
- order.models.RichiedeM**: cod_locale (ForeignKey), cod_ordine (ForeignKey), nome_menu (ForeignKey), quantita (SmallIntegerField).
- order.models.RichiedeP**: cod_locale (ForeignKey), cod_ordine (ForeignKey), nome_prodotto (ForeignKey), quantita (SmallIntegerField).

Relationships are defined as follows:

- ContentType** to **Permission**: 1 to 0..1.
- Permission** to **Group**: 0..* to 0..*.
- Group** to **User**: 0..* to 0..*.
- User** to **Commerciante**: 1 to 1.
- User** to **Utente**: 1 to 1.
- User** to **Recensione**: 0..* to 0..*.
- Commerciante** to **Recensione**: 1 to 0..*.
- Commerciante** to **Locale**: 0..* to 0..*.
- Commerciante** to **Prodotto**: 0..* to 0..*.
- Commerciante** to **Menu**: 0..* to 0..*.
- Commerciante** to **Chiusura**: 0..* to 0..*.
- Commerciante** to **FotoLocale**: 0..* to 0..*.
- Utente** to **Recensione**: 1 to 0..*.
- Utente** to **Menu**: 1 to 0..*.
- Utente** to **Chiusura**: 1 to 0..*.
- Utente** to **FotoLocale**: 1 to 0..*.
- Prodotto** to **Menu**: 0..* to 0..*.
- Prodotto** to **CompostoDa**: 0..* to 0..*.
- Menu** to **CompostoDa**: 1 to 0..*.
- Menu** to **OrdineInAttesa**: 1 to 0..*.
- CompostoDa** to **OrdineInAttesa**: 1 to 0..*.
- OrdineInAttesa** to **RichiedeM**: 1 to 0..*.
- OrdineInAttesa** to **RichiedeP**: 1 to 0..*.
- RichiedeM** to **RichiedeP**: 0..* to 0..*.

Organizzazione del codice

Il progetto si suddivide in 5 applicazioni:

1. **accounts**: Per la gestione dei differenti tipi di utenti: user, utente e commerciante e le meccaniche di registrazione e modifica dei dati personali.
2. **localManagement**: Si occupa dei locali e della loro gestione. Visto il ruolo centrale che il modello di Locale ricopre (come è possibile notare dall'UML) questa è la parte più corposa del progetto.
3. **order**: Per la gestione degli ordini.
4. **search**: Per la visualizzazione della homepage del sito che permette la ricerca ordinata dei locali registrati, secondo i parametri selezionati dall'utente.
5. **user**: Per la gestione di strumenti relativi all'utente, come le carte di credito e le recensioni.

Sono presenti anche ulteriori cartelle:

- **assets**: Contiene i file statici del progetto suddivisi in ulteriori cartelle (CSS, JavaScript, loghi e icone).
- **media_db**: Contiene le immagini, organizzate per locale e prodotto, caricate durante le operazioni di creazione o modifica.
- **templates**: Contiene i template di base, di amministrazione e registrazione al sito.
- **picture**: Contiene alcuni esempi di immagini e foto da poter caricare. Rappresenta una directory qualsiasi all'interno del proprio sistema operativo.

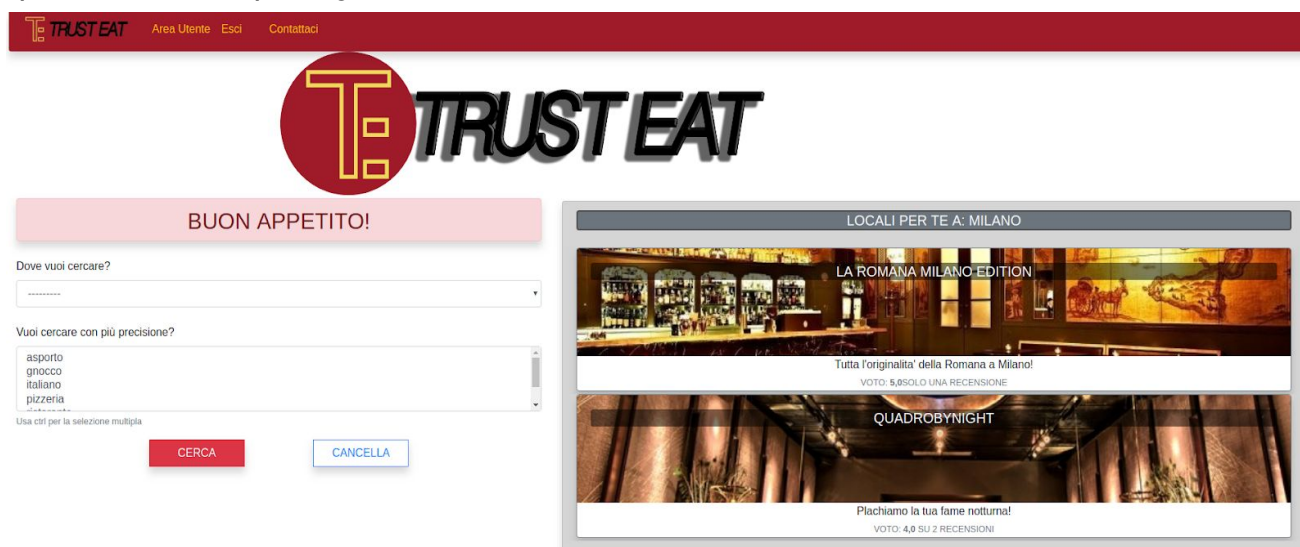
Infine è presente il file **maps.py**, internamente alla cartella base del progetto (*TrustEat*). Questo file contiene alcune funzioni che permettono una più rapida e semplice interfaccia alle *API di Google Maps*.

In particolare queste funzioni permettono di ottenere le coordinate geografiche (data una località, indirizzo e numero civico) e distanza e tempo di percorrenza (date due posizioni).

Google fornisce le proprie API con molte restrizioni per gli sviluppatori che desiderano provare le funzionalità da loro offerte. Per questo motivo entrambe le funzioni sono parzialmente funzionanti: le richieste esterne sono disattivate, fornendo invece delle risposte standard, per evitare tempi eccessivamente lunghi durante i caricamenti delle pagine.

Utilizzo dell'applicazione

Questa è l'home page del sito, dal quale si possono filtrare i locali, già ordinati in base alla tua località e al voto delle recensioni, secondo altri criteri, quali un posto specifico o uno o più tag.



Nella navbar in alto sono presenti alcuni link utili, come *Area Utente*, *Esci* e *Contattaci*.

In questo particolare caso la schermata è stata catturata utilizzando un utente loggato.

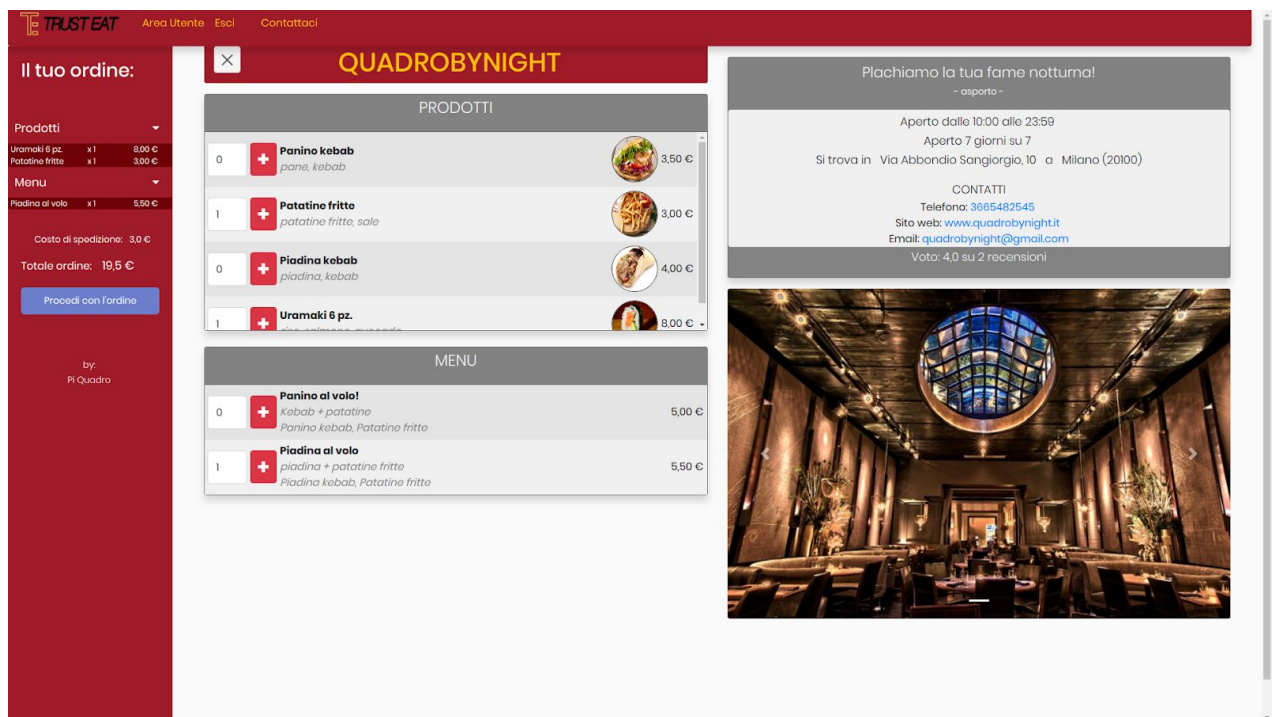
Un utente non loggato invece può utilizzare un link alla *Registrazione*, che si presenta nel seguente modo:

Si può notare la possibilità di registrarsi anche tramite social: *Google* o *Facebook*. La scelta di utilizzare uno di queste opzioni porta ad un'altra pagina dove verrà richiesto di inserire i dati mancanti, necessari per il corretto funzionamento del sito, che non è possibile prendere tramite i social considerati.

Infine è presente anche il link per il form di registrazione del *commerciante*, in quanto vengono richieste informazioni differenti.

N.B.: L'accesso con Facebook in realtà si blocca dopo il reindirizzamento al social. Questo a causa delle impostazioni di Facebook, che non concedono il reindirizzamento a siti non sicuri (quindi sprovvisti di protocollo HTTPS).

Una volta loggato, ecco un esempio di come l'utente vede un locale visitato:

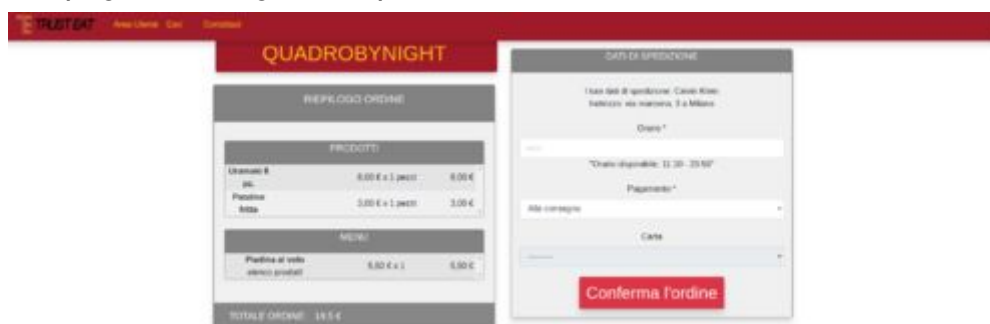


In questa schermata dai colori sgargianti possiamo notare come essa sia fondamentalmente ripartita in 3 colonne che analizzeremo da destra verso sinistra. Nella prima colonna troviamo tutte le informazioni sul locale, quali descrizione, tag, orari, giorni di chiusura, contatti e un link alle recensioni.

In quella centrale invece troviamo l'elenco dei prodotti e dei menù, caratterizzati da ingredienti, foto(per i prodotti) e prezzi. Sarà inoltre possibile ordinare un prodotto, aggiungendone la quantità nel relativo campo e premendo +. (Se non sei un Utente, mancherà la possibilità di poter ordinare).


Nella colonna di sinistra troviamo quello che potremmo definire un carrello, è presente il riepilogo della merce selezionata, il prezzo del trasporto imposto dal locale e i proprietari.

Infine è possibile procedere all'ordine cliccando il relativo bottone, che conduce ad una pagina del seguente tipo:



In questa schermata si ottiene un rapido riepilogo dell'ordine, la scelta dell'orario di consegna e il metodo di pagamento: *alla consegna* o *carta di credito*.

Una volta confermato si ottiene l'effettiva pagina di riepilogo, contenente tutte le informazioni necessarie per la gestione dell'ordine (sia per gli utenti che per i commercianti, in quest'ultimo caso le informazioni della carta saranno nascoste). Rispetto alla precedente pagina, alcune delle informazioni aggiuntive riguardano, per esempio, l'indirizzo del cliente e l'ora di consegna richiesta. Questa pagina potrà essere stampata, o digitalizzata, cliccando sull'apposito bottone.

 [Area Utente](#) [Esci](#) [Contattaci](#)


Ordine #51 effettuato a [QuadroByNight](#)
Situato in Via Abbondio Sangiorgio, 10 a Milano (20100)
Richiesto per le ore 15:00 il giorno Lunedì, 14 Gennaio 2019
Per conto di Calvin, Calvin Klein (calvin@klein.com 335448522)
Presso via marcona, 3 a Milano (20100)
Elenco prodotti

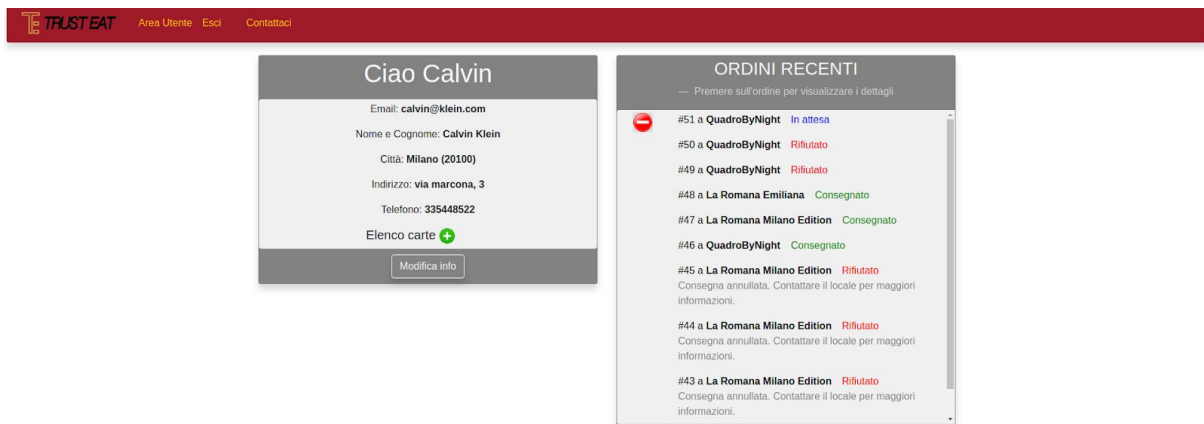
- Uramaki 6 pz. 8,00 € x 1 → 8,00 €
- Patatine fritte 3,00 € x 1 → 3,00 €

Elenco menu

- Piadina al volo 5,50 € x 1 → 5,50 €

Costo di consegna: 3,0 €
Totale da pagare: 19,5 €
Metodo di pagamento: alla consegna
[Stampa copia](#)

Uno storico degli ordini effettuato dall'utente è comunque presente nell'*area utente* (un commerciante invece otterrà l'elenco dei propri locali).



Da questa schermata è possibile, oltre ad aggiungere o eliminare carte di credito, anche eliminare gli ordini che non sono stati ancora accettati (o rifiutati) da parte del commerciante.

Ecco,invece, come si presenta l'area utente per il commerciante:



Qui è possibile scorrere l'elenco dei locali amministrati avendo anche la possibilità di aggiungerne dei nuovi.

Esempio di come il commerciante vede un proprio locale:

The screenshot displays the Trust Eat interface for a restaurant named "QUADROBYNIGHT". On the left, a sidebar contains management options: "Gestisci gli ordini", "Modifica il locale", "Modifica i prodotti", "Modifica i menu", and "Elimina il locale". The main content area is divided into two sections: "PRODOTTI" (Products) and "MENU".

PRODOTTI

Panino kebab <i>pane, kebab</i>	3,50 €
Patatine fritte <i>patatine fritte, sale</i>	3,00 €
Piadina kebab <i>piadina, kebab</i>	4,00 €
Uramaki 6 pz. <i>riso, salmone, avocado</i>	8,00 €

MENU

Panino al volo! <i>Kebab + patatine</i>	5,00 €
Piadina al volo <i>piadina + patatine fritte</i>	5,50 €

On the right, a panel titled "Plachiamo la tua fame notturna!" provides restaurant details: "Aperto dalle 10:00 alle 23:59", "Aperto 7 giorni su 7", "Si trova in Via Abbondio Sangiorgia, 10 a Milano (20100)", "CONTATTI", "Telefono: 3665482545", "Sito web: www.quadrobynight.it", "Email: quadrobynight@gmail.com", and "Voto: 4,0 su 2 recensioni!". Below this is a photograph of the restaurant's interior.

Dalla lateral bar il commerciante potrà gestire il suo locale o scegliere se eliminarlo.

Cliccando sul pulsante relativo alla gestione degli ordini si è rediretti alla seguente pagina, nella quale è possibile accettare o rifiutare gli ordini richiesti dagli utenti.

The screenshot shows the Trust Eat interface for "QUADROBYNIGHT" with the "ORDINI IN ATTESA" (Orders in Progress) section active. The main content area displays "Nessun ordine in attesa..." (No orders in progress...). On the right, a panel titled "Da consegnare" (To be delivered) shows a list of orders. The first order is #51, with an amount of 19,5 € (including delivery), a delivery time of 15:00, and a location of "via marcona, 3 a Milano (20100)". Below the list is a map showing the location of the orders. The map includes labels for various cities and regions, such as "Busto Arsizio", "Monza", "Milano", "Novara", "Pavia", "Cremona", "Lombardia", "Emilia-Romagna", "Toscana", "Lazio", "Abruzzo", "Molise", "Basilicata", "Calabria", "Sicilia", and "Sardinia".

La mappa sottostante permette una più semplice visualizzazione degli ordini da consegnare rispetto al locale considerato.

Nella colonna a destra è invece possibile scorrere l'elenco degli ordini *Da consegnare*, *Consegnati* o *Rifiutati* utilizzando l'apposito bottone di selezione.

N.B. *None in None* rappresentano rispettivamente la distanza ed il tempo di consegna, implementazioni funzionanti ma disabilitate di default. Per maggiori dettagli in merito riferirsi a [Organizzazione del codice](#) nel file **maps.py**.

Infine, il sito offre anche la possibilità agli utenti di lasciare delle recensioni sui locali presso i quali hanno effettuato gli ordini (e ai commercianti è data la possibilità di risposta). La schermata ha il seguente aspetto:

The screenshot displays the 'QUADROBYNIGHT' website interface. At the top, a dark red navigation bar contains the 'TRUSTEAT' logo and links for 'Area Utente', 'Esci', and 'Contattaci'. Below this, a red banner features the 'QUADROBYNIGHT' logo in yellow. The main content area is titled 'RECENSIONI' (Reviews) and lists three reviews:

Rating	Review Text	User	Date
3	Cibo accettabile, consegna puntuale ma un po' caro per la qualità che offre	Louis	13 Gennaio 2019
4	Grazie per il suo feedback, cercheremo di migliorare!		14 Gennaio 2019
5	Tutto perfetto!	Calvin	13 Gennaio 2019

Below the reviews, there is a section titled 'Rispondi alle recensioni...' (Respond to reviews...). It includes a dropdown menu to 'Seleziona l'utente a cui rispondere*' (Select the user to respond to*), a text input field for 'Risposta*' (Response*), and a button labeled 'INSERISCI' (Post).

N.B. Questa è la schermata vista da un commerciante proprietario del locale, quella dell'utente è molto simile, in quanto solamente il form inferiore è differente. Gli altri tipi di utenti (utenti anonimi, commercianti non proprietari del locale visitato, amministratori) non avranno la possibilità di scrivere commenti o rispondere, ma solo di visualizzare le recensioni, pertanto il form sottostante non sarà presente nelle loro pagine.

JavaScript

Di seguito sono elencate le funzioni create nel file `assets\js\functions.js` per la gestione e il controllo delle interfacce web.

function `autoReload(value)`: Questa funzione permette il reindirizzamento automatico in **value** millisecondi.

function `checkStatusPayment()`: Se l'utente ha selezionato Carta di credito" come metodo di pagamento, abilita la scelta di quest'ultima.

function `checkValues()`: Controlla che l'orario di consegna richiesto sia compreso tra l'orario di apertura e quello di chiusura del locale, considerando anche l'orario corrente. Controlla inoltre, in caso l'utente abbia scelto di pagare con carta, che essa sia stata effettivamente selezionata.

In caso i dati immessi siano errati il pulsante viene disabilitato per impedire la prosecuzione dell'ordine finchè essi non siano corretti.

function `stampa()`: Permette la digitalizzazione e/o la stampa di una pagina.

function `setDate()`: Imposta, al caricamento della pagina, la data attuale come data di scadenza della carta di credito, nel form di registrazione utente.

function `checkCard()`: Nel form di registrazione utente, qualora l'utente abbia espresso la volontà di aggiungere una carta per gli acquisti, controlla che sia stato inserito il numero di carta e che esso sia di 16 cifre. In caso di errore, viene mostrato un alert e disabilito il bottone, inibendo così all'utente la possibilità di creare un account.

function `checkIntestatario()`: Nel form di registrazione utente, qualora l'utente abbia espresso la volontà di aggiungere una carta per gli acquisti, controlla che l'intestatario della carta sia stato inserito. In caso di errore, viene mostrato un alert e disabilito il bottone, inibendo così all'utente la possibilità di creare un account.

function `checkData()`: Nel form di registrazione utente, qualora l'utente abbia espresso la volontà di aggiungere una carta per gli acquisti, controlla che la data di scadenza venga inserita, e che non sia nel passato. In caso di errore, viene mostrato un alert e disabilito il bottone, inibendo così all'utente la possibilità di creare un account.

function *checkChoice()*: Nel form di registrazione utente, controlla se l'utente ha espresso o meno la propria volontà di aggiungere o meno una carta di credito. In caso di errore, mostra un alert disabilitando il bottone, inibendo così all'utente la possibilità di creare un account.

function *checkPassword()*: Nel form di registrazione utente, controlla che le due password inserite siano uguali e che essa sia almeno di 8 caratteri. In caso di errore, viene mostrato un alert e disabilitato il bottone, inibendo così all'utente la possibilità di creare un account.

function *check_check()*: Nel form di registrazione utente, controlla che tutti i controlli precedenti siano andati a buon fine, ed in tal caso, ripristina il bottone, permettendo così all'utente di completare la registrazione.

function *conf_del(question)*: Funzione che mostra un pop-up che richiede la conferma dell'operazione. Utilizzato per operazione delicate come eliminazione di dati o gestione degli ordini.

function *selection_view(button_selected)*: Funzione che permette la visualizzazione dei tre tipi di ordini (*Da consegnare, Consegnati, Rifiutati*) nella pagina di gestione degli ordini di un locale.

Le successive funzioni permettono la corretta visualizzazione della mappa.

Per questo motivo sono entrambe poste nel file

order/templates/order/list_order.html

function *initMap()*: Funzione che si occupa di inizializzare la mappa di Google Maps: imposta il centro della visuale sul locale considerato e applica il segnalino sulle sue coordinate.

function *setMarkers(map)*: Funzione che aggiunge alla mappa dei segnalini relativi alle posizioni dei vari ordini che il locale considerato deve consegnare.

Test

localList

Il seguente test serve per verificare che gli orari, di apertura e chiusura di un locale siano corretti: l'orario di apertura dev'essere precedente all'orario di chiusura, infatti è stato deciso di gestire gli ordini e i locali a giornata.

- `test_orario_lower()` : mostra il caso l'orario sia corretto.
- `test_orario_equal()` : mostra il caso in cui i due orari inseriti siano uguali, tale caso è stato ritenuto non ammissibile.
- `test_orario_greater()` : mostra il caso in cui l'orario di apertura risulta posteriore all'orario di chiusura.

```
class LocalTests(TestCase):
```

```
    def test_orario_lower(self):
        Localita.objects.create(
            cap='41100',
            nome_localita='Modena')
        local = Locale(
            nome_locale='Prova',
            orario_apertura='10:50',
            orario_chiusura='13:50',
            cap=Localita.objects.get(nome_localita='Modena'),
            via='viale Italia', num_civico=3,
            telefono='059000000',)
        self.assertEqual(local.correct_open_close_time(), True)
```

```
    def test_orario_equal(self):
        Localita.objects.create(
            cap='41100',
            nome_localita='Modena')
        local = Locale(
            nome_locale='Prova',
            orario_apertura='13:50',
            orario_chiusura='13:50',
            cap=Localita.objects.get(nome_localita='Modena'),
            via='viale Italia', num_civico=3,
            telefono='059000000',)
        self.assertEqual(local.correct_open_close_time(), False)
```

```

def test_orario_greater(self):
    Localita.objects.create(
        cap='41100',
        nome_localita='Modena')
    local = Locale(
        nome_locale='Prova',
        orario_apertura='13:50',
        orario_chiusura='10:50',
        cap=Localita.objects.get(nome_localita='Modena'),
        via='viale Italia',
        num_civico=3,
        telefono='059000000',)
    self.assertEqual(local.correct_open_close_time(), False)

```

Order

Il seguente test serve per verificare che il valore impostato nel campo data (destinato a definire il giorno in cui l'ordine è effettuato) sia corrispondente alla data corrente.

- `test_order_valid()` : mostra il caso la data sia corretta.
- `test_order_future()` : mostra il caso in cui la data inserita sia futura alla data corrente.
- `test_order_past()` : mostra il caso in cui la data inserita sia passata rispetto alla data corrente.

```

class OrdersTests(TestCase):

```

```

    def test_order_valid(self):
        ordine = OrdineInAttesa(
            email_id=2,
            data=datetime.datetime.now(),
            orario_richiesto='23:00',
            metodo_pagamento='alla consegna')
        self.assertEqual(ordine.check_order_data(), True)

```

```

    def test_order_future(self):
        ordine = OrdineInAttesa(
            email_id=2,
            data=datetime.datetime.now() +
                datetime.timedelta(days=5),

```

```

        orario_richiesto='23:00',
        metodo_pagamento='alla consegna')
self.assertEqual(ordine.check_order_data(), False)

def test_order_past(self):
    ordine = OrdineInAttesa(
        email_id=2,
        data=datetime.datetime.now() -
            datetime.timedelta(days=5),
        orario_richiesto='23:00',
        metodo_pagamento='alla consegna')
self.assertEqual(ordine.check_order_data(), False)

```

User

Il seguente test serve per verificare che la carta di credito inserita non sia scaduta (rispetto alla data corrente).

Nella data il giorno non è tenuto in considerazione nel calcolo, e in caso mese e anno coincidano con quelli correnti, è stato deciso di considerare la carta valida fino alla fine del mese.

- `test_card_valid()` : mostra il caso in cui la carta sia valida e lontana dalla scadenza.
- `test_card_past()` : mostra il caso in cui la carta risulti scaduta da tempo.
- `test_card_equal()` : mostra il caso in cui la scadenza coincida con l'anno e il mese corrente.

```

class CardsTests(TestCase):

    def test_card_valid(self):
        card = CartaDiCredito(
            numero_carta='1435134134165432',
            intestatario='prova',
            scadenza=datetime.date(2030, 12, 1))
self.assertEqual(card.is_valid(), True)

    def test_card_past(self):
        card = CartaDiCredito(
            numero_carta='1435134134165432',
            intestatario='prova',
            scadenza=datetime.date(2016, 12, 1))
self.assertEqual(card.is_valid(), False)

```

```
def test_card_equal(self):  
    card = CartaDiCredito(  
        numero_carta='1435134134165432',  
        intestatario='prova',  
        scadenza=datetime.datetime.today())  
    self.assertEqual(card.is_valid(), True)
```