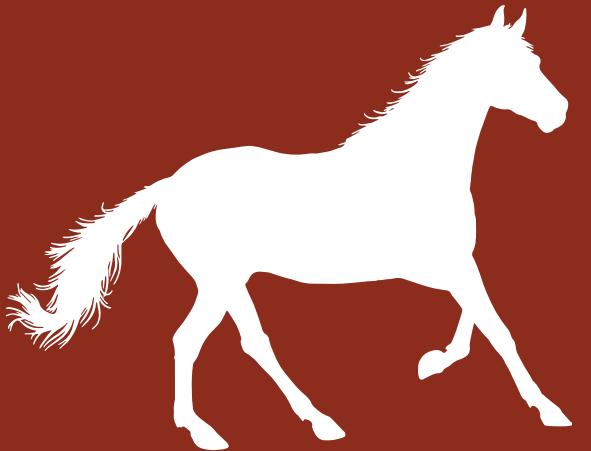


STALLIONS

Steak House



STALLIONS

Steak House

STALLIONS

**PRONTI PER VEDERE IL PROGRAMMA CHE
GESTISCE LA NOSTRA STEAKHOUSE?**

STALLIONS

Steak House

The screenshot shows a code editor window with a dark theme. The title bar reads "STALLIONS MAIN". The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar with the text "tpsit stalla". The left sidebar has icons for file operations like Open, Save, Find, and Run. The main pane displays Python code for a file named "main.py". The code defines a class "MenuApp" that initializes a window titled "STALLIONS MENU" with a size of 1200x800 and a background color "#f2e6d9". It imports "tkinter" and "ttk" modules, and uses "messagebox" from "tkinter". It also imports "create_complete_menu", "customer", "register_user", and "login_user" from "models" and "gestione" respectively, and "load_images" from "immagini". The "create_widgets" method handles the creation of a logo image and label, a title label, and a menu canvas with a scrollbar.

```
File Edit Selection View Go Run Terminal Help ← → tpsit stalla
src/main.py • immagini.py images.rar
1 import tkinter as tk
2 from tkinter import ttk
3 from tkinter import messagebox
4 from models import create_complete_menu, customer
5 from gestione import register_user, login_user
6 from immagini import load_images
7
8 # Finestra di Menu
9 class MenuApp:
10     def __init__(self, root, customer=None):
11         self.root = root
12         self.root.title("STALLIONS MENU")
13         self.root.geometry("1200x800")
14         self.root.configure(bg="#f2e6d9")
15
16         self.menu = create_complete_menu() # Assicurati che il menu venga caricato correttamente
17         self.images = load_images() # Carica le immagini dei piatti
18         self.current_order = [] # Lista per tenere traccia dei piatti selezionati
19         self.order_quantities = {} # Dizionario per tenere traccia delle quantità degli ordini
20         self.customer = customer # Cliente loggato
21
22         self.create_widgets()
23
24     def create_widgets(self):
25         try:
26             self.logo_image = tk.PhotoImage(file="stallone.png") # Assicurati che il percorso dell'immagine sia corretto
27             self.logo_label = tk.Label(self.root, image=self.logo_image, bg="#f2e6d9")
28             self.logo_label.pack(pady=20) # Aggiungi il logo sopra il titolo
29         except Exception as e:
30             print(f"Errore nel caricare l'immagine del logo: {e}")
31
32         self.title_label = tk.Label(self.root, text="STALLIONS MENU", font=("Courier New", 28, "bold"), bg="#8b4513", fg="#fff")
33         self.title_label.pack(fill="x", pady=10)
34
35         self.menu_canvas = tk.Canvas(self.root, bg="#f2e6d9", highlightthickness=0)
36         self.menu_scrollbar = ttk.Scrollbar(self.root, orient="vertical", command=self.menu_canvas.yview)
37
Ln 23, Col 30 Spaces: 0 0 △ 0 0 0 0
```

STALLIONS MAIN

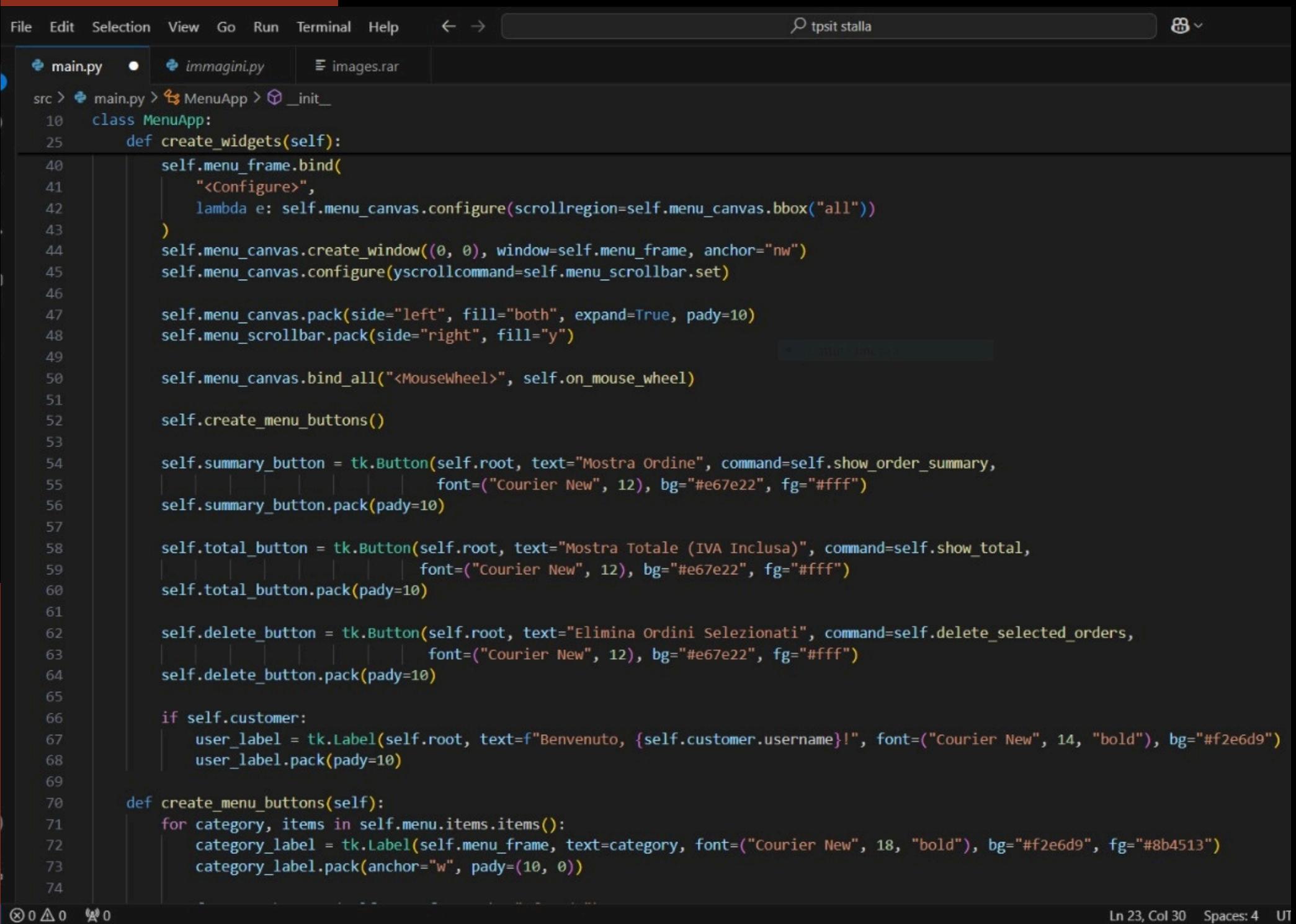
Questo codice crea un'applicazione grafica in Python utilizzando la libreria tkinter per un sistema di ordinazioni di un ristorante. L'app include una schermata di login/registrazione, una schermata del menu dove il cliente può fare ordini, e funzionalità per visualizzare e modificare l'ordine. Andiamo a dividere il main in diverse parti per mostrarvi le sue funzionalità

In questo spezzone di codice abbiamo principalmente importato le librerie per creare l'interfaccia gui e gestire la logica dell'applicazione, poi abbiamo creato la classe che rappresenta la finestra del menu, creato il costruttore, caricato il menu e le immagini e aggiunto i widgets e aggiunto i canvas del menu:

- `self.menu_canvas`: Questo crea un widget canvas con un colore di sfondo specifico e senza bordo di evidenziazione.
- `self.menu_scrollbar`: Questo crea una barra di scorrimento verticale collegata al canvas, permettendo lo scorrimento verticale degli elementi del menu.

STALLIONS

Steak House



```
File Edit Selection View Go Run Terminal Help ↶ → ○ tpsit stalla
main.py immagini.py images.rar
src > main.py > MenuApp > __init__
10  class MenuApp:
11      def __init__(self):
12          self.root = tk.Tk()
13          self.root.title("STALLIONS MAIN")
14          self.root.geometry("1200x600")
15
16          self.menu_frame = tk.Frame(self.root, bg="#f2e6d9", width=200, height=400)
17          self.menu_frame.pack(side="left", fill="both", expand=True)
18
19          self.menu_canvas = tk.Canvas(self.menu_frame, bg="#fff", width=180, height=350)
20          self.menu_canvas.pack(side="right", fill="y")
21
22          self.menu_scrollbar = tk.Scrollbar(self.menu_frame, orient="vertical", command=self.menu_canvas.yview)
23          self.menu_scrollbar.pack(side="right", fill="y")
24
25          self.menu_canvas.bind_all("<Configure>", lambda e: self.menu_canvas.configure(scrollregion=self.menu_canvas.bbox("all")))
26
27          self.menu_canvas.create_window((0, 0), window=self.menu_frame, anchor="nw")
28
29          self.menu_canvas.configure(yscrollcommand=self.menu_scrollbar.set)
30
31          self.menu_frame.bind_all("<MouseWheel>", self.on_mouse_wheel)
32
33          self.create_menu_buttons()
34
35          self.summary_button = tk.Button(self.root, text="Mostra Ordine", command=self.show_order_summary,
36                                         font=("Courier New", 12), bg="#e67e22", fg="#fff")
37          self.summary_button.pack(pady=10)
38
39          self.total_button = tk.Button(self.root, text="Mostra Totale (IVA Inclusa)", command=self.show_total,
40                                         font=("Courier New", 12), bg="#e67e22", fg="#fff")
41          self.total_button.pack(pady=10)
42
43          self.delete_button = tk.Button(self.root, text="Elimina Ordini Selezionati", command=self.delete_selected_orders,
44                                         font=("Courier New", 12), bg="#e67e22", fg="#fff")
45          self.delete_button.pack(pady=10)
46
47          if self.customer:
48              user_label = tk.Label(self.root, text=f"Benvenuto, {self.customer.username}!", font=("Courier New", 14, "bold"), bg="#f2e6d9")
49              user_label.pack(pady=10)
50
51          def create_menu_buttons(self):
52              for category, items in self.menu.items.items():
53                  category_label = tk.Label(self.menu_frame, text=category, font=("Courier New", 18, "bold"), bg="#f2e6d9", fg="#8b4513")
54                  category_label.pack(anchor="w", pady=(10, 0))
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
```

Ln 23, Col 30 Spaces: 4 UT

STALLIONS MAIN

- In questo secondo spezzone di main troviamo:
Creazione del Frame del Menu: Viene creato un frame (menu_frame) all'interno del canvas (menu_canvas) con un colore di sfondo specifico.
- Configurazione dello Scorrimento: Il canvas viene configurato per aggiornare la regione di scorrimento in base alle dimensioni del frame (menu_frame), e viene associato a una scrollbar verticale (menu_scrollbar).
- Creazione della Finestra nel Canvas: Viene creata una finestra nel canvas con menu_frame come contenuto, posizionata in alto a sinistra.
- Scroll con il Mouse: Viene abilitato lo scorrimento del contenuto del canvas usando la rotella del mouse.
- Pulsanti per le Azioni dell'Utente: Vengono creati e posizionati tre pulsanti per mostrare l'ordine, il totale (IVA inclusa), e eliminare gli ordini selezionati.
- Messaggio di Benvenuto: Se un cliente è loggato, viene creato un messaggio di benvenuto con il nome utente del cliente.
- Etichette delle Categorie del Menu: Viene creato un metodo per generare etichette delle categorie di piatti nel menu e posiziarle nel frame del menu.

STALLIONS

Steak House

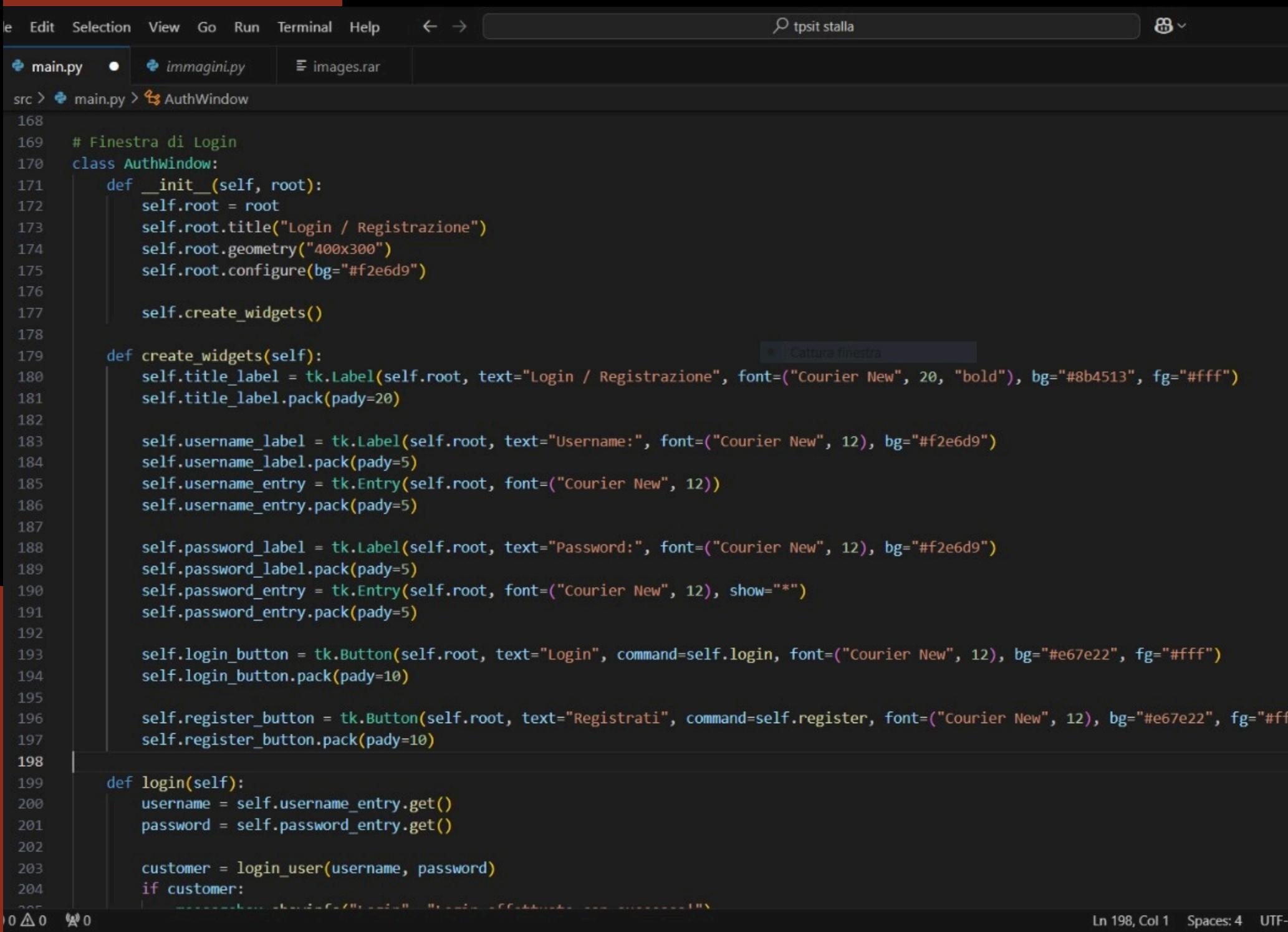
```
File Edit Selection View Go Run Terminal Help ↶ ↷ 🔍 tpsit stalla
main.py immagini.py images.rar
src > main.py > MenuApp > __init__
10  class MenuApp:
70      def create_menu_buttons(self):
77          for item in items:
78              image = self.images.get(item.name)
79              button = tk.Button(frame, text=f"{item.name} - {item.price}€", image=image, compound="left",
80                                 command=lambda i=item: self.add_item_to_order(i),
81                                 font=("Courier New", 12), bg="#e67e22", fg="#fff")
82              button.pack(side="left", padx=5, pady=5)
83              button.image = image
84
85      def add_item_to_order(self, item):
86          # Aggiungi o aggiorna la quantità dell'item
87          if item.name in self.order_quantities:
88              self.order_quantities[item.name] += 1
89          else:
90              self.order_quantities[item.name] = 1
91
92          # Aggiungi l'item alla lista dell'ordine
93          self.current_order.append(item)
94
95      def show_order_summary(self):
96          summary_window = tk.Toplevel(self.root)
97          summary_window.title("Riepilogo Ordine")
98          summary_window.geometry("400x600")
99          summary_window.configure(bg="#f2e6d9")
100
101          summary_label = tk.Label(summary_window, text="Riepilogo Ordine", font=("Courier New", 18, "bold"), bg="#8b4513", fg="#fff")
102          summary_label.pack(fill="x", pady=10)
103
104          order_text = ""
105          for item in self.order_quantities:
106              quantity = self.order_quantities[item]
107              order_text += f"{item} - {quantity}x\n" # Formato come 'item x2'
108
109          order_label = tk.Label(summary_window, text=order_text, font=("Courier New", 12), bg="#f2e6d9", fg="#8b4513")
110          order_label.pack(pady=10)
111
Ln 23, Col 30 Spaces: 4
```

STALLIONS MAIN

- Creazione del Frame per ogni categoria di piatti:
Viene creato un frame (frame) all'interno di menu_frame con un colore di sfondo specifico, ancorato a sinistra.
- Creazione dei Pulsanti per i Piatti: Per ogni piatto in una categoria, viene creato un pulsante con il nome, il prezzo e l'immagine del piatto. Questi pulsanti sono posizionati nel frame con spazi tra di loro.
- Metodo per Aggiungere un Elemento all'Ordine:
Aggiorna la quantità dell'elemento nel dizionario order_quantities.
Aggiunge l'elemento alla lista current_order.
- Metodo per Mostrare il Riepilogo dell'Ordine:
Crea una finestra di riepilogo con titolo e dimensioni specifici.
Crea e visualizza un testo che mostra i piatti ordinati e le rispettive quantità.

STALLIONS

Steak House



The screenshot shows a code editor with a dark theme. The title bar says "tpsit stalla". The left sidebar shows files: "main.py" (selected), "immagini.py", and "images.rar". The main area contains Python code for a Tkinter application:

```
 168
169 # Finestra di Login
170 class AuthWindow:
171     def __init__(self, root):
172         self.root = root
173         self.root.title("Login / Registrazione")
174         self.root.geometry("400x300")
175         self.root.configure(bg="#f2e6d9")
176
177         self.create_widgets()
178
179     def create_widgets(self):
180         self.title_label = tk.Label(self.root, text="Login / Registrazione", font=("Courier New", 20, "bold"), bg="#8b4513", fg="#fff")
181         self.title_label.pack(pady=20)
182
183         self.username_label = tk.Label(self.root, text="Username:", font=("Courier New", 12), bg="#f2e6d9")
184         self.username_label.pack(pady=5)
185         self.username_entry = tk.Entry(self.root, font=("Courier New", 12))
186         self.username_entry.pack(pady=5)
187
188         self.password_label = tk.Label(self.root, text="Password:", font=("Courier New", 12), bg="#f2e6d9")
189         self.password_label.pack(pady=5)
190         self.password_entry = tk.Entry(self.root, font=("Courier New", 12), show="*")
191         self.password_entry.pack(pady=5)
192
193         self.login_button = tk.Button(self.root, text="Login", command=self.login, font=("Courier New", 12), bg="#e67e22", fg="#fff")
194         self.login_button.pack(pady=10)
195
196         self.register_button = tk.Button(self.root, text="Registrati", command=self.register, font=("Courier New", 12), bg="#e67e22", fg="#fff")
197         self.register_button.pack(pady=10)
198
199     def login(self):
200         username = self.username_entry.get()
201         password = self.password_entry.get()
202
203         customer = login_user(username, password)
204         if customer:
```

STALLIONS MAIN

In questa parte ci focalizziamo sulla finestra di login:

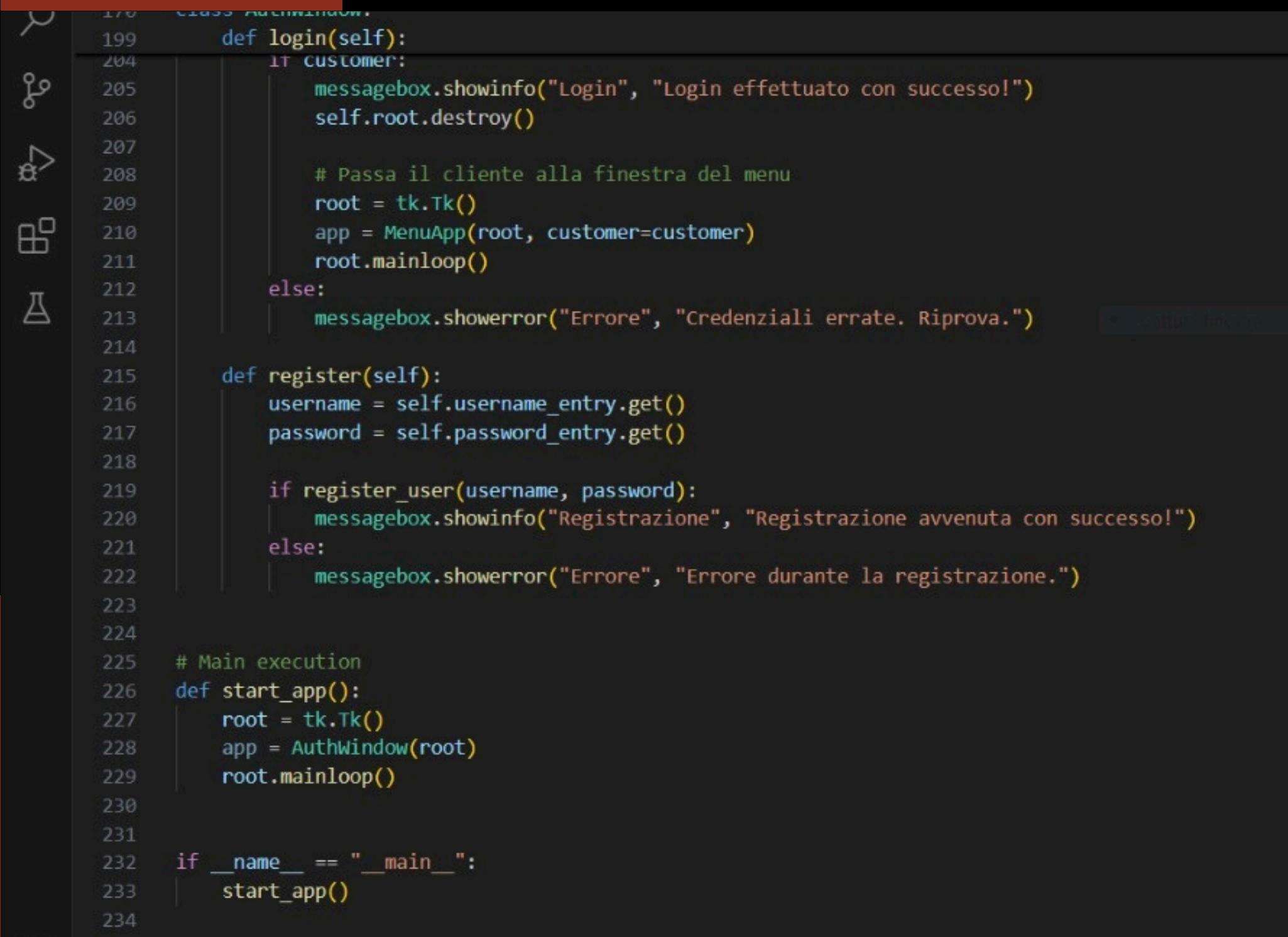
1. **Costruttore (`__init__`):**
 - Imposta il titolo, le dimensioni e il colore di sfondo della finestra principale.
 - Chiama il metodo `create_widgets()` per creare i widget.
2. **Metodo `create_widgets()`:**
 - Crea e posiziona i vari widget nella finestra, tra cui etichette, campi di input per l'username e la password, e pulsanti per il login e la registrazione.
3. **Metodo `login()`:**
 - Recupera i valori inseriti nei campi username e password.
 - Chiama la funzione `login_user()` per effettuare il login.
 - Se il login ha successo, mostra un messaggio di conferma e chiude la finestra di login.

Widget Creati

- Etichetta del Titolo: "Login / Registrazione".
- Campo di Input per Username: Con etichetta "Username".
- Campo di Input per Password: Con etichetta "Password", nasconde il testo inserito.
- Pulsante di Login: Avvia il metodo login.
- Pulsante di Registrazione: (presumibilmente) Avvia il metodo register.

STALLIONS

Steak House



```
170 CLASS AuthWindow:
171     def __init__(self):
172         self.root = tk.Tk()
173         self.username_entry = tk.Entry(self.root)
174         self.password_entry = tk.Entry(self.root, show="*")
175
176     def login(self):
177         customer = self.username_entry.get()
178         password = self.password_entry.get()
179
180         if register_user(customer, password):
181             messagebox.showinfo("Login", "Login effettuato con successo!")
182             self.root.destroy()
183
184             # Passa il cliente alla finestra del menu
185             root = tk.Tk()
186             app = MenuApp(root, customer=customer)
187             root.mainloop()
188
189         else:
190             messagebox.showerror("Errore", "Credenziali errate. Riprova.")
191
192     def register(self):
193         username = self.username_entry.get()
194         password = self.password_entry.get()
195
196         if register_user(username, password):
197             messagebox.showinfo("Registrazione", "Registrazione avvenuta con successo!")
198         else:
199             messagebox.showerror("Errore", "Errore durante la registrazione.")
200
201     # Main execution
202     def start_app():
203         root = tk.Tk()
204         app = AuthWindow(root)
205         root.mainloop()
206
207
208 if __name__ == "__main__":
209     start_app()
```

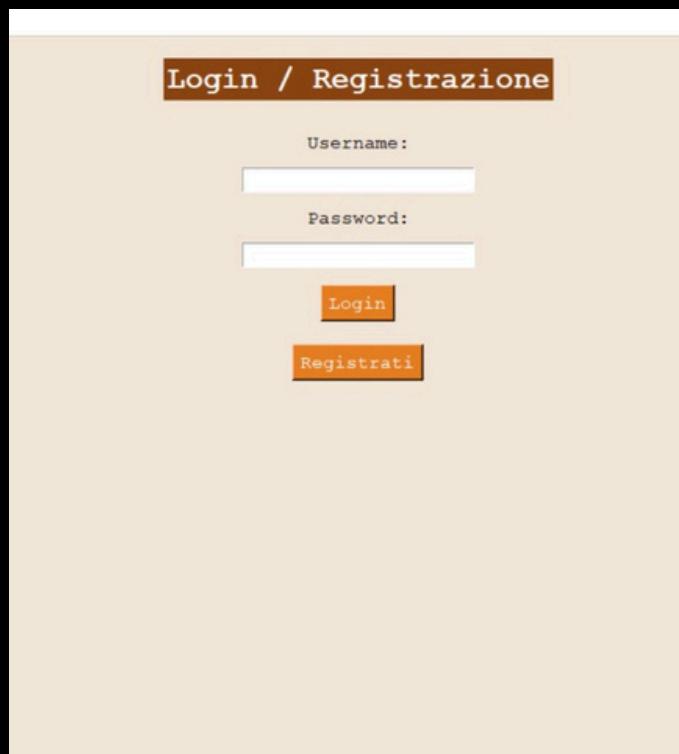
STALLIONS MAIN

Ed infine la parte finale del nostro main!

- Passaggio alla Finestra del Menu:
 - Se il login è avvenuto con successo, viene creata una nuova finestra principale (root) e viene avviata l'applicazione del menu (MenuApp) con il cliente autenticato. La finestra di login viene chiusa.
 - Se le credenziali sono errate, viene mostrato un messaggio di errore.
- Metodo register()
- Registrazione dell'Utente:
 - Recupera i valori inseriti nei campi username e password.
 - Chiama la funzione register_user() per registrare il nuovo utente.
 - Se la registrazione ha successo, viene mostrato un messaggio di conferma.
 - Se c'è un errore durante la registrazione, viene mostrato un messaggio di errore.
- Esecuzione Principale
- Funzione start_app():
 - Crea la finestra principale (root) e avvia l'applicazione di autenticazione (AuthWindow).
 - Esegue il loop principale dell'interfaccia (root.mainloop()).
- Punto di Ingresso del Programma:
 - Se il file viene eseguito direttamente, viene chiamata la funzione start_app().

Dopo aver visto il codice del main non spetta che vedere come si presenta!

INTERFACCIA MENU' STALLIONS



The screenshot displays the "STALLIONS MENU" application. The main window shows the following menu categories:

- Primi Piatti**: Includes a dish labeled "Classic - 12.0€" with a small image of meat.
- Secondi Piatti**: Includes two dishes: "Filetto Pepe Verde - 22.0€" and "Tagliata Black Angus - 25.0€", each with an image of the dish.
- Contorni**: Includes two sides: "Patatine Fritte - 5.0€" and "Insalata Mista - 6.0€", each with an image.
- Dessert**: Shows two dessert items, though their names are not clearly legible.

On the right side of the main window, there are several buttons: "Mostra Ordine", "Mostra Totale (IVA Inclusa)", "Cattura finestra", "Riepilogo Ordine", "Ina Ordini Selezionati", and "Benvenuto, 3!". A separate window titled "Riepilogo Ordine" shows the selected items: "Tagliata Black Angus - 1x" and "Filetto Pepe Verde - 1x". Another window titled "Totale con IVA" displays the total: "Totale: 47.00€", "IVA (22%): 10.34€", and "Totale con IVA: 57.34€".

STALLIONS

Steak House

STALLIONS MENU'

Per quanto riguarda il vero e proprio funzionamento del codice nel menu.py abbiamo utilizzato:

- MenuApp è una finestra principale che permette agli utenti di visualizzare un riepilogo degli ordini effettuati e vedere il totale dell'ordine con l'IVA.
- Le funzionalità di base come il riepilogo dell'ordine e la visualizzazione del totale sono presenti, ma non includono il calcolo dinamico degli ordini o l'elenco degli articoli selezionati.
- self.root: la finestra principale dell'applicazione. Ha il titolo "Stallions Menu", una dimensione di 1200x800 pixel e uno sfondo di colore beige (#f2e6d9).
- self.current_order: una lista vuota che sarà utilizzata per tenere traccia degli ordini effettuati dall'utente (anche se non è ancora implementata la selezione dei piatti).
- self.customer: un oggetto che rappresenta il cliente che ha effettuato il login. Se il cliente è passato alla finestra del menu, viene mostrato il suo nome.
- title_label: Un'etichetta che visualizza il titolo "STALLIONS MENU", con uno stile particolare (font "Courier New", grassetto e colore bianco su uno sfondo marrone).
- Se un cliente è loggato, viene mostrato il messaggio di benvenuto con il nome dell'utente (self.customer.username).
- summary_button: Un pulsante che, quando cliccato, mostra un riepilogo dell'ordine effettuato.
- total_button: Un altro pulsante che mostra il totale dell'ordine, comprensivo di IVA.

```
File Edit Selection View Go Run Terminal Help ↵ → tpsit stalla

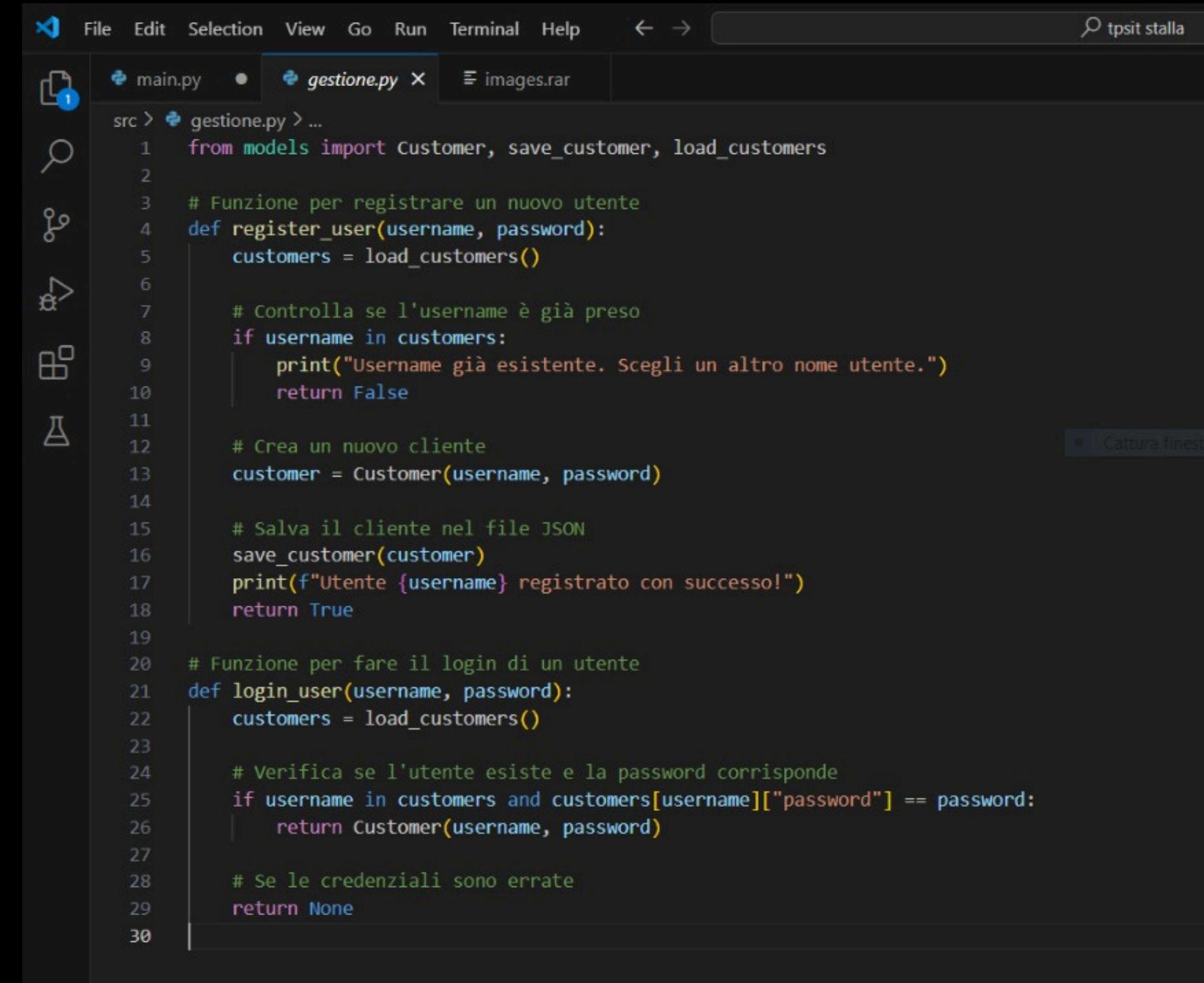
src > main.py < menu.py < images.rar
1 import tkinter as tk
2 from tkinter import ttk
3 from tkinter import messagebox
4 from gestione import register_user, login_user, customer
5
6 class MenuApp:
7     def __init__(self, root, customer=None):
8         self.root = root
9         self.root.title("STALLIONS MENU")
10        self.root.geometry("1200x800")
11        self.root.configure(bg="#f2e6d9")
12
13        self.current_order = [] # Lista per tenere traccia dei piatti selezionati
14        self.customer = customer # Cliente loggato
15
16        self.create_widgets()
17
18    def create_widgets(self):
19        # Titolo della finestra
20        self.title_label = tk.Label(self.root, text="STALLIONS MENU", font=("Courier New", 28, "bold"), bg="#8b4513", fg="#ffff")
21        self.title_label.pack(fill="x", pady=10)
22
23        # Visualizza il nome dell'utente
24        if self.customer:
25            user_label = tk.Label(self.root, text=f"Benvenuto, {self.customer.username}!", font=("Courier New", 14, "bold"), bg="#f2e6d9")
26            user_label.pack(pady=10)
27
28        # Aggiungi button per l'ordine e altre funzionalità
29        self.summary_button = tk.Button(self.root, text="Mostra Ordine", command=self.show_order_summary,
30                                         font=("Courier New", 12), bg="#ee7e22", fg="#ffff")
31        self.summary_button.pack(pady=10)
32
33        self.total_button = tk.Button(self.root, text="Mostra Totale (IVA Inclusa)", command=self.show_total,
34                                     font=("Courier New", 12), bg="#ee7e22", fg="#ffff")
35        self.total_button.pack(pady=10)
36
37    def show_order_summary(self):
38        summary_window = tk.Toplevel(self.root)
39        summary_window.title("Riepilogo Ordine")
40        summary_window.geometry("400x600")
41        summary_window.configure(bg="#f2e6d9")
42
43        summary_label = tk.Label(summary_window, text="Riepilogo Ordine", font=("Courier New", 18, "bold"), bg="#8b4513", fg="#ffff")
44        summary_label.pack(fill="x", pady=10)
45
46    def show_total(self):
47        total_window = tk.Toplevel(self.root)
48        total_window.title("Totale con IVA")
49        total_window.geometry("300x200")
50        total_window.configure(bg="#f2e6d9")
51
52        total_label = tk.Label(total_window, text="Totale: 50.00€", font=("Courier New", 14), bg="#f2e6d9", fg="#8b4513")
53        total_label.pack(pady=20)
54
```

STALLIONS GESTIONE

Questo codice si occupa della gestione della registrazione e del login di un utente, utilizzando un sistema basato su un file JSON per memorizzare i dati degli utenti. Di seguito, spiego ogni parte del codice.

`load_customers()`: Carica i dati degli utenti da un file (presumibilmente un file JSON) e restituisce un dizionario in cui le chiavi sono gli `username` e i valori sono i dati associati (ad esempio, la `password`).

`save_customer()` salva i dati del nuovo cliente (probabilmente nel file JSON) per far sì che l'utente possa essere recuperato in futuro.



The screenshot shows a code editor window with a dark theme. The title bar says "gestione.py". The left sidebar shows files: "main.py", "gestione.py", and "images.rar". The main area contains the following Python code:

```
File Edit Selection View Go Run Terminal Help ← → ⌂ tpsit stalla

src > gestione.py > ...
1  from models import Customer, save_customer, load_customers
2
3  # Funzione per registrare un nuovo utente
4  def register_user(username, password):
5      customers = load_customers()
6
7      # Controlla se l'username è già preso
8      if username in customers:
9          print("Username già esistente. Scegli un altro nome utente.")
10         return False
11
12      # Crea un nuovo cliente
13      customer = Customer(username, password)
14
15      # Salva il cliente nel file JSON
16      save_customer(customer)
17      print(f"Utente {username} registrato con successo!")
18      return True
19
20  # Funzione per fare il login di un utente
21  def login_user(username, password):
22      customers = load_customers()
23
24      # Verifica se l'utente esiste e la password corrisponde
25      if username in customers and customers[username]["password"] == password:
26          return Customer(username, password)
27
28      # Se le credenziali sono errate
29      return None
30
```

WHAT WOULD THE GESTIONE.PY BE LIKE IN JAVA?

```
C: > Users > pc > AppData > Local > Temp > Rar$Dla12600.43694.rartemp > gestione.java
1 import java.util.HashMap;
2 import java.util.Map;
3
4 class Customer {
5     private String username;
6     private String password;
7
8     public Customer(String username, String password) {
9         this.username = username;
10        this.password = password;
11    }
12
13    public String getUsername() {
14        return username;
15    }
16
17    public String getPassword() {
18        return password;
19    }
20}
21
22 public class UserAuthentication {
23
24     private static Map<String, Customer> customers = new HashMap<>();
25
26     public static boolean registerUser(String username, String password) {
27
28         if (customers.containsKey(username)) {
29             System.out.println("Username già esistente. Scegli un altro nome utente.");
30             return false;
31         }
32
33         Customer customer = new Customer(username, password);
34
35         customers.put(username, customer);
36         System.out.println("Utente " + username + " registrato con successo!");
37         return true;
38     }
39}
```

Introduction

Purpose: Demonstrates a basic system for:

- User registration with unique usernames and passwords.
- User login with credential validation.

Structure:

Registration: Adds new users to the system.

Login: Authenticates existing users.

Key Features

Data Storage: User data stored in a HashMap (username as key, Customer object as value).

Validation:

Registration: Prevents duplicate usernames.

Login: Validates username and password.

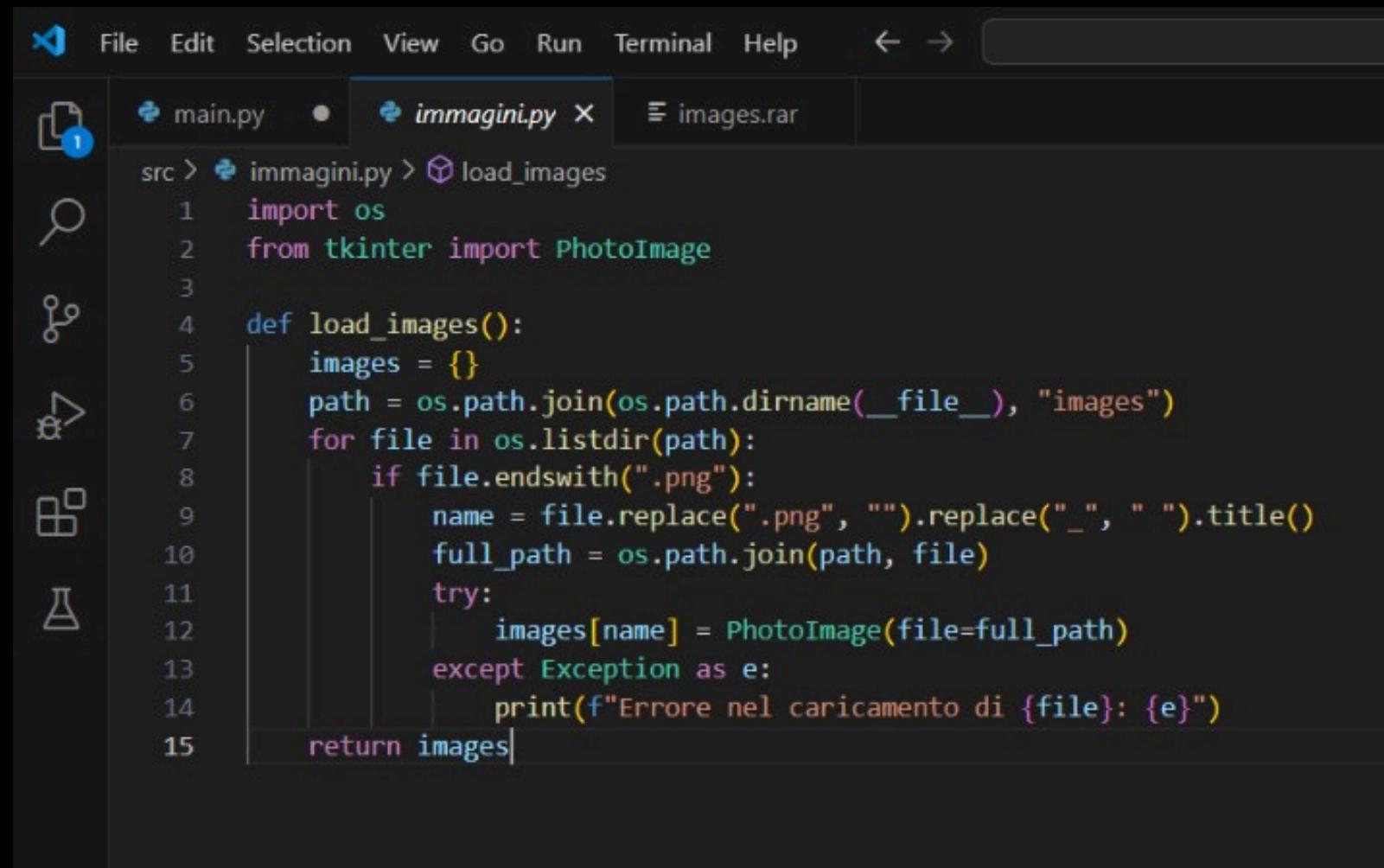
Feedback: Clear success and error messages guide the user.

Limitations

Security: Passwords stored in plain text (no encryption).

Scalability: Uses in-memory storage; unsuitable for large-scale applications.

Concurrency: HashMap is not thread-safe.



The image shows a code editor interface with two files open: `main.py` and `immagini.py`. The `immagini.py` file contains the following code:

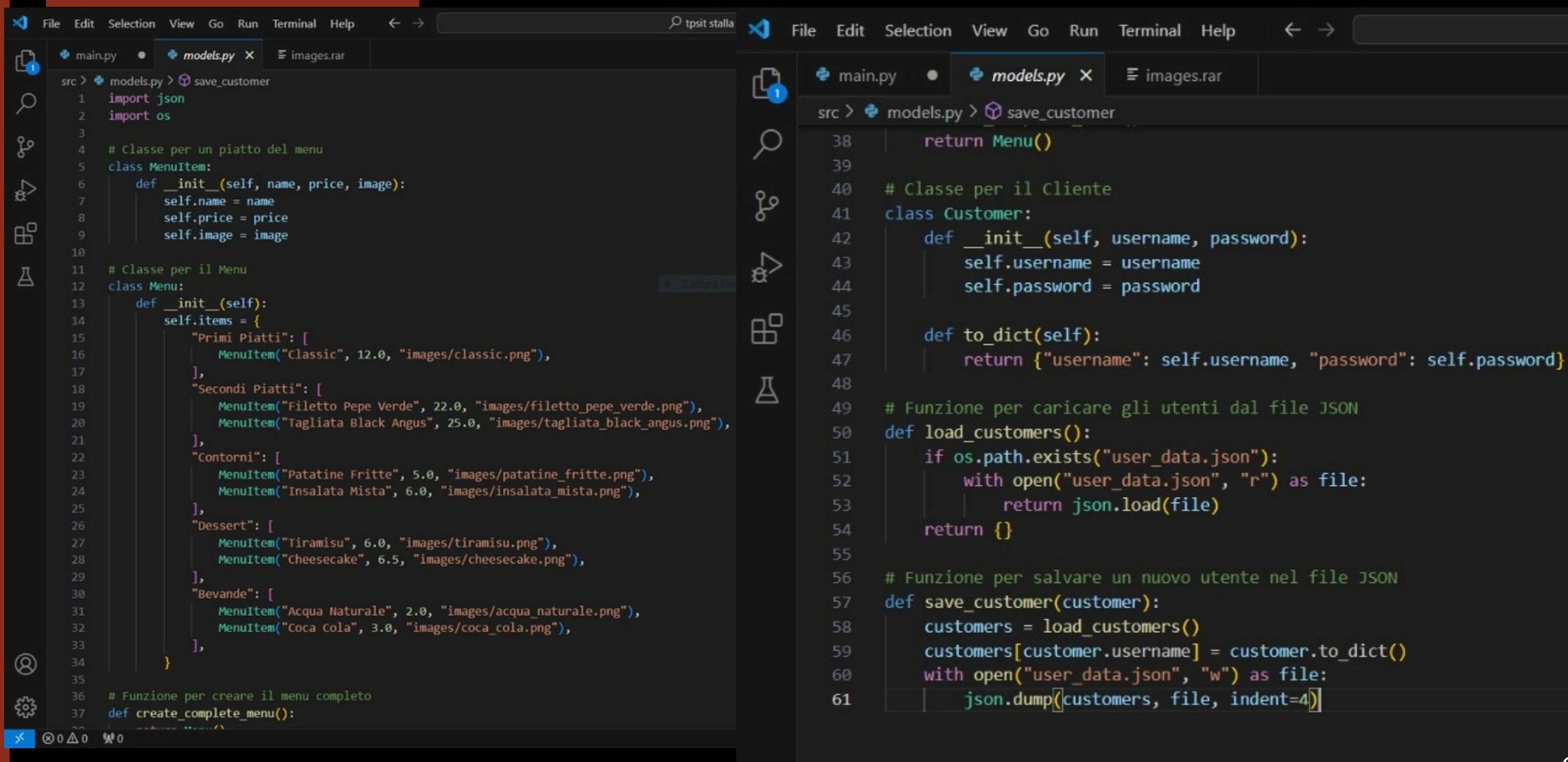
```
src > immagini.py > load_images
1 import os
2 from tkinter import PhotoImage
3
4 def load_images():
5     images = {}
6     path = os.path.join(os.path.dirname(__file__), "images")
7     for file in os.listdir(path):
8         if file.endswith(".png"):
9             name = file.replace(".png", "").replace("_", " ").title()
10            full_path = os.path.join(path, file)
11            try:
12                images[name] = PhotoImage(file=full_path)
13            except Exception as e:
14                print(f"Errore nel caricamento di {file}: {e}")
15
16 return images
```

HOW WE INSERTED THE IMAGES

- Purpose: The function `load_images()` loads PNG images from a specified directory and stores them in a dictionary.
- Steps:
 1. Initialize Dictionary: Creates an empty dictionary `images` to hold the images.
 2. Set Path: Constructs the path to the "images" directory.
 3. Iterate Through Files: Loops through all files in the directory and filters for PNG files.
 4. Format Names and Load Images: For each PNG file, formats the name, constructs the full path, and attempts to load the image using `PhotoImage`.
 5. Handle Errors: Prints an error message if there's an issue loading any image.
 6. Return Images: Returns the dictionary containing the loaded images.

STALLIONS

Steak House



```
File Edit Selection View Go Run Terminal Help < > tpsit stalla File Edit Selection View Go Run Terminal Help < > tpsit stalla
src > main.py > models.py > save_customer
src > models.py > save_customer
1 import json
2 import os
3
4 # Classe per un piatto del menu
5 class MenuItem:
6     def __init__(self, name, price, image):
7         self.name = name
8         self.price = price
9         self.image = image
10
11 # Classe per il Menu
12 class Menu:
13     def __init__(self):
14         self.items = {
15             "Primi Piatti": [
16                 MenuItem("Classic", 12.0, "images/classic.png"),
17             ],
18             "Secondi Piatti": [
19                 MenuItem("Filetto Pepe Verde", 22.0, "images/filetto_pepe_verde.png"),
20                 MenuItem("Tagliata Black Angus", 25.0, "images/tagliata_black_angus.png"),
21             ],
22             "Contorni": [
23                 MenuItem("Patatine Fritte", 5.0, "images/patatine_fritte.png"),
24                 MenuItem("Insalata Mista", 6.0, "images/insalata_mista.png"),
25             ],
26             "Dessert": [
27                 MenuItem("Tiramisu", 6.0, "images/tiramisu.png"),
28                 MenuItem("Cheesecake", 6.5, "images/cheesecake.png"),
29             ],
30             "Bevande": [
31                 MenuItem("Acqua Naturale", 2.0, "images/acqua_naturale.png"),
32                 MenuItem("Coca Cola", 3.0, "images/coca_colा.png"),
33             ],
34         }
35
36     # Funzione per creare il menu completo
37     def create_complete_menu():
38
39         # Funzione per restituire un oggetto Menu
40         return Menu()
41
42     # Classe per il Cliente
43     class Customer:
44         def __init__(self, username, password):
45             self.username = username
46             self.password = password
47
48         def to_dict(self):
49             return {"username": self.username, "password": self.password}
50
51     # Funzione per caricare gli utenti dal file JSON
52     def load_customers():
53         if os.path.exists("user_data.json"):
54             with open("user_data.json", "r") as file:
55                 return json.load(file)
56         return {}
57
58     # Funzione per salvare un nuovo utente nel file JSON
59     def save_customer(customer):
60         customers = load_customers()
61         customers[customer.username] = customer.to_dict()
62         with open("user_data.json", "w") as file:
63             json.dump(customers, file, indent=4)
```

STALLIONS MODELS.PY

La classe MenuItem rappresenta un singolo piatto del menu. Ogni oggetto MenuItem ha tre proprietà:

- name: Il nome del piatto (ad esempio, "Classic", "Filetto Pepe Verde").
- price: Il prezzo del piatto.
- image: Il percorso dell'immagine associata al piatto.

La classe ha un costruttore (`__init__`) che inizializza questi tre attributi quando viene creato un oggetto di tipo MenuItem.

La classe Menu rappresenta l'intero menu del ristorante. Contiene un dizionario di categorie (ad esempio "Primi Piatti", "Secondi Piatti", ecc.) come chiavi, e per ogni categoria, una lista di oggetti MenuItem.

Il costruttore della classe inizializza il dizionario `self.items` con alcune categorie di piatti e piatti predefiniti associati a ciascuna categoria. Ogni piatto è un'istanza della classe MenuItem.

Questa funzione "create_complete_menu" crea e restituisce un oggetto Menu, che rappresenta il menu completo del ristorante

load_customers

Questa funzione legge i dati degli utenti da un file JSON chiamato `user_data.json`. Se il file esiste, lo apre in modalità lettura e carica il contenuto in un dizionario. Se il file non esiste, restituisce un dizionario vuoto.

La funzione `save_customer` riceve un oggetto `Customer`, carica i dati degli utenti esistenti tramite la funzione `load_customers`, aggiunge il nuovo utente (usando il nome utente come chiave nel dizionario), e poi salva il dizionario aggiornato nel file JSON `user_data.json`.

- La funzione `customer.to_dict()` converte l'oggetto `Customer` in un dizionario.
- Il file JSON viene aperto in modalità scrittura e il dizionario degli utenti viene salvato con una formattazione leggibile (grazie all'argomento `indent=4` di `json.dump`).

FILE JSON

I file JSON (JavaScript Object Notation) sono un formato di dati basato su testo, usato per rappresentare e scambiare informazioni in modo semplice e leggibile. JSON è ampiamente utilizzato in applicazioni web, specialmente per il trasferimento di dati tra client e server.

Un file JSON contiene oggetti (coppie chiave-valore) e array (liste di valori). La sintassi è semplice:

Le chiavi sono stringhe racchiuse tra virgolette doppie.

I valori possono essere stringhe, numeri, booleani, oggetti, array o null.

Gli oggetti sono racchiusi tra parentesi graffe {}, mentre gli array sono tra parentesi quadre [].

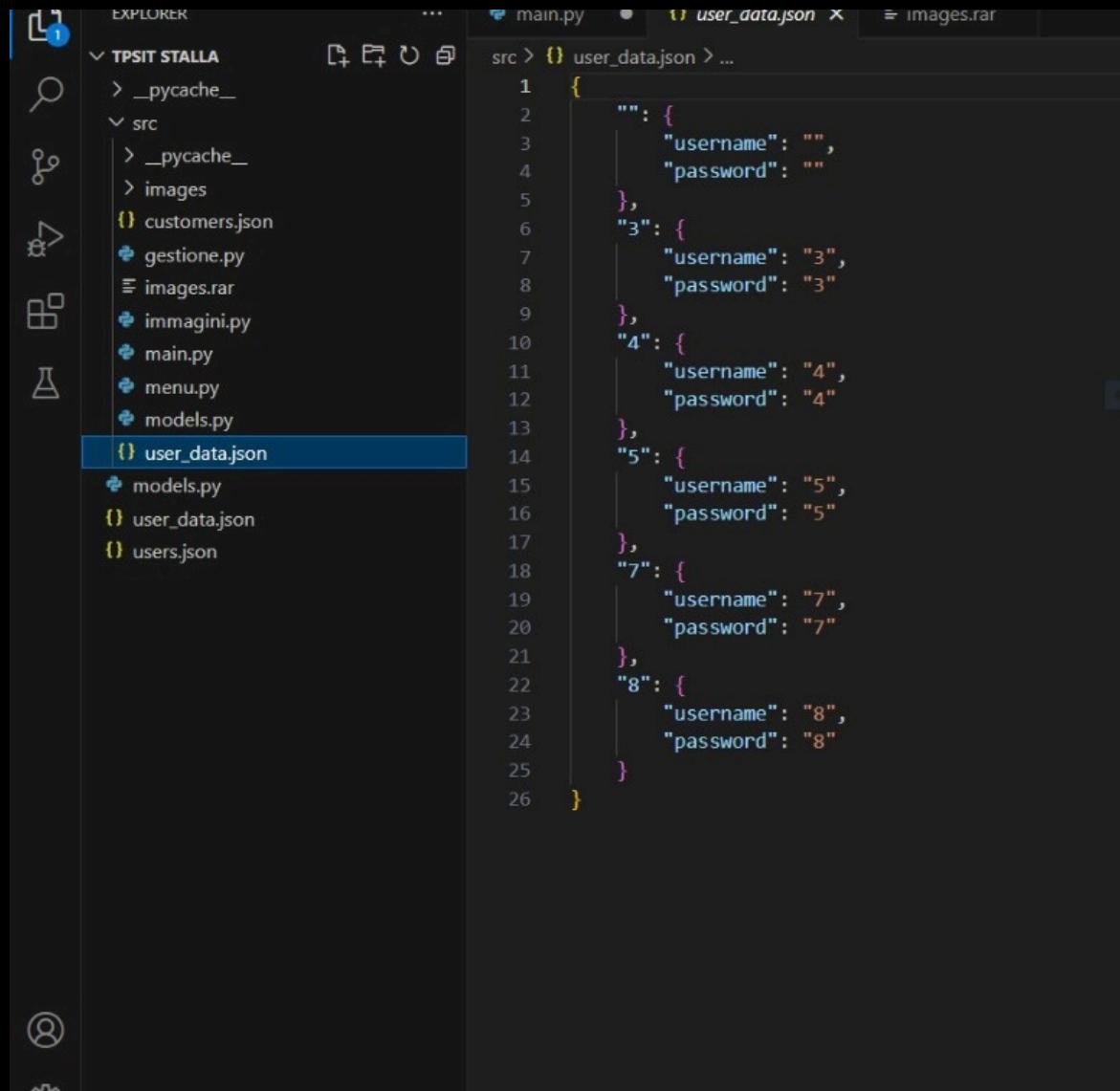
Vantaggi del JSON

- **Semplicità:** È un formato facilmente leggibile e scrivibile, sia per gli esseri umani che per i computer.
- **Indipendenza dal linguaggio:** Anche se JSON deriva da JavaScript, è indipendente dal linguaggio di programmazione. Molti linguaggi, come Python, Java, PHP, Ruby, e altri, supportano JSON nativamente.
- **Compatibilità con API web:** È il formato di scambio dati più comune nelle API RESTful, in cui i server e i client scambiano dati JSON tramite HTTP.

Tipi di Dati in JSON

- I tipi di dati supportati da JSON sono:
- **Stringhe:** Testo racchiuso tra virgolette doppie "Hello".
- **Numeri:** Possono essere interi o decimali.
- **Booleani:** true o false.
- **Oggetti:** Racchiusi tra parentesi graffe {}.
- **Array:** Racchiusi tra parentesi quadre [].
- **Null:** Rappresentato come null, usato per indicare un valore vuoto o mancante.

USER_DATA JSON



```
1  {
2   "": {
3     "username": "",
4     "password": ""
5   },
6   "3": {
7     "username": "3",
8     "password": "3"
9   },
10  "4": {
11    "username": "4",
12    "password": "4"
13  },
14  "5": {
15    "username": "5",
16    "password": "5"
17  },
18  "7": {
19    "username": "7",
20    "password": "7"
21  },
22  "8": {
23    "username": "8",
24    "password": "8"
25  }
26 }
```

Il file JSON è composto da un oggetto principale che contiene delle chiavi (ad esempio "d", "ed", "3", "5", "4") che rappresentano gli identificatori degli utenti. Ogni identificatore è associato a un altro oggetto che contiene le informazioni dell'utente, come username, password, e talvolta anche email.

JAVA CLASS

The screenshot shows a Java code editor with two tabs open: 'stall.java' and 'stall.java'. The code is identical in both tabs. It defines a class 'Steakhouse' with methods for managing a menu, orders, and customers.

```
src > stall.java
1 package controller;
2
3
4 import model.customer;
5 import model.Menuitem;
6 import model.order;
7
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class Steakhouse {
12     private List<Menuitem> menu;
13     private List<Order> orders;
14     private List<Customer> customers;
15
16     public Steakhouse() {
17         this.menu = new ArrayList<>();
18         this.orders = new ArrayList<>();
19         this.customers = new ArrayList<>();
20         populateMenu();
21     }
22
23     private void populateMenu() {
24         menu.add(new Menuitem("Bistecca di Wagyu di tipo A", 250.00));
25         menu.add(new Menuitem("Filetto di Vitello", 14.50));
26         menu.add(new Menuitem("Bistecca alla Fiorentina", 25.99));
27         menu.add(new Menuitem("Spare Pork Ribs", 20.00));
28         menu.add(new Menuitem("Costata Frisona Baltica", 24.00));
29         menu.add(new Menuitem("Costata di Manzo", 28.50));
30         menu.add(new Menuitem("Hamburger Gourmet", 15.75));
31         menu.add(new Menuitem("Bistecca di cavallo ", 18.00));
32         menu.add(new Menuitem("Patatine Fritte", 3.99));
33         menu.add(new Menuitem("Insalata Mista", 4.50));
34     }
35
36     public void printMenu() {
37         System.out.println("Menu:");
38         for (int i = 0; i < menu.size(); i++) {
39             System.out.println((i + 1) + ". " + menu.get(i));
40         }
41     }
42
43     public MenuItem getMenuItem(int index) {
44         if (index >= 0 && index < menu.size()) {
45             return menu.get(index);
46         }
47         return null;
48     }
49
50     public void addOrder(Order order) {
51         orders.add(order);
52     }
53
54     public void listOrders() {
55         System.out.println("Ordini Totali:");
56         for (int i = 0; i < orders.size(); i++) {
57             System.out.println("Ordine #" + (i + 1));
58             orders.get(i).printOrder();
59         }
60     }
61
62     public void addCustomer(Customer customer) {
63         customers.add(customer);
64     }
65
66     public Customer findCustomerById(String id) {
67         for (Customer customer : customers) {
68             if (customer.getId().equals(id)) {
69                 return customer;
70             }
71         }
72         return null;
73     }
74
75     public void listCustomers() {
76         System.out.println("Clienti:");
77         for (Customer customer : customers) {
78             System.out.println(customer);
79         }
80     }
81 }
```

```
public Customer findCustomerById(String id) {
    for (Customer customer : customers) {
        if (customer.getId().equals(id)) {
            return customer;
        }
    }
    return null;
}

public void listCustomers() {
    System.out.println("Clienti:");
    for (Customer customer : customers) {
        System.out.println(customer);
    }
}
```

"This slide highlights the Steakhouse class, which handles menu items, orders, and customers for a steakhouse. It includes methods for adding and listing items, making it an efficient tool for managing restaurant operations."

Purpose:

A Java class to manage a steakhouse's menu, customers, and orders.

Key Features:

Menu Management:

Predefined menu with items like steaks and sides, retrievable and displayable.

Order Handling:

Add and list customer orders for tracking.

Customer Management:

Add and find customers by unique IDs.

Key Methods:

populateMenu(), printMenu(), addOrder(Order)

listOrders(), addCustomer(Customer), findCustomerById(String id)

Attributes:

menu: Stores menu items.

orders: Tracks customer orders.

customers: Stores customer data.

PYTHON E JAVA

DIFFERENCES BETWEEN PYTHON AND JAVA

Python

- Syntax: Simpler and more readable, with less boilerplate code.
- Error Handling: Uses exceptions with try-except.
- Performance: Fast development but lower performance.
- Libraries: Strong support for science, data, and machine learning.
- Community: Large community focused on scripting and prototypes.

Java

- Syntax: More verbose and complex, but more structured and secure.
- Error Handling: Robust exception handling with compile-time security.
- Performance: Better performance, suitable for scalable applications.
- Libraries: Ideal for enterprise systems and distributed applications.
- Community: Strong support for enterprise applications, more complex for beginners.



STALLIONS

Steak House

GRAZIE
PER L'ATTENZIONE

REALIZZATO DA:

- AMODIO RAFFAELE;
- D'AURIA ALESSANDRO;
- SCHETTINO EMANUEL;
- HOWLADAR SAKIBUL