TableTales

Francesco Bellodi
Luglio 2023

1 Descrizione della traccia

1.1 Cos'è TableTales

TableTales è una web app, ideata sul modello di siti ben conosciuti come TripAdvisor.it. Tuttavia, a differenza di piattaforme più generaliste, TableTales concentra la sua attenzione esclusivamente sul settore della ristorazione.

1.2 Utenti

All'interno della piattaforma TableTales, sono stati definiti tre principali ruoli, ognuno dei quali è dotato di specifici privilegi che ne determinano le capacità di interazione con il sito:

- Guest: Il ruolo di "Guest" rappresenta un visitatore occasionale o un potenziale utente del sito. Senza la necessità di registrazione o accesso, il Guest ha il privilegio di navigare le pagine web della piattaforma. Può esplorare la lista dei ristoranti presenti, leggere le informazioni relative a ciascun locale e visualizzare i commenti e le recensioni rilasciate da altri utenti. Tuttavia, le sue azioni sono limitate alla visualizzazione, senza la possibilità di interagire attivamente.

- User: Un "User" è un visitatore registrato e autenticato. Oltre ai privilegi garantiti al Guest, l'User ha la possibilità di interagire in modo più profondo con la piattaforma. Può prenotare tavoli nei ristoranti desiderati e condividere le proprie esperienze attraverso la redazione di recensioni. Questo ruolo, quindi, consente una partecipazione attiva e una maggiore immersione nelle funzionalità offerte da TableTales.

- L'"Owner" rappresenta un utente speciale. Si tratta di un User che ha fatto un ulteriore passo, registrando il proprio locale sulla piattaforma. Questa registrazione conferisce all'Owner una serie di privilegi avanzati legati alla gestione del proprio ristorante o dei propri ristoranti. Ha la facoltà di modificare le informazioni relative al suo locale e, se necessario, eliminarlo. Tuttavia, pur avendo queste capacità amministrative, l'Owner conserva tutti i privilegi di un User standard. Ciò significa che

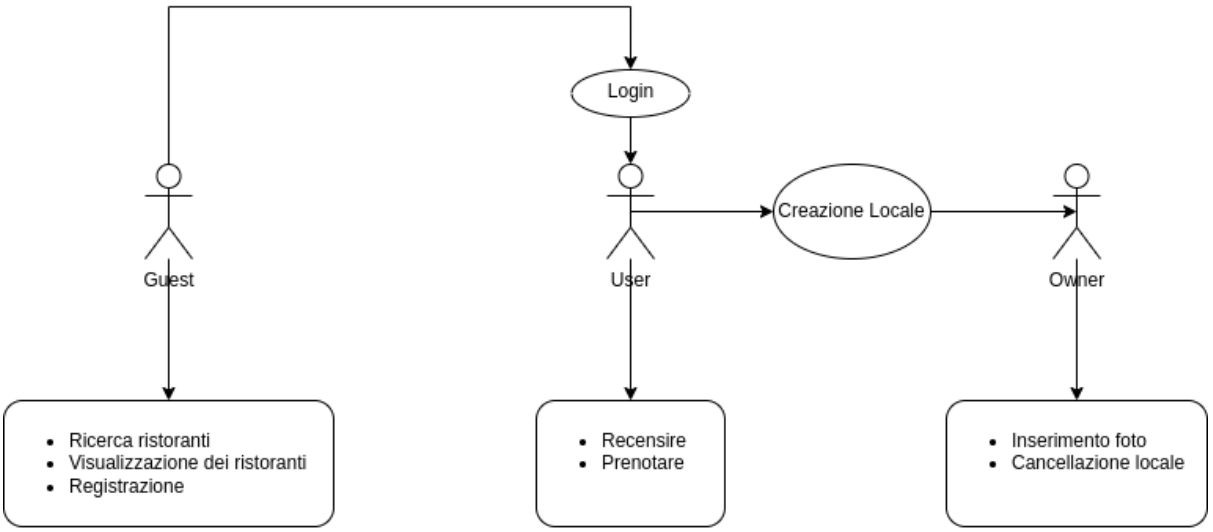
può navigare sul sito, prenotare in altri ristoranti e partecipare alla comunità di TableTales come qualsiasi altro membro registrato.

2 Descrizione della traccia

2.1 Funzionalità e caratteristiche

La web application di TableTales è stata sviluppata con l'intento di offrire servizi sia agli utenti generali che ai proprietari di ristoranti.
Gli User possono sfruttare il sito per visualizzare i profili dei ristoranti, effettuare ricerche, scrivere e leggere recensioni, e procedere con la prenotazione dei tavoli. La funzionalità di prenotazione offre inoltre una coda di attesa, nella quale gli User possono prenotare per un posto non attualmente disponibile, nell'eventualità che un altro User disdica la propria prenotazione.

D'altro canto, la piattaforma offre agli Owner di ristoranti, l'opportunità di inserire e gestire i propri locali. Una volta inserito il ristorante, gli User potranno interagire con il profilo del locale attraverso le azioni precedentemente menzionate.



2.2 Database e modelli

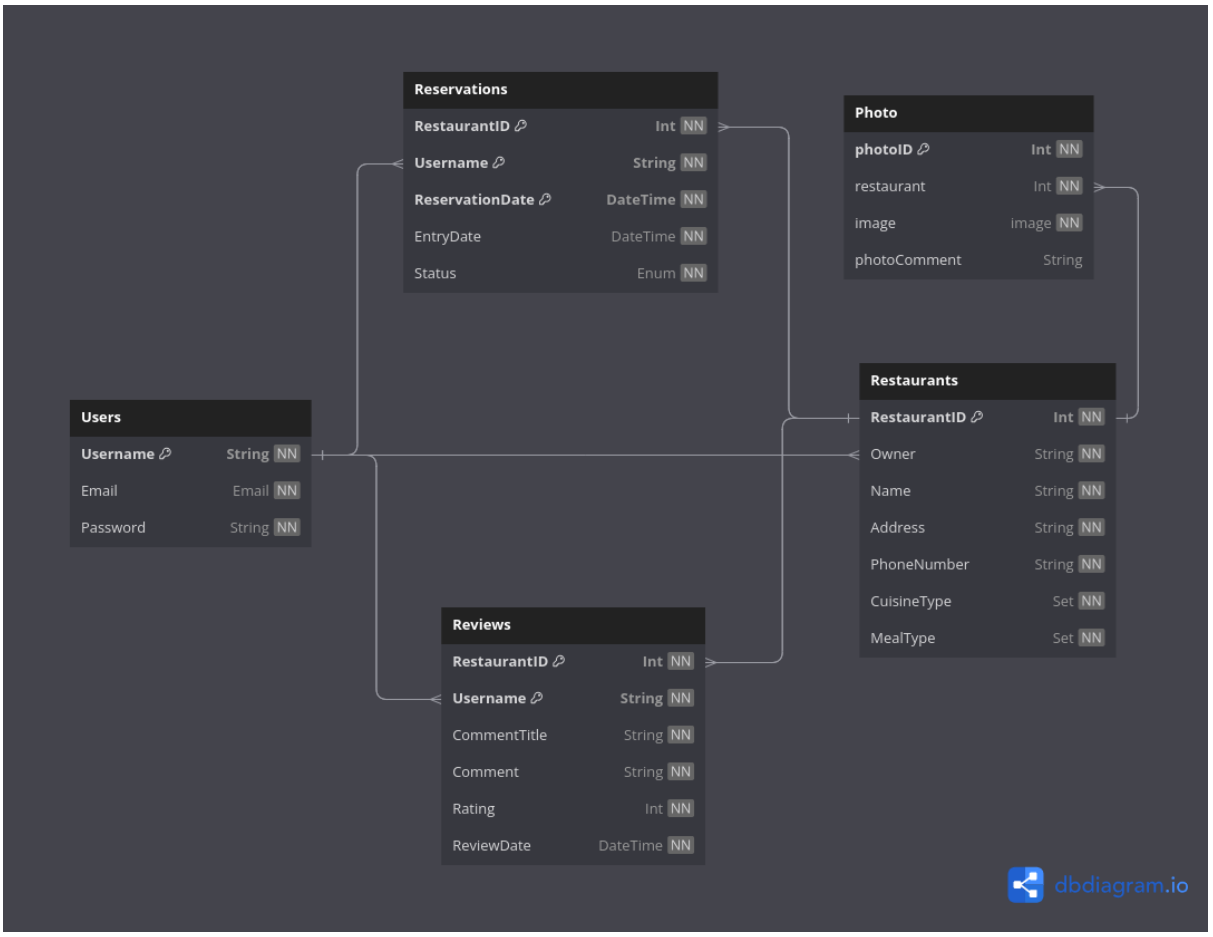
Nella fase iniziale di sviluppo del sito TableTales, è stata adottata come soluzione database il sistema SQLite. Questa decisione è stata motivata da diverse ragioni:

- Integrazione con Django: SQLite è la scelta predefinita per Django, il framework su cui è stato costruito il sito. Questa integrazione nativa facilita lo sviluppo iniziale, consentendo una rapida prototipazione e implementazione.

- Leggerezza e Portabilità: SQLite è un database leggero e serverless, il che lo rende ideale per progetti in fase di avvio, in particolare quando i dati e gli accessi al database sono limitati.

- Facilità di Utilizzo: Grazie alla sua semplicità strutturale, SQLite offre una curva di apprendimento agevole per gli sviluppatori, permettendo una gestione efficace dei dati senza la necessità di configurare server complessi.

Tuttavia, è fondamentale sottolineare che, mentre SQLite rappresenta una soluzione efficace per la fase iniziale di un progetto, potrebbe non essere la scelta più adatta man mano che il sito cresce e le esigenze di scalabilità e prestazioni diventano più pressanti.



2.2.1 Table Users:

Per questo modello è stato utilizzato quello presente all'interno di django Auth. non vengono utilizzate ulteriori informazioni

2.2.2 Table Restaurants:

Modello per rappresentare il ristorante, con tutte le sue informazioni annesse.

RestaurantID: Chiave primaria. Identificativo univoco per ogni ristorante. Non può essere nullo.
Owner: Rappresenta l'username del proprietario del ristorante. Si riferisce all'attributo "Username" della tabella Users.
Name: Nome del ristorante.
Address: Indirizzo fisico del ristorante.
PhoneNumber: Numero di telefono del ristorante.
CuisineType: Un insieme che specifica il tipo di cucina offerta dal ristorante (ad es. italiano, indiano).
MealType: Un insieme che indica i tipi di pasti serviti (ad es. colazione, pranzo, cena).

2.2.3 Table Reservations:

RestaurantID: Identificativo del ristorante presso cui è stata effettuata la prenotazione. Chiave esterna che si riferisce a Restaurants.RestaurantID.
Username: L'utente che ha effettuato la prenotazione. Chiave esterna che si riferisce a Users.Username.
ReservationDate: Data e ora in cui l'utente prevede di visitare il ristorante.
EntryDate: Data e ora della prenotazione.
Status: Uno stato enumerativo che indica lo stato della prenotazione (confermata, in attesa)

2.2.4 Table Reviews:

RestaurantID: Identificativo del ristorante che è stato recensito. Chiave esterna che si riferisce a Restaurants.RestaurantID.
Username: L'utente che ha scritto la recensione. Chiave esterna che si riferisce a Users.Username.
CommentTitle: Titolo della recensione.
Comment: Il corpo della recensione.
Rating: Una valutazione numerica (da 1 a 5) data al ristorante.
ReviewDate: Data in cui è stata scritta la recensione.

2.2.5 Table Photo:

photoID: Chiave primaria. Identificativo univoco per ogni foto caricata.
restaurant: Identificativo del ristorante associato alla foto. Chiave esterna che si riferisce a Restaurants.RestaurantID.
image: L'immagine o foto del ristorante.
photoComment: Commento o descrizione associata alla foto.

3 Tecnologie utilizzate

3.1 Backend

Django è il framework web scelto per la realizzazione del backend di TableTales, in seguito alle lezioni approfondite durante il corso di "Tecnologie Web". Django offre una serie di vantaggi distintive:

3.1.1 MVC e DRY

Django segue il pattern Model-View-Controller (MVC), anche se spesso viene definito come Model-View-Template (MVT) nel contesto Django. Questo pattern promuove una separazione delle preoccupazioni, rendendo il codice più organizzato e facilmente manutenibile. Inoltre, Django è fortemente basato sul principio "Don't Repeat Yourself" (DRY), assicurando che ogni pezzo di informazione abbia una singola rappresentazione chiara.

3.1.2 ORM Integrato

L'Object-Relational Mapping (ORM) di Django facilita le operazioni con il database, permettendo agli sviluppatori di interagire con il database usando il linguaggio Python, senza la necessità di scrivere query SQL.

3.1.3 Estendibilità:

Grazie ai "Django Apps", è possibile modularizzare il progetto in componenti riutilizzabili. Questo non solo facilita lo sviluppo e la manutenzione, ma permette anche di estendere la piattaforma con nuove funzionalità in modo semplice e organizzato.

3.1.4 Comunità e Documentazione

Django vanta una comunità di sviluppatori vasta e attiva. Questo garantisce una ricca base di risorse, tutorial e documentazione ufficiale, risultando in un supporto inestimabile durante lo sviluppo.

3.2 Protocollo di comunicazione

Il protocollo di comunicazione utilizzato per far comunicare backend e frontend è il protocollo http in quanto in fase di implementazione è stato sicuramente più semplice sviluppare le richieste in questa modalità.
Django, essendo un framework web, è intrinsecamente progettato per operare su HTTP, rendendo la gestione delle richieste HTTP semplice e diretta. Attraverso la sua architettura MVT, Django gestisce efficientemente le richieste HTTP, traducendole in operazioni sul database, elaborazioni o rendering di template.

3.3 Frontend

Per quanto riguarda la parte front-end di TableTales, la decisione è ricaduta su Bootstrap 5, un noto framework CSS. La scelta di Bootstrap è data da:

3.3.1 Responsive Design

Uno dei principali vantaggi di Bootstrap è la sua capacità intrinseca di fornire un design reattivo. Con l'ampio utilizzo di dispositivi mobili per accedere ai siti web, è essenziale che la piattaforma sia fruibile e visivamente accattivante su tutti i dispositivi, da desktop a smartphone. Bootstrap rende questa transizione fluida con le sue griglie flessibili e classi predefinite.

3.3.2 Componenti Pre-costruiti:

Bootstrap offre una vasta gamma di componenti pre-costruiti come modali, popover, tooltip, caroselli e molto altro. Questi componenti accelerano notevolmente lo sviluppo, riducendo il tempo e lo sforzo necessari per creare questi elementi da zero.

3.3.3 Integrazione

Integrare Bootstrap con Django è molto semplice. Questo consente agli sviluppatori di sfruttare al meglio le funzionalità di entrambi gli strumenti, rendendo lo sviluppo sia efficiente che efficace.

4 Test svolti

4.1 restaurantApp

Vengono esaminate le classi di test RestaurantModelTests e PhotoModelTests per assicurarsi che le funzionalità delle entità Restaurant e Photo dell'applicazione Django funzionino come previsto.

4.1.1 RestaurantModelTests

Questa classe si occupa dei test riguardanti il modello Restaurant:

4.1.1.1 setUp

Questa funzione viene eseguita prima di ogni metodo di test. Viene utilizzata per impostare un ambiente di test iniziale:
Viene creato un utente di test con il nome utente “testuser” e la password “testpassword”.
Viene creato un ristorante di test con dettagli come nome, indirizzo, tipo di cucina e tipo di pasto.

4.1.1.2 test_restaurant_creation

Testa se il ristorante viene creato e salvato correttamente nel database:

Verifica che il nome del ristorante sia "Test Restaurant".
Verifica che l'indirizzo sia "123 Test Street".

4.1.1.3 test_average_rating_update

Testa il metodo average_rating_update del modello Restaurant: Crea due recensioni per il ristorante di test, con valutazioni 3 e 5.
Verifica se il metodo restituisce una media corretta delle valutazioni, in questo caso 4.

4.1.1.4 test_meal_type_as_list:

Verifica se il metodo converte correttamente la stringa meal_type in una lista.

4.1.1.5 test_cuisine_type_as_list

Testa il metodo cuisine_type_as_list:

Verifica se il metodo converte correttamente la stringa cuisine_type in una lista. Qui, dovrebbe restituire una lista con 'Italiana' e 'Giapponese'.

4.1.2 PhotoModelTests

Questa classe si occupa dei test riguardanti il modello Photo:

4.1.2.1 setUp

Questa funzione, come nella classe precedente, viene eseguita prima di ogni metodo di test. Imposta un ambiente di test simile:

Crea un utente e un ristorante di test.
Crea una foto di test associata al ristorante di test con un commento "Test Comment".

4.1.2.2 test_photo_creation

Testa se la foto viene creata e salvata correttamente:

Verifica che il commento sulla foto sia "Test Comment".
Verifica che il nome del ristorante associato alla foto sia "Test Restaurant".

4.1.3 RestaurantViewsTest

Questa classe si occupa dei test relativi alle viste associate al modello Restaurant:

4.1.3.1 setUp

Prima di ogni test:

Crea un utente con il nome utente testuser e la password testpass123.
Crea un ristorante di test associato all'utente creato.
Inizializza un client di test Django.

4.1.3.2 test_restaurant_create_view

Testa la vista di creazione di un ristorante:

Esegue il login con l'utente di test.
Accede alla vista di creazione di un ristorante e verifica che la pagina venga visualizzata correttamente.
Invia una richiesta POST con dati di un nuovo ristorante e verifica che il ristorante venga creato e visualizzato correttamente.

4.1.3.3 test_restaurant_owner_list_view:

Testa la vista che mostra la lista dei ristoranti di un proprietario:

Esegue il login con l'utente di test.
Accede alla vista e verifica che mostri la lista dei ristoranti associati all'utente di test.

4.1.3.4 test_restaurant_list_view:

Testa la vista che mostra la lista di tutti i ristoranti:

Accede alla vista e verifica che mostri il ristorante di test creato nel metodo setUp.

4.1.3.5 test_restaurant_detail_view:

Testa la vista di dettaglio di un ristorante:

Accede alla pagina di dettaglio del ristorante di test e verifica che mostri correttamente le informazioni del ristorante.

4.1.3.6 test_restaurant_delete_view:

Testa la vista di eliminazione di un ristorante:

Esegue il login con l'utente di test.
Invia una richiesta POST per eliminare il ristorante di test.
Verifica che il ristorante venga eliminato e non venga più mostrato.

4.1.3.7 test_restaurant_update_view:

Testa la vista di aggiornamento di un ristorante:

Esegue il login con l'utente di test.
Accede alla vista di aggiornamento del ristorante di test e verifica che mostri correttamente il modulo di modifica.

4.1.3.8 test_photo_create_view:

Testa la vista di creazione di una foto per un ristorante:

Esegue il login con l'utente di test.
Accede alla vista di creazione di una foto per il ristorante di test e verifica che mostri correttamente il modulo di inserimento foto.

4.2 reservationApp

Questa documentazione descrive i test cases relativi al modello Reservation all'interno dell'applicazione Django. I test cases assicurano che il modello Reservation funzioni come previsto.

4.2.1 ReservationModelTest

Questa classe gestisce i test relativi al modello Reservation.

4.2.1.1 setUp:

Prima di ogni test:

Crea un utente con il nome utente testuser e la password 12345.
Crea un ristorante di test associato a questo utente.
Crea una prenotazione per il ristorante di test per l'utente, impostata per il giorno successivo all'ora attuale.

4.2.1.2 test_reservation_creation:

Testa la creazione di una prenotazione:

Verifica che il ristorante associato alla prenotazione sia il ristorante di test.
Verifica che l'utente associato alla prenotazione sia l'utente di test.
Verifica che lo stato della prenotazione sia 'pending'.
Verifica che il numero di persone per la prenotazione sia 3.

4.2.1.2 test_reservation_str:

Testa la rappresentazione in stringa del modello Reservation:

Crea una stringa di rappresentazione attesa per la prenotazione di test.

Verifica che la stringa restituita dal metodo __str__ del modello Reservation corrisponda alla stringa attesa.

4.2.1.3 test_reservation_status_choices:

Testa le scelte disponibili per lo stato (status) della prenotazione:

Verifica che le opzioni 'confirmed' (Confermata), 'pending' (In attesa) siano presenti tra le scelte di stato per una prenotazione.

4.2.2 TestReservationViews

Questa classe gestisce i test relativi alle ViewsReservation.

4.2.2.1 setUp

Prima di ogni test:

Crea un'istanza Client per effettuare richieste.
Crea un utente di test con nome utente testuser e password testpassword.
Crea un ristorante di test associato all'utente di test.

4.2.2.2 test_user_reservations_view

Questo test verifica la vista che mostra le prenotazioni dell'utente:

Effettua il login con l'utente di test.
Crea una prenotazione per l'utente di test nel ristorante di test.
Recupera la vista delle prenotazioni dell'utente.
Verifica che la risposta abbia codice di stato 200 (OK).
Verifica che nella risposta ci sia una lista di prenotazioni (in questo caso l'elenco dovrebbe contenere solo una prenotazione).

4.2.2.3 test_reservation_delete_view

Questo test verifica la vista che permette di cancellare una prenotazione:

Effettua il login con l'utente di test.
Crea una prenotazione per l'utente di test nel ristorante di test.
Effettua una richiesta POST per cancellare la prenotazione.
Verifica che la risposta abbia un codice di stato 302 (Redirect).
Verifica che la prenotazione non esista più nel database.

4.3 reviewsApp

Questa documentazione descrive i test cases relativi al modello Reviews all'interno dell'applicazione Django. I test cases assicurano che il modello Reviews funzioni come previsto.

4.3.1 ReviewModelTestCase

4.3.1.1 setUp

Questo metodo viene eseguito prima di ogni singolo test. Si occupa di impostare un ambiente di test per i metodi che seguiranno.

Crea un utente chiamato "testuser".
Crea un ristorante associato a quell'utente.
Crea una recensione per quel ristorante da parte di quell'utente.

4.3.1.2 test_review_creation

Verifica la creazione di una recensione:

Controlla se il numero totale delle recensioni nel database è 1.
Controlla se il titolo della recensione coincide con "Amazing!".
test_str_representation
Verifica la rappresentazione sotto forma di stringa di una recensione:

Si assicura che la rappresentazione come stringa della recensione sia correttamente formattata mostrando il nome del ristorante e lo username dell'utente che ha scritto la recensione.

4.3.1.3 test_str_for_logged_user

Testa la funzione che fornisce una rappresentazione formattata della recensione per gli utenti loggati:

Verifica che la data della recensione sia presente nella stringa.
Si assicura che la stringa contenga il valore della valutazione (5 in questo caso).
Controlla che il commento ("The food was delicious.") sia incluso nella stringa.

4.3.1.4 test_rating_validators

Verifica la validazione del campo di valutazione:

Crea una recensione con una valutazione inferiore a 1 e verifica se solleva un'eccezione di validazione (ValidationError).
Crea una recensione con una valutazione superiore a 5 e verifica nuovamente se viene sollevata un'eccezione di validazione.
Questo test assicura che le recensioni non possano avere valutazioni al di fuori dell'intervallo da 1 a 5.

4.3.2 ReviewViewsTest

4.3.2.1 setUp

Questo metodo viene eseguito prima di ogni singolo test per impostare un ambiente di test.

Crea un utente (testuser).
Crea un ristorante associato a testuser.
Crea una recensione per quel ristorante da parte di testuser.
Inizializza un client di test. Questo sarà utilizzato per effettuare richieste HTTP alle viste.

4.3.2.2 test_review_create_view

Questo test controlla la vista per la creazione di una nuova recensione.

Esegue il login come testuser.
Effettua una richiesta GET alla vista review_create per il ristorante creato in setUp.
Si assicura che la risposta abbia uno stato HTTP di 200 (successo) e che contenga la stringa 'Lascia una recensione'.
Successivamente, prepara un set di dati (post_data) per essere inviato alla vista come una richiesta POST per creare una nuova recensione.
Effettua una richiesta POST alla vista review_create con post_data.
Controlla che la risposta abbia uno stato HTTP di 200 e che contenga la stringa 'Excellent!', che è il commento della nuova recensione creata.

4.3.2.3 test_user_reviews_list_view

Questo test verifica la vista che elenca tutte le recensioni create da un utente specifico.

Esegue il login come testuser.
Effettua una richiesta GET alla vista user_reviews.
Si assicura che la risposta abbia uno stato HTTP di 200.
Controlla che la risposta contenga la stringa 'Good food!', che è il commento della recensione creata nel metodo setUp.

5 Difficoltà

Iniziazione a Django: Avvicinarmi a Django per la prima volta è stata una sfida. Nonostante la mia familiarità con il linguaggio Python, Django, essendo un framework complesso e vasto, ha richiesto tempo per essere padroneggiato. Tuttavia, ho potuto sfruttare efficacemente la mia conoscenza pregressa di Python e mi sono appoggiato alla documentazione ufficiale e ad altre risorse online per superare gli ostacoli.

Progettazione Grafica: Anche se la progettazione grafica non era l'elemento centrale del corso, ho ritenuto essenziale presentare un'interfaccia utente attraente e funzionale. Questo ha rappresentato una sfida aggiuntiva, poiché richiedeva competenze non strettamente legate al backend o alla logica dell'applicazione. In questo contesto, ho trovato in Bootstrap

un valido alleato, consentendomi di realizzare design responsive e accattivanti con un investimento di tempo ottimizzato.