UNIVERSITÀ TELEMATICA PEGASO

CORSO DI LAUREA TRIENNALE

IN INFORMATICA PER LE AZIENDE DIGITALI L-31

“MongoDB come Database di Backend per una Web App di Prenotazione Alberghiera: Progettazione e Implementazione"

RELATORE:

Filippo Sciarrone

CANDIDATO:

Mauro Garofalo

Matricola n.0312200838

Anno Accademico

2024-2025

Sommario

Indice delle figure

1. INTRODUZIONE

La ricerca condotta si propone di esplorare l'efficacia di MongoDB come database di backend per una Web App di prenotazione alberghiera. Nel contesto dell'industria alberghiera, l'efficienza nella gestione delle prenotazioni è fondamentale per garantire un'esperienza positiva agli ospiti e massimizzare il rendimento degli hotel.

Il presente lavoro si articola in diverse fasi. Nella Revisione della Letteratura vengono esaminate le basi della gestione delle prenotazioni alberghiere, l'introduzione ai database NoSQL e un'analisi dettagliata delle caratteristiche di MongoDB. I Requisiti del Sistema definiscono le necessità funzionali e non funzionali della Web App, mentre la Progettazione del Database delinea lo schema dei dati per la gestione delle prenotazioni utilizzando MongoDB.

L'Architettura della Web App descrive la struttura complessiva del sistema, con un focus sul ruolo di MongoDB nel backend e nell'interazione con il frontend. L'Implementazione della Web App illustra lo sviluppo pratico del frontend e del backend, con particolare attenzione all'integrazione di MongoDB e alla gestione delle operazioni CRUD.

I Test e la Valutazione valutano le prestazioni del sistema e la scalabilità, mentre la Discussione dei Risultati analizza e interpreta i risultati ottenuti, confrontando MongoDB con altre tecnologie e discutendo delle implicazioni pratiche e teoriche.

Infine, le Conclusioni e Sviluppi Futuri riassumono i principali risultati, le implicazioni e suggeriscono possibili direzioni future per la ricerca. In sintesi, questo studio offre una panoramica dettagliata dell'utilizzo di MongoDB come database di backend per una Web App

di prenotazione alberghiera, con importanti contributi per il settore alberghiero e per lo sviluppo di applicazioni web avanzate.

1.1. Contesto Alberghiero e Problematiche per la Raccolta Dati

Il settore alberghiero rappresenta un pilastro fondamentale dell'industria del turismo, contribuendo in modo significativo all'economia globale e offrendo una vasta gamma di servizi e strutture destinate ad ospitare viaggiatori provenienti da tutto il mondo. La gestione efficace di un'attività alberghiera richiede non solo un'attenzione particolare all'esperienza degli ospiti, ma anche la capacità di raccogliere, analizzare e interpretare una vasta gamma di dati per ottimizzare le operazioni, migliorare l'efficienza e garantire la soddisfazione del cliente.

1.2. Contesto Alberghiero

Le strutture alberghiere, che vanno dalle piccole pensioni ai grandi complessi turistici, si distinguono per la loro complessità e diversità di operazioni. Ogni struttura è caratterizzata da una serie di interazioni e transazioni che coinvolgono ospiti, dipendenti, fornitori e altri stakeholder, creando una rete intricata di processi e flussi di lavoro. L'obiettivo comune di ogni struttura alberghiera è quello di fornire un'esperienza memorabile e piacevole agli ospiti, garantendo nel contempo un'efficace gestione delle risorse e delle operazioni interne.

1.3. Problematiche per la Raccolta Dati

La raccolta dei dati nel contesto alberghiero può presentare diverse sfide e problematiche, che includono:

1.4. Diversità dei Dati

Le strutture alberghiere generano una vasta gamma di dati provenienti da molteplici fonti, tra cui prenotazioni, transazioni finanziarie, feedback degli ospiti, dati operativi e altro ancora. Questa diversità di dati richiede l'integrazione e l'analisi efficace di informazioni provenienti da sistemi e processi eterogenei.

1.5. Privacy e Sicurezza

I dati raccolti dalle strutture alberghiere possono contenere informazioni sensibili sugli ospiti, come dati personali, informazioni finanziarie e preferenze. È essenziale garantire la sicurezza e la riservatezza di tali dati per conformarsi alle normative sulla privacy e preservare la fiducia degli ospiti.

1.6. Integrazione dei Sistemi

Le strutture alberghiere utilizzano una varietà di sistemi e tecnologie per gestire le loro operazioni, tra cui sistemi di gestione degli ospiti (PMS), sistemi di prenotazione online, sistemi di gestione delle entrate (RMS) e altro ancora. L'integrazione efficace di questi sistemi è cruciale per garantire un flusso efficiente di dati e informazioni tra le diverse piattaforme.

1.7. Analisi dei Dati

Una volta raccolti i dati, è fondamentale analizzarli in modo efficace per estrarre insight utili per migliorare le operazioni alberghiere, ottimizzare i ricavi e migliorare l'esperienza degli ospiti. L'analisi dei dati può essere complessa a causa della vasta quantità di informazioni disponibili e della necessità di interpretare correttamente i dati per prendere decisioni informate.

1.8. Riassunto

In conclusione, il settore alberghiero è caratterizzato da una serie di sfide uniche per la raccolta e l'analisi dei dati. Affrontare queste problematiche richiede soluzioni innovative, tecnologie all'avanguardia e una profonda comprensione delle esigenze specifiche del settore alberghiero. Nella mia tesi, esplorerò in dettaglio le sfide e le soluzioni per la raccolta e l'analisi dei dati nel contesto alberghiero, con un focus particolare sulla mia esperienza pratica nell'implementazione di sistemi di gestione dei dati in una struttura alberghiera.

2. Obiettivi della tesi:

- Analizzare approfonditamente le caratteristiche di MongoDB come database NoSQL e confrontarle con i requisiti specifici del sistema di gestione delle prenotazioni alberghiere.
- Definire i requisiti funzionali e non funzionali della Web App di prenotazione alberghiera, identificando le esigenze degli utenti e degli stakeholder.
- Progettare uno schema dei dati ottimizzato per la gestione delle prenotazioni alberghiere utilizzando MongoDB, garantendo prestazioni elevate e scalabilità del sistema.
- Implementare una Web App di prenotazione alberghiera utilizzando MongoDB come database di backend, sviluppando sia il frontend che il backend con un'attenzione particolare alla sicurezza e all'efficienza delle operazioni.
- Condurre test approfonditi per valutare le prestazioni del sistema, inclusa la scalabilità e la gestione dei carichi di lavoro.
- Analizzare e interpretare i risultati dei test, confrontando MongoDB con altre tecnologie e discutendo delle implicazioni pratiche e teoriche dell'utilizzo di MongoDB come database di backend per la gestione delle prenotazioni alberghiere.
- Trarre conclusioni significative sull'efficacia di MongoDB nel contesto delle prenotazioni alberghiere e suggerire possibili sviluppi futuri per migliorare ulteriormente il sistema.
- Questi obiettivi guideranno lo sviluppo e la conduzione della ricerca per raggiungere lo scopo della tesi, offrendo importanti contributi teorici e pratici nel campo delle applicazioni web avanzate nel settore alberghiero.

3. Revisione della Letteratura

In questa sezione, esamineremo le principali ricerche, teorie e approcci metodologici nel campo della gestione delle prenotazioni alberghiere, dei database NoSQL e dell'utilizzo di MongoDB come database di backend per le Web App.

3.1. Concetti di base sulla gestione delle prenotazioni alberghiere

- **Prenotazioni:** Le prenotazioni rappresentano il processo attraverso il quale gli ospiti prenotano camere d'albergo per un determinato periodo di tempo. Questo può avvenire tramite diversi canali, come il sito web dell'hotel, le agenzie di viaggio online, le prenotazioni telefoniche o tramite walk-in. Le prenotazioni possono essere confermate o in sospeso, a seconda delle condizioni stabilite dall'hotel e della disponibilità delle camere.
- **Sistemi di Prenotazione:** Gli hotel utilizzano sistemi di prenotazione per gestire e registrare tutte le prenotazioni ricevute. Questi sistemi possono essere integrati con il sito web dell'hotel o essere software dedicati che consentono di monitorare le prenotazioni, gestire la disponibilità delle camere, stabilire tariffe e generare conferme di prenotazione.
- **Gestione delle Disponibilità:** La gestione delle disponibilità delle camere è un aspetto cruciale della gestione delle prenotazioni alberghiere. Gli hotel devono monitorare costantemente la disponibilità delle loro camere in base alle prenotazioni confermate, alle camere fuori servizio per manutenzione o altri motivi, e alle richieste degli ospiti. La gestione delle disponibilità mira a massimizzare l'occupazione delle camere e a garantire un'allocazione efficiente delle risorse.

- **Politiche di Cancellazione:** Le politiche di cancellazione stabiliscono le condizioni e i termini per la cancellazione di una prenotazione da parte degli ospiti. Queste politiche possono variare in base alla tipologia della prenotazione (ad esempio, tariffe non rimborsabili), alla stagionalità, agli eventi speciali e ad altri fattori. Le politiche di cancellazione influenzano la flessibilità e la gestione dei rischi dell'hotel.
- **Gestione delle Tariffe:** La gestione delle tariffe delle camere è un aspetto strategico della gestione delle prenotazioni alberghiere. Gli hotel devono stabilire e gestire le tariffe in base a vari fattori come la domanda e l'offerta, la stagionalità, gli eventi locali e le politiche dell'hotel. Questo include la definizione di tariffe standard, tariffe scontate, offerte speciali e pacchetti promozionali per attrarre gli ospiti e massimizzare il fatturato.

In sintesi, la gestione delle prenotazioni alberghiere coinvolge una serie complessa di processi e pratiche finalizzate a garantire una gestione efficiente delle prenotazioni, una massimizzazione dell'occupazione delle camere e una soddisfazione ottimale degli ospiti. La comprensione di questi concetti di base è essenziale per il successo operativo e commerciale di qualsiasi struttura ricettiva.

4. Panoramica sui database NoSQL e MongoDB

I database NoSQL rappresentano una categoria di sistemi di gestione dei dati progettati per affrontare le sfide di scalabilità, flessibilità dello schema e prestazioni che spesso si presentano con i database relazionali tradizionali. A differenza dei database relazionali, che utilizzano uno schema rigido e un linguaggio di interrogazione strutturato come SQL, i database NoSQL offrono un modello di dati più flessibile e scalabile, consentendo di memorizzare e gestire dati non strutturati o semi-strutturati in modo efficiente.

Tra i principali tipi di database NoSQL troviamo i database di documenti, i database di chiavi-valore, i database di colonne e i database di grafi. Ogni tipo di database NoSQL è ottimizzato per un tipo specifico di dati e per determinati casi d'uso, offrendo vantaggi unici in termini di prestazioni, scalabilità e flessibilità.

MongoDB è uno dei database NoSQL più popolari e ampiamente utilizzati ed è caratterizzato dal suo modello di dati orientato ai documenti. In MongoDB, i dati sono memorizzati in documenti JSON (JavaScript Object Notation), che sono strutture di dati flessibili e autosufficienti. I documenti sono organizzati in collezioni, che sono simili alle tabelle nei database relazionali, ma senza uno schema rigido.

MongoDB offre numerosi vantaggi rispetto ai database relazionali tradizionali, tra cui la flessibilità dello schema, la scalabilità orizzontale, l'alta disponibilità, le prestazioni elevate e la facilità d'uso. Grazie alla sua architettura distribuita e scalabile, MongoDB è in grado di gestire volumi di dati enormi e di supportare carichi di lavoro ad alta intensità di lettura e scrittura.

Inoltre, MongoDB offre una vasta gamma di funzionalità avanzate, tra cui l'indicizzazione, l'aggregazione, la replicazione, la sharding e la gestione transazionale. Queste funzionalità consentono agli sviluppatori di creare applicazioni scalabili, affidabili e ad alte prestazioni per

una varietà di casi d'uso, inclusi siti web ad alta intensità di traffico, applicazioni mobile, analisi dei dati e molto altro ancora.

In conclusione, MongoDB è un potente database NoSQL che offre numerosi vantaggi rispetto ai database relazionali tradizionali. Con il suo modello di dati flessibile, la scalabilità orizzontale e le funzionalità avanzate, MongoDB è diventato una scelta popolare per una vasta gamma di applicazioni, dalla gestione delle prenotazioni alberghiere alla gestione dei dati in tempo reale e all'analisi dei Big Data.

4.1. Approfondimento sull'utilizzo di MongoDB come database di backend per le Web App.

MongoDB è diventato una scelta sempre più diffusa come database di backend per le Web App, offrendo una serie di vantaggi che lo rendono particolarmente adatto per gestire grandi volumi di dati e carichi di lavoro ad alta intensità di lettura e scrittura. Esaminiamo più approfonditamente le caratteristiche e i vantaggi che rendono MongoDB una scelta ideale per le Web App, concentrandoci sul contesto del sistema di gestione delle prenotazioni alberghiere.

4.1.1. Flessibilità dello Schema: Una delle caratteristiche distintive di MongoDB è la sua flessibilità dello schema. A differenza dei database relazionali, che richiedono uno schema rigido e predefinito, MongoDB consente agli sviluppatori di memorizzare dati non strutturati o semi-strutturati senza la necessità di definire uno schema preciso in anticipo. Questo è particolarmente vantaggioso nelle Web App, dove i requisiti possono cambiare rapidamente e la struttura dei dati può essere complessa e variabile. Nel contesto delle prenotazioni alberghiere, dove possono essere necessarie informazioni aggiuntive o personalizzate per ogni prenotazione, la flessibilità dello schema di MongoDB consente di gestire facilmente questa complessità senza dover apportare modifiche allo schema esistente.

4.1.2. Scalabilità Orizzontale: MongoDB è progettato per scalare orizzontalmente su cluster di server, consentendo di gestire grandi volumi di dati e carichi di lavoro ad alta intensità di lettura

e scrittura. Questa capacità di scalabilità orizzontale è essenziale per le Web App che devono gestire un elevato numero di transazioni e utenti simultaneamente. Nel caso delle prenotazioni alberghiere, dove possono verificarsi picchi di traffico durante i periodi di alta stagione o eventi speciali, la scalabilità orizzontale di MongoDB consente di garantire una risposta rapida e affidabile alle richieste degli ospiti, garantendo al contempo un'esperienza senza interruzioni.

4.1.3. Query Potenti e Flessibili: MongoDB offre un potente sistema di query che consente di recuperare e manipolare i dati in modo efficiente e flessibile. Utilizzando il linguaggio di query MongoDB (noto come MongoDB Query Language o MQL), gli sviluppatori possono eseguire una vasta gamma di operazioni di interrogazione, filtraggio, aggregazione e ordinamento dei dati. Questo è particolarmente utile nel contesto delle prenotazioni alberghiere, dove possono essere necessarie query complesse per recuperare informazioni specifiche sulle prenotazioni, come disponibilità delle camere, dettagli degli ospiti e tariffe.

4.1.4. Indicizzazione Ottimizzata: MongoDB supporta un'ampia gamma di opzioni di indicizzazione per ottimizzare le prestazioni delle query, inclusi diversi tipi di indicizzazione come gli indici singoli, composti e geospaziali. Queste opzioni di indicizzazione consentono di migliorare le prestazioni delle query e ridurre i tempi di risposta, garantendo una maggiore efficienza nell'accesso ai dati. Nel contesto delle prenotazioni alberghiere, l'indicizzazione ottimizzata di MongoDB consente di eseguire rapidamente query su campi comuni come la data di arrivo, la data di partenza e il tipo di camera, garantendo una risposta rapida alle richieste degli ospiti.

4.1.5. Affidabilità e Disponibilità: MongoDB offre funzionalità avanzate di replicazione e ridondanza dei dati per garantire l'affidabilità e l'alta disponibilità del sistema. Utilizzando la replica dei dati e la distribuzione geografica dei dati su più nodi, MongoDB può garantire che i dati siano sempre accessibili e che non si verifichino perdite di dati in caso di guasti hardware o errori del sistema. Questa affidabilità e disponibilità sono fondamentali per le Web App che

gestiscono dati sensibili come le prenotazioni alberghiere, garantendo che le informazioni degli ospiti siano sempre protette e accessibili.

In conclusione, MongoDB offre una serie di vantaggi significativi come database di backend per le Web App, inclusa la flessibilità dello schema, la scalabilità orizzontale, le query potenti e flessibili, l'indicizzazione ottimizzata e l'affidabilità e la disponibilità del sistema. Nel contesto delle prenotazioni alberghiere, l'utilizzo di MongoDB come database di backend consente di gestire facilmente la complessità dei dati e garantire un'esperienza ottimale agli ospiti, fornendo al contempo una risposta rapida e affidabile alle richieste degli utenti.

5. Requisiti di sistema

5.1. Analisi dei Requisiti Funzionali e Non Funzionali della Web App di Prenotazione Alberghiera

L'analisi dei requisiti funzionali e non funzionali è un passaggio cruciale nel processo di sviluppo di una Web App di prenotazione alberghiera. In questa sezione, esamineremo in dettaglio i requisiti funzionali e non funzionali necessari per soddisfare le esigenze degli utenti e garantire il corretto funzionamento e le prestazioni ottimali della Web App.

5.2. Requisiti Funzionali

I requisiti funzionali descrivono le funzionalità specifiche che la Web App deve fornire per soddisfare le esigenze degli utenti. Questi requisiti delineano le azioni che gli utenti devono essere in grado di eseguire all'interno dell'applicazione. Alcuni esempi di requisiti funzionali per la Web App di prenotazione alberghiera includono:

5.2.1. Registrazione degli Utenti: Gli utenti devono essere in grado di creare un account all'interno dell'applicazione per accedere alle funzionalità di prenotazione e gestione delle prenotazioni.

5.2.2. Ricerca delle Camere: Gli utenti devono poter cercare camere disponibili in base a criteri come la data di check-in, la data di check-out, il numero di ospiti e le preferenze di camera.

5.2.3. Prenotazione delle Camere: Gli utenti devono poter prenotare una camera disponibile selezionando le date desiderate e inserendo le informazioni di pagamento.

5.2.4. Gestione delle Prenotazioni: Gli utenti devono poter visualizzare, modificare o cancellare le proprie prenotazioni esistenti.

5.2.5. Visualizzazione delle Informazioni sull'Hotel: Gli utenti devono poter visualizzare informazioni dettagliate sull'hotel, inclusi servizi offerti, foto delle camere e recensioni degli ospiti.

5.3. Requisiti Non Funzionali

I requisiti non funzionali descrivono gli aspetti qualitativi del sistema, come prestazioni, sicurezza e usabilità. Questi requisiti delineano le caratteristiche che il sistema deve possedere per garantire una buona esperienza agli utenti e garantire il corretto funzionamento dell'applicazione. Alcuni esempi di requisiti non funzionali per la Web App di prenotazione alberghiera includono:

***5.3.1. Prestazioni*:** La Web App deve essere in grado di gestire un elevato volume di traffico e fornire risposte rapide agli utenti, minimizzando i tempi di caricamento delle pagine e le latenze di rete.

5.3.2. Sicurezza: La Web App deve implementare misure di sicurezza per proteggere i dati sensibili degli utenti, inclusi dati personali e informazioni di pagamento, utilizzando tecniche di crittografia e autenticazione.

5.3.3. Usabilità: La Web App deve essere intuitiva e facile da usare per gli utenti, con un'interfaccia utente chiara e intuitiva che consente agli utenti di navigare facilmente attraverso le diverse funzionalità dell'applicazione.

5.3.4. Affidabilità: La Web App deve essere affidabile e disponibile, con un tempo di attività elevato e tempi di inattività minimi per garantire una buona esperienza agli utenti.

5.3.5. Scalabilità: La Web App deve essere in grado di scalare orizzontalmente per gestire un elevato volume di utenti e transazioni senza compromettere le prestazioni del sistema.

In sintesi, l'analisi dei requisiti funzionali e non funzionali della Web App di prenotazione alberghiera è essenziale per definire le specifiche e i criteri di successo per lo sviluppo e l'implementazione dell'applicazione. Questi requisiti forniscono una base solida per la

progettazione e lo sviluppo del sistema, garantendo che soddisfi le esigenze degli utenti e offra un'esperienza ottimale agli utenti.

5.4. Identificazione dei Requisiti Specifici per l'Utilizzo di MongoDB come Database di Backend

L'implementazione di MongoDB come database di backend per la Web App di prenotazione alberghiera richiede l'identificazione e la definizione dei requisiti specifici per garantire un'efficace integrazione e un funzionamento ottimale del sistema. In questa sezione, analizzeremo i requisiti specifici relativi all'utilizzo di MongoDB come database di backend per la Web App di prenotazione alberghiera.

5.5. Schema dei Documenti

MongoDB è un database orientato ai documenti, il che significa che i dati sono organizzati in documenti JSON (JavaScript Object Notation). Pertanto, uno dei requisiti principali è la definizione dello schema dei documenti, compresi i campi e i tipi di dati per rappresentare informazioni come prenotazioni, ospiti e dettagli delle camere.

Ad esempio, uno schema dei documenti per le prenotazioni alberghiere potrebbe includere campi come data di check-in, data di check-out, numero di ospiti e tipo di camera. È importante progettare uno schema dei documenti che sia flessibile e adatto alle esigenze specifiche della Web App di prenotazione alberghiera, consentendo la gestione efficace dei dati e facilitando le operazioni di query e manipolazione dei dati.

5.6. Indicizzazione dei Dati

Un altro requisito importante è l'indicizzazione dei dati per ottimizzare le prestazioni delle query. MongoDB supporta una vasta gamma di opzioni di indicizzazione, compresi gli indici singoli, composti e geospaziali. È importante identificare i campi chiave per l'indicizzazione, come ad esempio i campi utilizzati frequentemente nelle query di ricerca e filtraggio delle prenotazioni alberghiere, per garantire una rapida risposta alle richieste degli utenti.

Ad esempio, un campo chiave per l'indicizzazione potrebbe essere la data di check-in delle prenotazioni, che viene spesso utilizzata per filtrare le prenotazioni disponibili in base alla data di arrivo degli ospiti. Indicizzare questo campo migliorerà le prestazioni delle query di ricerca delle prenotazioni alberghiere e garantirà una risposta rapida agli utenti.

5.7. Gestione della Scalabilità

MongoDB è progettato per scalare orizzontalmente su cluster di server, consentendo di gestire grandi volumi di dati e carichi di lavoro ad alta intensità di lettura e scrittura. Pertanto, un requisito importante è la progettazione di un'architettura scalabile che consenta di aumentare dinamicamente la capacità di elaborazione e memorizzazione dei dati in risposta al carico di lavoro.

Ad esempio, è possibile implementare una configurazione a cluster MongoDB utilizzando la replica dei dati e la distribuzione geografica dei dati su più nodi per garantire l'affidabilità e l'alta disponibilità del sistema. Inoltre, è possibile utilizzare tecniche come la frammentazione dei dati per distribuire i dati su più nodi e garantire una distribuzione uniforme del carico di lavoro.

In sintesi, l'identificazione dei requisiti specifici per l'utilizzo di MongoDB come database di backend per la Web App di prenotazione alberghiera è essenziale per garantire un'efficace integrazione e un funzionamento ottimale del sistema. Questi requisiti includono la definizione dello schema dei documenti, l'indicizzazione dei dati e la progettazione di un'architettura scalabile per gestire grandi volumi di dati e carichi di lavoro ad alta intensità.

6. Progettazione del Database

6.1. Progettazione del Database

La progettazione del database è una fase fondamentale nello sviluppo di un'applicazione, in quanto influisce direttamente sulla struttura dei dati e sull'efficienza delle operazioni di archiviazione e recupero. In questo capitolo, verrà presentata la progettazione del database per il sistema di gestione delle prenotazioni di camere alberghiere.

6.2. Architettura del Database

Il database utilizzato per il sistema di gestione delle prenotazioni è basato su MongoDB, un database NoSQL flessibile e scalabile. MongoDB è stato scelto per la sua capacità di gestire grandi volumi di dati in modo efficiente e per la sua flessibilità nel modellare dati complessi.

6.3. Schema dei Dati

Il database è organizzato in diversi documenti, ognuno dei quali rappresenta una camera all'interno dell'albergo. Ogni documento contiene i seguenti campi:


```
_id: "65b95de460b59c81584ffff4"
numero_camera : "101"
name : "Camera Deluxe"
type : "Deluxe"
price : 200
description : "Una camera spaziosa con vista panoramica."
available : true
```


Figura 1 Esempio di documento all'interno della collezione.

Come mostrato in Figura **1** Il nostro documento dovra' rispettare determinati criteri come

Nel nostro caso:

- **_id**: un identificatore univoco generato automaticamente per ogni documento.
- **numero_camera**: il numero univoco della camera.
- **nome**: il nome della camera.
- **tipo**: il tipo di camera (es. standard, deluxe, suite).
- **prezzo**: il prezzo della camera per notte.
- **disponibile**: un flag booleano che indica se la camera è disponibile per la prenotazione.

6.4. Scelte di Progettazione per Ottimizzare le Prestazioni e la Scalabilità del Database

6.4.1 Utilizzo di Indici

Per migliorare le prestazioni delle query di ricerca, verranno utilizzati indici su campi frequentemente interrogati come **numero_camera** e **disponibile**. Gli indici consentono di ridurre il tempo di ricerca e di accelerare l'accesso ai dati.

6.4.2. Partizionamento dei Dati

Considerando la crescita potenziale del numero di camere nel database, verrà implementato il partizionamento dei dati per distribuire il carico su più nodi e migliorare la scalabilità orizzontale del sistema. Ciò garantirà prestazioni elevate anche con un grande volume di dati.

6.4.3 Ottimizzazione delle Query

Le query verranno ottimizzate per ridurre il tempo di esecuzione e l'utilizzo delle risorse del database. Verranno utilizzate tecniche come l'indicizzazione selettiva e l'ottimizzazione delle query per garantire una risposta rapida alle richieste degli utenti.

6.4.5. Monitoraggio delle Prestazioni

Verranno implementati strumenti di monitoraggio delle prestazioni per monitorare le prestazioni del database in tempo reale. Ciò consentirà di identificare eventuali problematiche e di apportare correzioni tempestive per garantire un funzionamento ottimale del sistema.

6.4.6. Backup e Ripristino dei Dati

Saranno implementati processi automatici di backup e ripristino dei dati per garantire l'integrità e la disponibilità dei dati in caso di guasto del sistema o di perdita di dati. I backup verranno archiviati in un luogo sicuro e aggiornati regolarmente per garantire la continuità operativa.

7. Architettura della Web App

La Web App di prenotazione alberghiera è progettata seguendo un'architettura moderna e scalabile che consente agli utenti di prenotare camere d'albergo in modo rapido e efficiente. L'architettura complessiva della Web App comprende diverse componenti interconnesse che collaborano per fornire un'esperienza utente fluida e intuitiva.

7.1 Descrizione dell'Architettura Complessiva

L'architettura della Web App si basa su un'architettura a microservizi, dove diverse funzionalità sono suddivise in servizi modulari e indipendenti. Nel nostro progetto stiamo utilizzando un'architettura RESTful per definire e gestire le nostre API. Le API RESTful consentono di creare servizi web che seguono i principi di REST (Representational State Transfer), che include l'uso di metodi HTTP standard come GET, POST, PUT e DELETE per operare sulle risorse. Le risorse sono identificate da URL univoci e manipolate tramite queste richieste HTTP. Questo approccio offre un'organizzazione chiara e prevedibile delle operazioni sulle risorse, facilitando lo sviluppo e il mantenimento delle applicazioni web.

Componenti Principali dell'Architettura:

7.1.2. Frontend: La parte visibile agli utenti, realizzata utilizzando **HTML** per la struttura del documento web. **CSS** per lo stile e la formatazione del documento HTML e JavaScript per la logica e la gestione degli eventi del documento HTML.

Tutto cio' fornisce un'interfaccia utente intuitiva e reattiva per la prenotazione delle camere d'albergo.

7.1.3. Backend: Il backend della Web App, implementato in Node.js con Express.js, gestisce la logica di business e la comunicazione con il database MongoDB. È responsabile di elaborare le richieste degli utenti, autenticare gli utenti e fornire dati dinamici al frontend.

7.1.4. Database MongoDB: MongoDB è utilizzato come database principale per memorizzare e gestire i dati relativi alle camere d'albergo, alle prenotazioni e agli utenti. Grazie al suo modello di dati flessibile e scalabile, MongoDB si integra perfettamente con l'architettura della Web App, consentendo una gestione efficiente dei dati.

7.1.5. API Gateway: Un API Gateway, implementato con Express.js, funge da punto di ingresso unificato per tutte le richieste provenienti dal frontend. Gestisce l'indirizzamento delle richieste ai servizi backend corretti e fornisce un'interfaccia standardizzata per l'interazione con il frontend.

7.2. Ruolo di MongoDB nell'Architettura

MongoDB svolge un ruolo fondamentale nell'architettura della Web App di prenotazione alberghiera. Come database principale, MongoDB è responsabile della memorizzazione e della gestione dei dati relativi alle camere d'albergo, alle prenotazioni e agli utenti. Grazie alla sua architettura orientata ai documenti, MongoDB consente di modellare i dati in modo flessibile e di eseguire operazioni complesse in modo efficiente.

7.3. Interazione con il Frontend e Altri Componenti del Sistema

Il frontend della Web App interagisce direttamente con MongoDB attraverso le API RESTful fornite dal backend. Quando un utente effettua una ricerca di camere d'albergo o invia una prenotazione, il frontend invia una richiesta al backend, che a sua volta interroga MongoDB per recuperare i dati richiesti. Una volta ottenuti i dati, il backend li elabora e li invia al frontend per essere visualizzati agli utenti.

Inoltre, MongoDB interagisce con altri componenti del sistema, come il backend e l'API Gateway, per fornire e gestire i dati in modo efficiente. Utilizzando query MongoDB ottimizzate e indici adeguati, è possibile garantire prestazioni elevate e tempi di risposta rapidi per le richieste degli utenti.

8. Implementazione della Web App e Struttura del Progetto:

8.1. Istallazione e popolazione.

Come accennato prima, per l'implementazione della nostra webapp, le tecnologie usate sono: Node.js ed Express per il nostro backend, Axios molto importante per la gestione delle nostre chiamate e React per quanto riguarda il nostro Frontend.Iniziamo dunque a mostrare passo passo come strutturare la nostra Webapp con esempi, passaggi ed immagini. Per prima cosa istalliamo il nostro Database noSQL MongoDB tramite il sito ufficiale:https://www.mongodb.com/try/download/compass ed assicuriamoci che sia compatibile con la nostra versione di Windows (nel mio caso W11 64 bit).

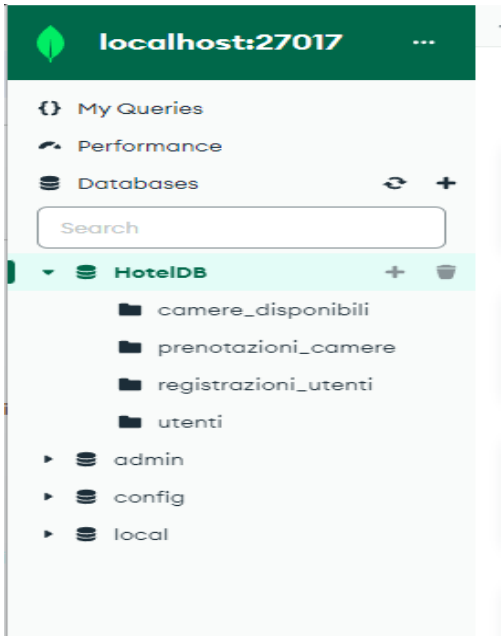


Figura 2- Esempio di creazione Database e relative Collezioni

Come mostrato in figura **2** , una volta Avviato il nostro MongoDB possiamo scegliere tramite l'opzione propostaci se operare in Locale o Remoto. Una volta avviato il nostro software, ci troveremo nella schermata principale dove creando il nostro database e assegnandogli un nome , ci chiederà' di inserire anche il nome della Collezione (molto importante in quanto andremo a inserire i nostri documenti in formato in formato JSON) .

8.2. Popolamento collezioni

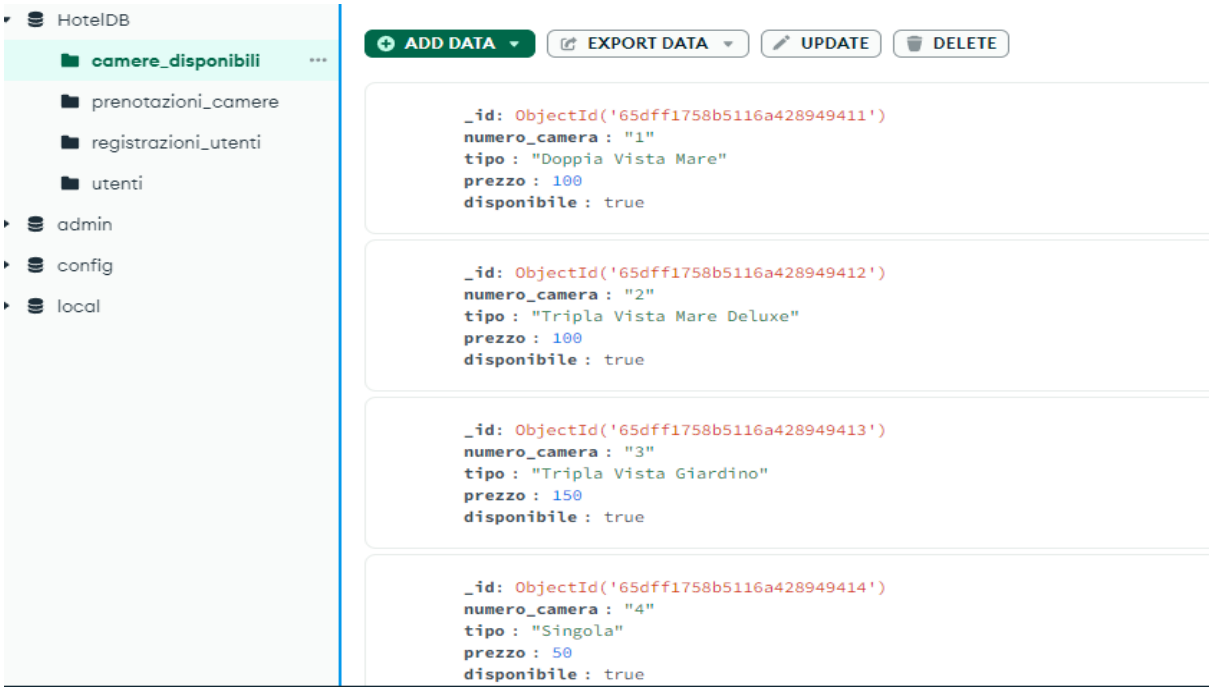


Figura 3- Esempio di popolamento collezione con inserimento dei documenti relativi alle camere, in formato JSON

Come mostrato in figura **3 ,** Abbiamo popolato la nostra collezione “camere_disponibili” con tutta la lista delle camere che l’utente tramite il frontend potra’ prenotare se dovessero essere disponibili. Il formato per ogni camera nel nostro caso è: id della camera che è generato automaticamente dal nostro DB, numero camera, nome, tipo, prezzo e disponibilita’ .

N.b. Abbiamo lasciato vuote le collezioni “prenotaizoni_camere” e “registrazioni utenti” , perché esse verranno popolate in futuro automaticamente come verra’ mostrato.

9. Sviluppo del Backend:

Adesso Dobbiamo creare il nostro ambiente Backend creando una cartella dal prompt dei comandi o manualmente sulla directory scelta.

Abbiamo creato una nuova cartella per il progetto e inizializzato un nuovo progetto Node.js utilizzando npm (Node Package Manager).

Abbiamo installato le dipendenze necessarie, inclusi Express.js per la gestione del server HTTP e Mongoose per la connessione e la gestione del database MongoDB.

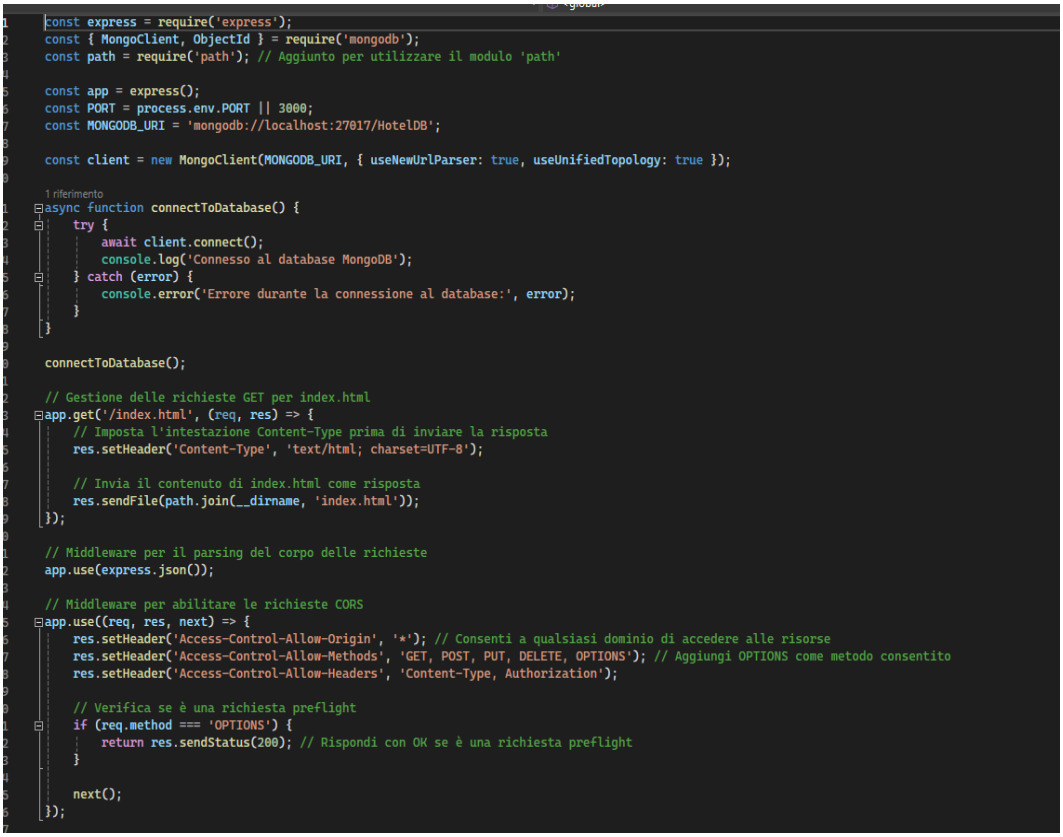
9.1. Definizione dei modelli di dati:

Abbiamo definito i modelli di dati utilizzando Mongoose per rappresentare le camere e le prenotazioni nel database MongoDB ovvero (**cameraSchema** e **prenotazioneSchema**) che rappresentano i dati che vogliamo memorizzare nel nostro database MongoDB.
Successivamente, abbiamo creato due modelli Camera e Prenotazione utilizzando mongoose.model, che ci consente di interagire con i dati del database tramite questi modelli.

Ogni modello è stato definito con un insieme di campi e vincoli di validazione tramite lo schema di Mongoose. Ad esempio, abbiamo specificato i tipi di dati, i requisiti di obbligatorietà (required: true) e altri vincoli come il prezzo della camera che deve essere un numero e la disponibilità della camera impostata su un valore predefinito di true.

Infine, abbiamo esportato i modelli in modo che possano essere utilizzati in altri file del nostro progetto, come nei file delle route o nei controller.

9.2. Configurazione del server e delle route API:



```
const express = require('express');
const { MongoClient, ObjectId } = require('mongodb');
const path = require('path'); // Aggiunto per utilizzare il modulo 'path'

const app = express();
const PORT = process.env.PORT || 3000;
const MONGODB_URI = 'mongodb://localhost:27017/HotelDB';

const client = new MongoClient(MONGODB_URI, { useNewUrlParser: true, useUnifiedTopology: true });

1 riferimento
async function connectToDatabase() {
    try {
        await client.connect();
        console.log('Connesso al database MongoDB');
    } catch (error) {
        console.error('Errore durante la connessione al database:', error);
    }
}

connectToDatabase();

// Gestione delle richieste GET per index.html
app.get('/index.html', (req, res) => {
    // Imposta l'intestazione Content-Type prima di inviare la risposta
    res.setHeader('Content-Type', 'text/html; charset=UTF-8');

    // Invia il contenuto di index.html come risposta
    res.sendFile(path.join(__dirname, 'index.html'));
});

// Middleware per il parsing del corpo delle richieste
app.use(express.json());

// Middleware per abilitare le richieste CORS
app.use((req, res, next) => {
    res.setHeader('Access-Control-Allow-Origin', '*'); // Consenti a qualsiasi dominio di accedere alle risorse
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS'); // Aggiungi OPTIONS come metodo consentito
    res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');

    // Verifica se è una richiesta preflight
    if (req.method === 'OPTIONS') {
        return res.sendStatus(200); // Rispondi con OK se è una richiesta preflight
    }

    next();
});
```

Figura 4– Esempio di implementazione del backend

Come mostrato nella figura **4**, abbiamo configurato un server Express.js per gestire le richieste HTTP provenienti dal frontend.

Abbiamo definito le route API per gestire le operazioni CRUD (Create, Read, Update, Delete) per le camere e le prenotazioni.

Abbiamo implementato i controller per gestire le operazioni CRUD per le camere e le prenotazioni.

I controller hanno incluso funzioni per creare nuove camere, recuperare camere disponibili, aggiornare le informazioni delle camere, eliminare camere, creare nuove prenotazioni, recuperare prenotazioni esistenti e altro ancora.

9.3. Integrazione con MongoDB:

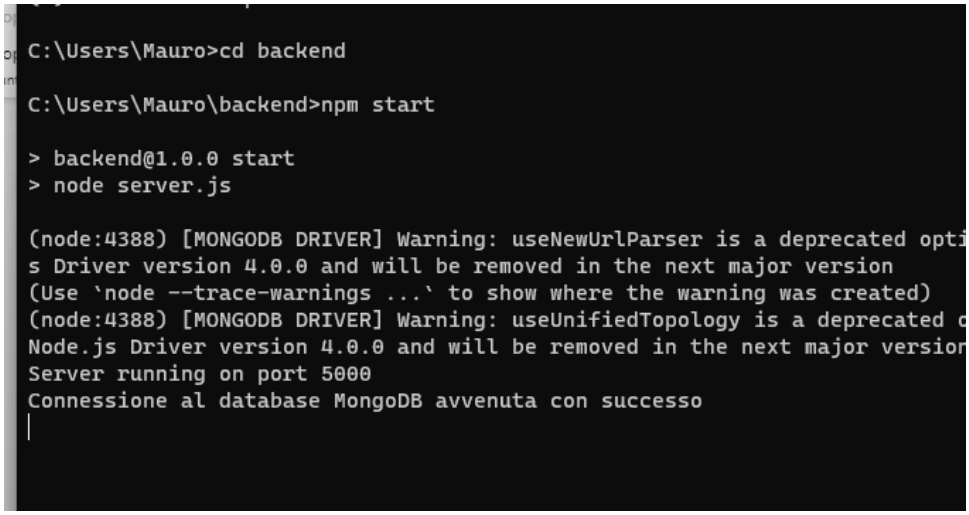


Figura 5– Dimostrazione dell'avvenuta connessione tra backend e Database.

Come mostrato in figura **5** abbiamo stabilito una connessione al database MongoDB utilizzando Mongoose, specificando l'URL del database e gestendo gli eventi di connessione e errore. Se non dovessimo riscontrare problemi vedremo come in figura la dicitura "Connessione al database MongoDB avvenuta con successo".

I modelli di dati definiti con Mongoose sono stati utilizzati per interagire con le collezioni MongoDB corrispondenti.

10. Sviluppo del Frontend:

10.1. Inizializzazione del progetto:

Abbiamo creato una nuova cartella per il frontend chiamata **pubblic e creato il nostro index.html con script.js e stile della pagina (style.css)**

Questo ci ha fornito una struttura di base per il frontend con cui lavorare, inclusi componenti predefiniti e una configurazione di sviluppo preimpostata.

10.1.1. Gestione dello stato:

Abbiamo utilizzato lo script.js, per gestire lo stato dinamico dei componenti, ad esempio per memorizzare le camere disponibili recuperate dal backend.

10.1.2. Chiamate API:

Abbiamo utilizzato Axios per effettuare chiamate API dal frontend al backend, ad esempio per recuperare le camere disponibili e per inviare nuove prenotazioni al server.

10.1.3. Rendering dinamico:

Abbiamo utilizzato JSX per definire la struttura HTML dei componenti e renderizzare dinamicamente i dati provenienti dal backend, ad esempio le camere disponibili e le informazioni sulla prenotazione.

10.1.4. Stile e presentazione:

Abbiamo utilizzato CSS per lo stile dei componenti e per la presentazione dell'interfaccia utente, garantendo una buona esperienza utente e un design coerente. Nel nostro caso abbastanza semplice in quanto è un ambiente in cui stiamo testando l’efficacia della web App.

11. Illustrazione della Procedura della nostra APP

11.1. Visualizzazione del frontend

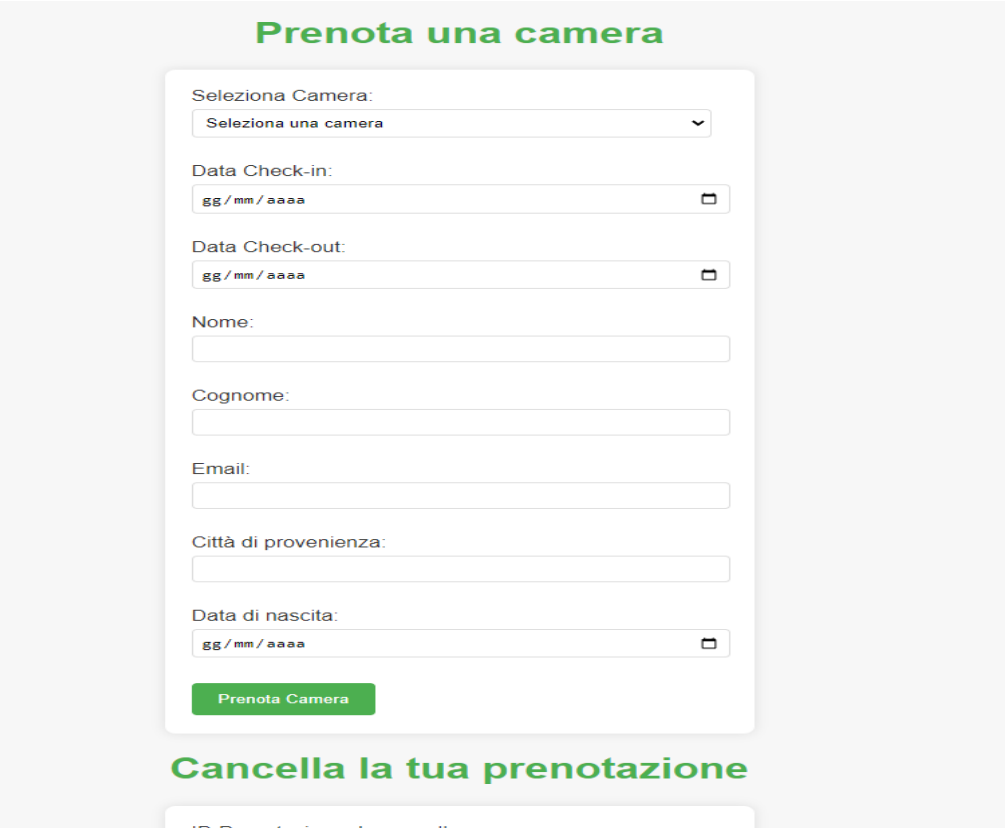


*Figura **6**– Parte della pagina / sezione Prenotazione camera con relativi campi da compilare*

Come evidenziato in figura **6** vediamo il nostro Frontend con una struttua HTML per la prenotazione o cancellazione camere con uno stile CSS creato da poter essere intuitivo e colorato.

Adesso possiamo vedere come selezionando una camera disponibile, e compilando i dati richiesti, Il nostro database si popolera, con tutte le informazioni disponibili che ci serviranno in futuro per interrogare il nostro database, con delle query che mostreremo in futuro.

Esempio: quante volte la camera 1 è stata prenotata in tale periodo ecc.

11.2 Selezione della camera disponibile

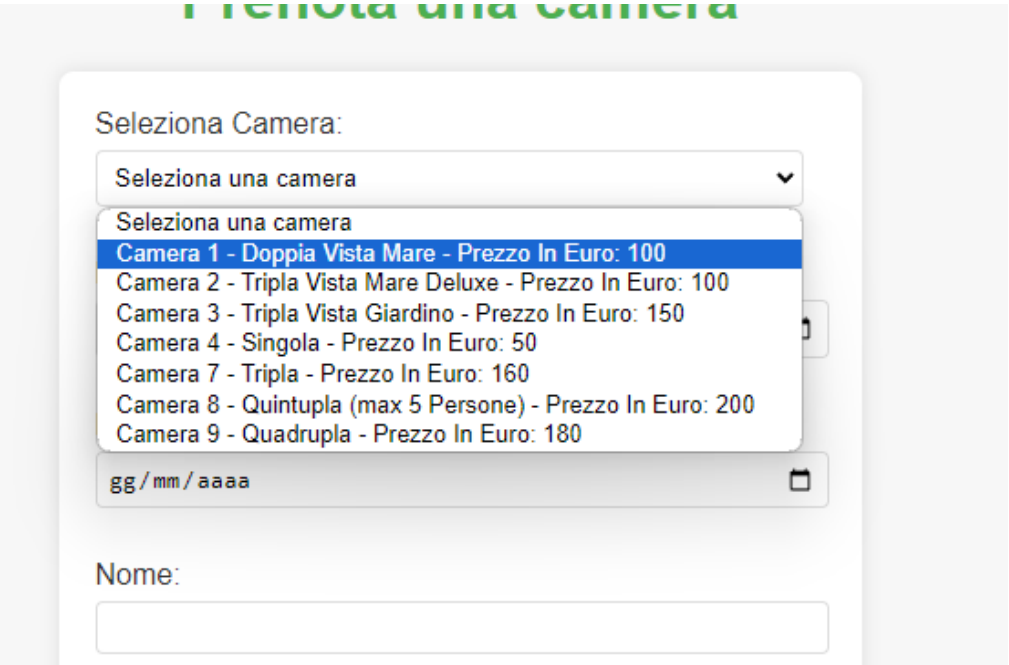


Figura 7– Collegamento del frontend al DB, che ci mostra le camere disponibili

Come Mostrato nella Figura **7** tramite chiamate inserite nel nostro frontend, possiamo visualizzare le camere disponibili nel menu' a tendina, selezionare quella desiderata compilare i campi richiesti e Prenotare.

11.3. Prenotazione effettuata.

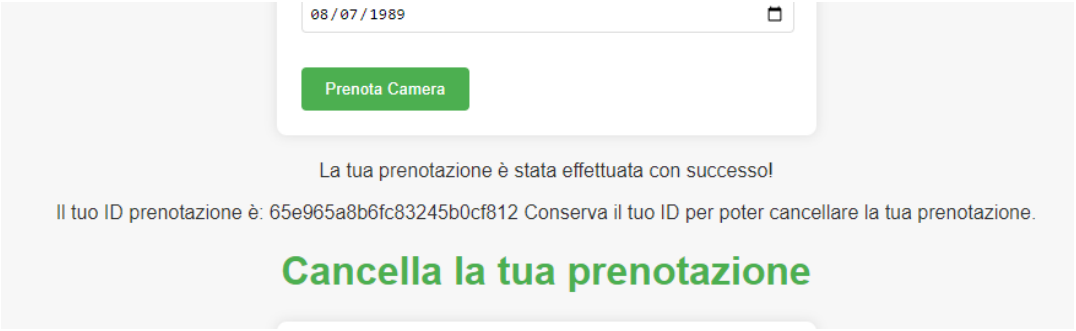


*Figura **8**– Prenotazione avvenuta con successo con generazione ID randomico per ogni prenotazione*

Come mostrato in Figura **8** Dopo aver cliccato su Prenota ecco che ci viene assegnato un ID univoco che potra' essere utilizzato per un eventuale cancellazione futura. Proprio per questo motivo nel messaggio viene visualizzato il testo "Conserva il tuo ID per poter cancellare la tua prenotazione".

11.4. Popolamento Automatico nel Database.

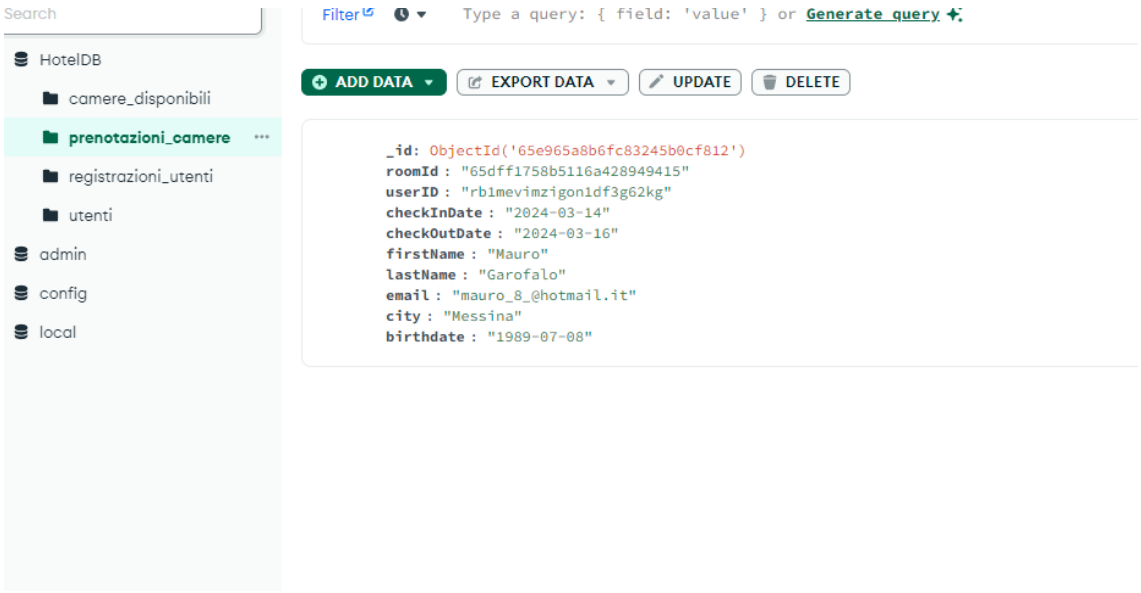


Figura ***9****– Popolazione automatica sulla collezione prenotazioni camere dopo la prenotazione dal Frontend da parte dell'utente.*

Come evidenziato in figura **9** vediamo che una volta prenotata la camera, il nostro database nella relativa collezione"prenotazioni_camere" si popolerà man mano l'utente finale prenoterà le camere e avendo compilato i campi come in precedenza mostrato visualizzeremo, le informazioni necessarie relative al cliente che ha prenotato.

11.5 Cancellazione Prenotazione

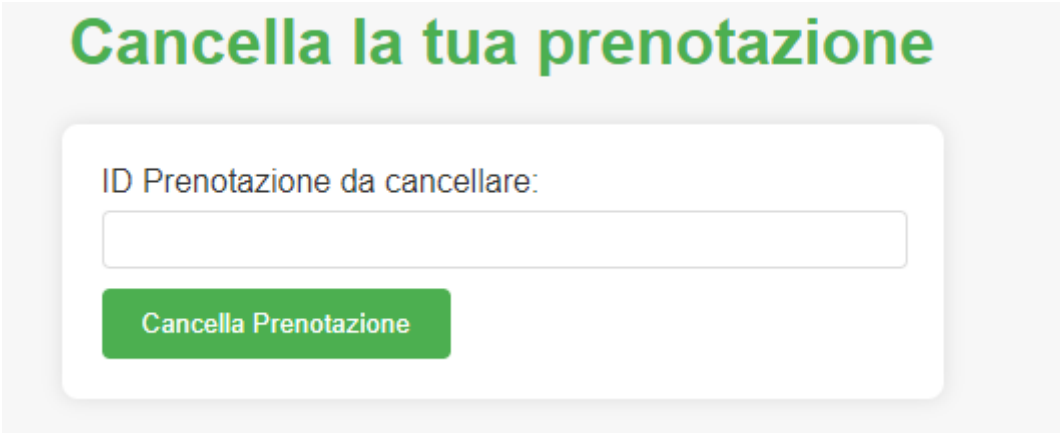


*Figura **10**– Campo su cui inserire l'id e inviare il comando Cancella Prenotazione.*

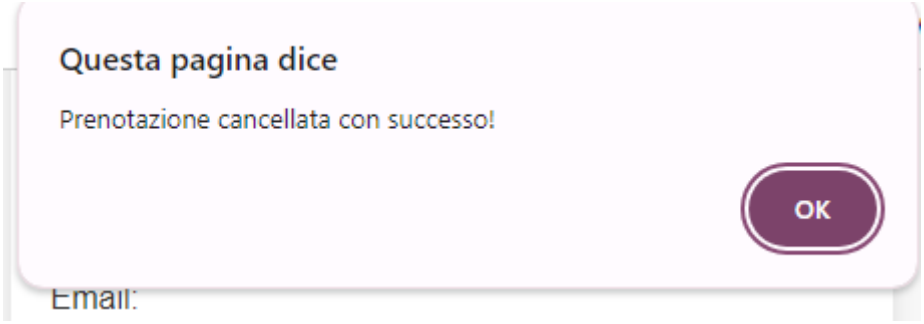


*Figura **11**– Pop up di avvenuta conferma di cancellazione della prenotazione.*

Nelle figure **10** e **11** ci viene mostrato cosa succede in caso di cancellazione, ovvero, l'utente avendo con se l'id assegnatogli, potra' inserirlo nel campo richiesto (figura **10)** e cancellare la prenotazione. Una volta cliccato su Cancella Prenotazione apparira' un messaggio d'allarme (figura **11**) che confermera' che la cancellazione è avvenuta con successo.

11.6. Cancellazione Automatica dalla collezione del database.

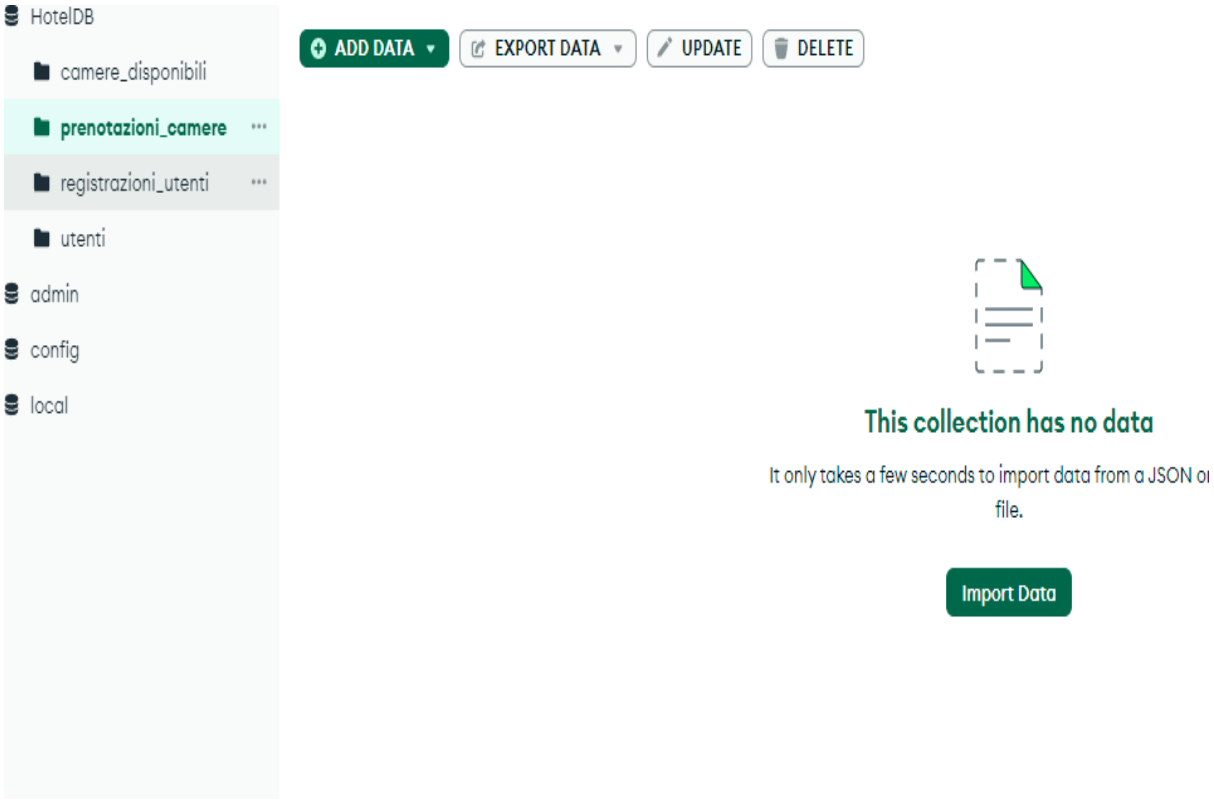


*Figura **12**– Cancellazione automatica dal database da parte dell'utente dal frontend.*

La figura **12** , ci mostra come una volta che l'utente abbia cancellato la prenotazione , nella nostra collezione del database automaticamente verranno cancellate le informazioni che avevamo visto in precedenza (figura **9**) cosi da poter rendere la camera nuovamente disponibile nel nostro portale.

11.7 Registrazione utenti.

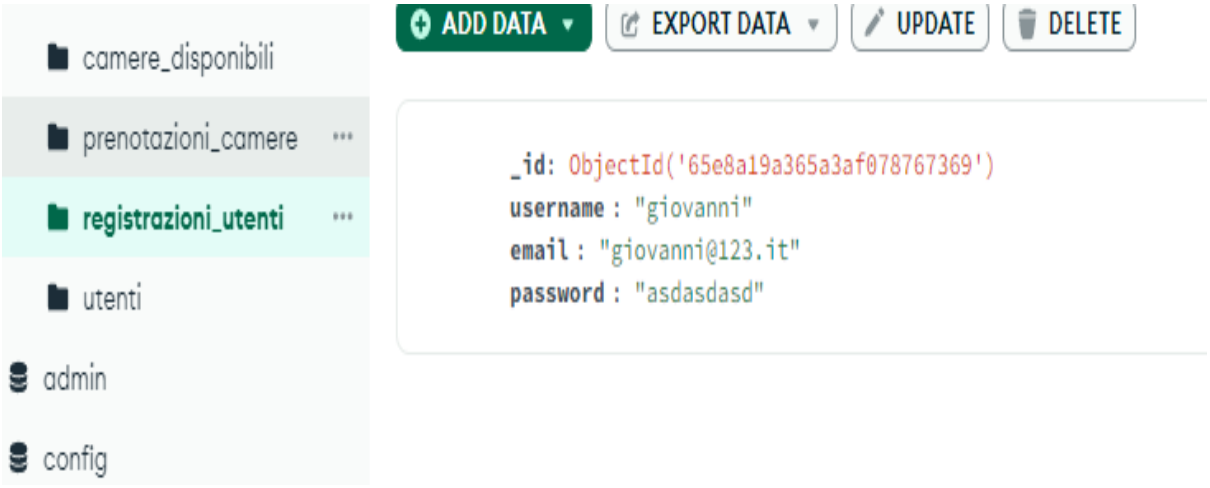


*Figura **13**– Esempio di popolamento collezione dopo la registrazione utente.*

Come la figura **13** mostra, abbiamo anche introdotto, Il modulo per la registrazione utente che funziona allo stesso modo della prenotazione, assegnando quindi un ID univoco ad ogni utente registrato, username , email e password che il cliente ci ha fornito compilando i campi o modulo richiesto.

.

12. UML Analisi e Schema del DB

Possiamo definire la parte UML di analisi e del diagramma del database per il progetto.

UML di Analisi:

Diagramma dei casi d'uso:

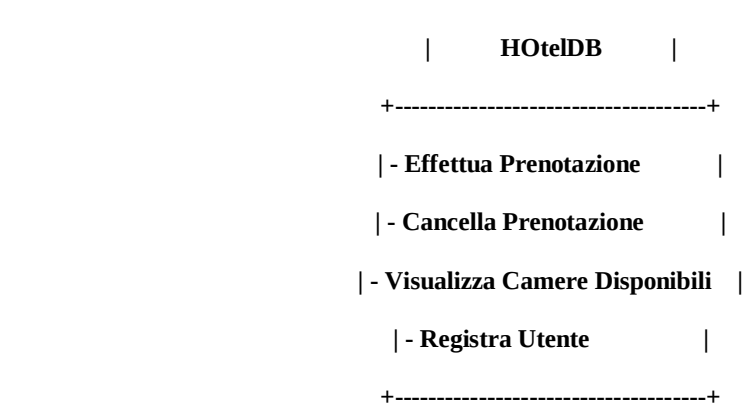
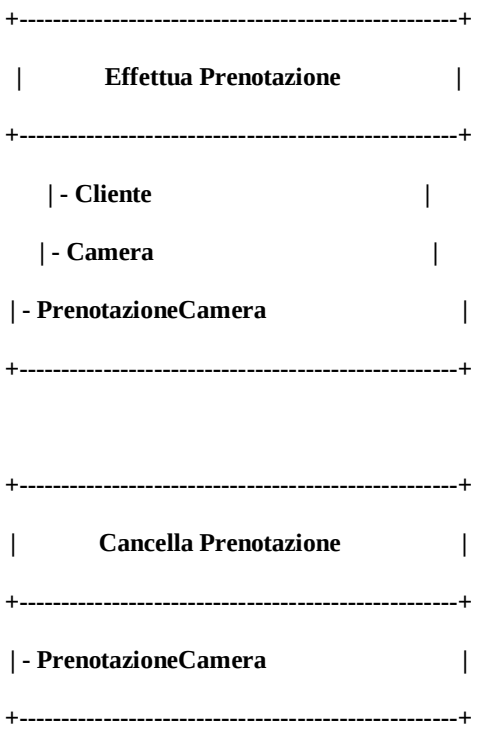
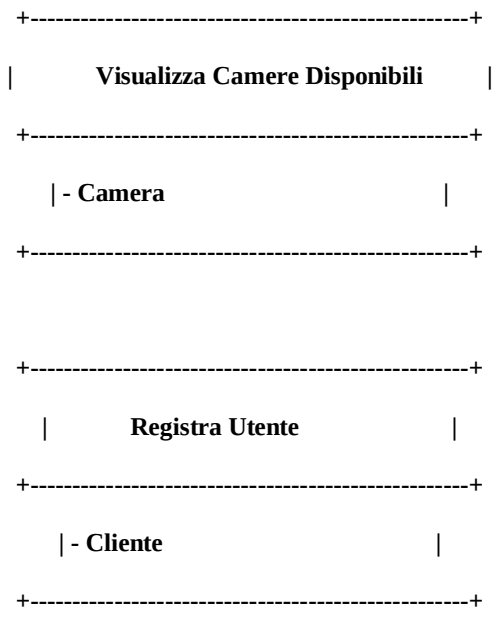


Diagramma delle sequenze:

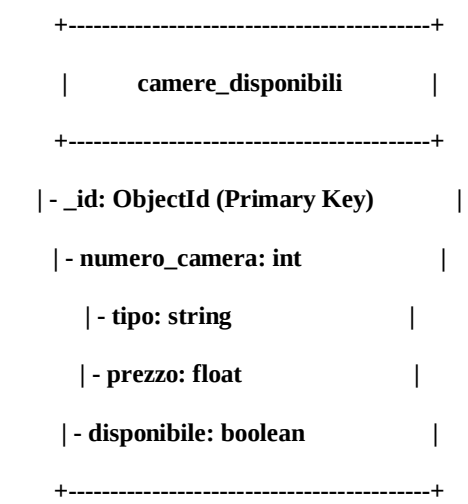




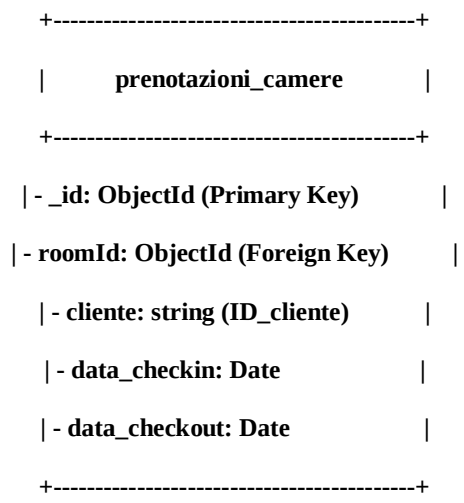
12.1. Diagramma del Database:

Schema del Database MongoDB:

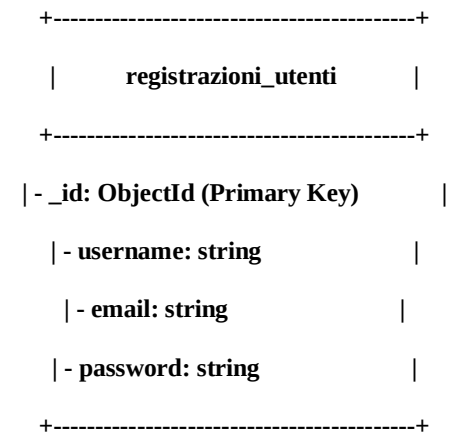
Collezione "camere_disponibili":



Collezione "prenotazioni_camere":



Collezione "registrazioni_utenti":



Questo schema rappresenta la struttura del database MongoDB per il progetto di gestione delle prenotazioni delle camere in un hotel. Le collezioni sono progettate per memorizzare le informazioni sulle camere disponibili, sulle prenotazioni effettuate dai clienti e sui dati degli utenti registrati.

13. Test e valutazione

13.1. Test della Web App e risultati ottenuti

Per valutare le prestazioni e l'efficacia della mia Web App di prenotazione alberghiera, ho condotto una serie di test approfonditi che hanno coinvolto sia il frontend che il backend del sistema. I test sono stati progettati per valutare diversi aspetti, tra cui la velocità di risposta, la scalabilità e l'efficienza complessiva dell'applicazione.

13.2 Test del frontend: Per testare il frontend della Web App, ho utilizzato tutto in locale lanciando il backend connesso al mio DB MongoDB , lanciando il mio frontend ovvero l'index.html la mia webapp si apre come mostrato in precedenza nelle immagini. I risultati di questi test hanno mostrato un'esperienza utente fluida e responsiva, con tempi di caricamento delle pagine adeguati e una navigazione intuitiva attraverso le varie sezioni dell'applicazione.

13.3 Test del backend: Per testare il backend della Web App, mi sono concentrato principalmente sull'efficienza delle operazioni CRUD eseguite su MongoDB, comprese le operazioni di lettura e scrittura dei dati delle camere e delle prenotazioni.

13.4 Utilità di MongoDB nel contesto del progetto: MongoDB ha dimostrato di essere un'ottima scelta per il backend della mia Web App di prenotazione alberghiera per diversi motivi:

- **Flessibilità dello schema:** MongoDB consente di modellare i dati in modo flessibile, adattandoli facilmente ai requisiti in continua evoluzione della mia applicazione.
- **Velocità di sviluppo:** Grazie al suo approccio orientato ai documenti e alla sua sintassi semplice, MongoDB ha accelerato lo sviluppo del backend, consentendo una rapida iterazione e implementazione delle nuove funzionalità.

- **Scalabilità:** MongoDB offre una scalabilità orizzontale nativa, consentendo alla mia applicazione di crescere facilmente con l'aumentare del carico di lavoro e del numero di utenti.

In conclusione, i test condotti hanno confermato l'efficacia di MongoDB nel supportare le esigenze del mio progetto di tesi, fornendo un backend robusto, performante e scalabile per la mia Web App di prenotazione alberghiera.

14. Conclusioni e sviluppi futuri

Dopo aver mostrato come il mio progetto della WebApp, ci mosta come attraverso MongoDB possiamo creare un sistema di prenotazione alberghiero, ci sarebbero molti sviluppi futuri da implementare in quanto il mio è solamente un test di come possiamo creare una WebApp con MongoDB. Ecco allora cosa potremmo fare per avere una webapp completa ed impeccabile.

- **Sistema di Autenticazione degli Utenti**: Implementa un sistema di autenticazione per consentire agli utenti di registrarsi, accedere e gestire le proprie prenotazioni.
- **Pagina Dettagli Camera**: Crea una pagina dettagliata per ogni camera che fornisca informazioni dettagliate sulla camera, come foto, descrizione, servizi inclusi, recensioni degli utenti, disponibilità e prezzi.
- **Sistema di Recensioni**: Aggiungi la possibilità per gli utenti di lasciare recensioni e valutazioni per le camere dopo aver soggiornato. Mostra le recensioni sulla pagina dettagli camera per aiutare gli utenti nella scelta.
- **Filtri di Ricerca Avanzati**: Implementa filtri di ricerca avanzati per consentire agli utenti di cercare camere in base a criteri specifici come prezzo, tipo di camera, servizi inclusi, recensioni e altro ancora.
- **Sistema di Prenotazione Avanzato**: Migliora il sistema di prenotazione consentendo agli utenti di selezionare date di arrivo e partenza, numero di ospiti, preferenze e altri dettagli. Aggiungi la possibilità di modificare o cancellare le prenotazioni esistenti. push.

- **Integrazione con Servizi di Pagamento**: Aggiungi un sistema di pagamento per consentire agli utenti di effettuare pagamenti online in modo sicuro e conveniente.
- **Pannello di Amministrazione**: Crea un pannello di amministrazione per i gestori dell'hotel per gestire camere, prenotazioni, utenti, recensioni e altre attività correlate.
- **Versione Mobile**: Ottimizza l'app per dispositivi mobili creando una versione responsiva o un'app mobile dedicata per consentire agli utenti di prenotare camere facilmente da smartphone e tablet.
- **Test e Ottimizzazione delle Prestazioni**: Effettua test approfonditi e ottimizza le prestazioni dell'app per garantire un'esperienza utente fluida e veloce anche in situazioni di traffico intenso.

15. Ringraziamenti.

Nella stesura di questa tesi, è stato per me fondamentale il supporto di tante persone: senza il loro aiuto il mio lavoro non sarebbe stato così completo e il mio percorso sarebbe stato sicuramente più difficile.

Ringrazio il professore Filippo Sciarrore per la massima disponibilita' offerta anche in notturna con varie mail.

Ringrazio il professore Stefano D'urso per avermi aiutato a risolvere qualche intoppo incontrato sullo sviluppo del mio progetto.

Volevo Ringraziare di vero cuore i miei genitori, mio fratello e mia cognata che mi hanno sempre supportato come hanno potuto.

Ringrazio i miei parenti, soprattutto mia zia Maria e mio zio Mario che hanno contribuito ad aiutarmi anche economicamente.

Ringrazio inoltre gli amici, Francesco e Salvatore Puleo ovvero i miei datori di lavoro che mi hanno dato l'opportunita' di studiare durante il mio percorso lavorativo con loro.

Ringrazio il mio padrino Andy, che mi ha sostenuto sempre.

Ringrazio i miei cugini, Seby, Nico, Masina e Giovanni che mi hanno dato molti consigli.

Ringrazio Daniela che mi ha convinto che tutto cio' era possibile esortandomi a riprendere gli studi che in passato avevo abbandonato.

In fine Ringrazio inoltre tutti coloro che mi sono stati vicini e creduto in me.