Hotel Recommendation

Full-Stack website

A project report by

DEV VATSA (CSE-DATA SCIENCE)	-2162029
PRATHAM R SHARMA(CSE-DATA SCIENCE)	-2162034
RISHI RANJAN SINHA(CSE-DATA SCIENCE)	-2162024
JAI SATYAM THAKUR(CSE-DATA SCIENCE)	-2162026

In supervision of

Professor SUDESHNA GHOSH

in partial fulfilment for the award of the degree of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING(Data Science)
HERITAGE INSTITUTE OF TECHNOLOGY, KOLKATA
An autonomous Institute Under

Maulana Abdul Kalam Azad University of Technology
Formerly Known as
West Bengal University of Technology

HERITAGE INSTITUTE OF TECHNOLOGY, KOLKATA MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

BONAFIDE CERTIFICATE

Certified that this project report "LuxeStay: Hotel Recommendation Full-Stack website" is the Bonafide work of "Mr. Dev Vatsa, Mr. Jai Satyam Thakur, Mr. Pratham R Sharma, and Mr. Rishi Ranjan Sinha" who carried out the project under my supervision.

Professor SUDESHNA GHOSH	Dr. SUBHAJIT DATTA
------------------------------------	------------------------------------

Dev Vatsa

Jai Styam Thakur

Pratham R Sharma

Rishi Ranjan Sinha

ACKNOWLEDGEMENT

We would take this opportunity to thank Prof. (Dr.) Basab Chaudhuri, Principal Heritage Institute of Technology for providing us with all the necessary facilities to make our project work successful. We would like to thank our Head of the Department Prof. (Dr.) Subhajit Datta for his kind assistance as and when required.

I would like to express my sincere gratitude to my mentor, **SUDESHNA GHOSH**, for her invaluable guidance, support, and encouragement throughout the course of this project. Her insights, constructive feedback, and constant motivation played a pivotal role in shaping this hotel recommendation system.

Her expertise and patience have been instrumental in helping me navigate through the challenges faced during the development of the system. I am truly grateful for the time and effort she dedicated to guiding me through the various stages of the project, from conceptualization to implementation.

I would also like to thank my peers and everyone who provided their assistance and encouragement during this project. Their suggestions and moral support have been invaluable in completing this project successfully.

Lastly, I would like to extend my gratitude to my family and friends for their continuous support and understanding during the course of this work.

ABSTRACT

This project presents a full-stack hotel recommendation system designed to enhance the user experience in selecting accommodations. The system integrates a machine learning-based recommendation engine with a robust, user-friendly dashboard and additional pages like login, registration, and user profile management.

The recommendation system leverages TF-IDF (Term Frequency-Inverse Document Frequency) and cosine similarity to analyze and match user preferences with hotel attributes, ensuring personalized suggestions. The frontend, developed using Next.js and styled with Tailwind CSS, provides an interactive interface for users to explore hotels. The backend, powered by FastAPI, processes user requests, interacts with the ML model, and manages data securely.

The project includes essential features such as secure authentication, a responsive design, and seamless navigation across pages like hotel recommendations, user registration, login, and profile management. The results demonstrate the system's capability to provide accurate and efficient hotel recommendations while maintaining a high standard of usability. Future work involves integrating hybrid recommendation methods, expanding datasets, and incorporating user feedback loops to further refine the system.

Table of Contents

INTRODUCTION

1.1 Project Overview

In the modern era of digitization, the hospitality industry has witnessed a paradigm shift, with users increasingly relying on online platforms to discover and book accommodations. This project aims to develop a comprehensive website that not only facilitates hotel discovery but also enhances the user experience through personalized recommendations. The system integrates cutting-edge recommendation algorithms, an intuitive user interface, and additional features to make hotel selection efficient and engaging.

The website provides users with detailed descriptions of each hotel, helping them make informed decisions. It includes an interactive map section for easy navigation, a reel section where users and hotels can upload short videos, and the ability for users to save their favorite hotels. These saved hotels further refine the recommendation engine by aligning suggestions with user preferences.

1.2 Problem Statement

Choosing the right hotel often involves manually browsing through numerous options, which can be time-consuming and overwhelming. Users struggle to find accommodations tailored to their specific needs and preferences, and existing systems often lack personalization. This project addresses this gap by utilizing advanced recommendation techniques to suggest hotels that closely match individual user preferences based on saved hotels, interactions, and descriptions.

1.3 Objectives

The primary objective of this project is to build a full-stack hotel recommendation system that offers an engaging and user-friendly experience. Key objectives include:

- Personalized Recommendations: Using saved hotels, user interactions, and hotel descriptions to generate accurate suggestions.
- Dynamic Features: Incorporating an interactive map and reel section to make hotel exploration more immersive and engaging.
- Seamless Navigation: Providing a robust platform with features like login, registration, and a dashboard for hotel recommendations.
- User-Centric Design: Enabling users to save hotels and use them to refine the recommendations over time.

LITERATURE

2.1 Existing Systems

The development of hotel recommendation systems has been driven by the need to enhance user experiences in selecting accommodations. Existing systems primarily rely on traditional recommendation techniques such as content-based filtering, collaborative filtering, and hybrid models. Prominent platforms like Booking.com and Airbnb use these techniques to offer suggestions based on user interactions, past bookings, and reviews. While effective, these systems often face limitations such as the cold start problem, lack of contextual recommendations, and insufficient integration of diverse features like media content or geolocation data.

2.2 Recommendation Techniques

The backbone of any recommendation system lies in its underlying algorithm. Below are some widely used techniques relevant to hotel recommendation systems:

1. Content-Based Filtering:
    - Relies on the features of items (e.g., hotel descriptions, amenities, locations) to match user preferences.
    - Uses natural language processing (NLP) techniques like TF-IDF (Term Frequency-Inverse Document Frequency) and cosine similarity to analyze textual descriptions and generate personalized suggestions.
    - Strength: Effective for new users with no prior interactions.
    - Limitation: Struggles with diversity in recommendations as it tends to focus only on similar items.
2. Collaborative Filtering:
    - Suggests items based on user-user or item-item similarities derived from interaction data.
    - User-Based Collaborative Filtering identifies users with similar preferences, while Item-Based Collaborative Filtering focuses on shared item characteristics.
    - Strength: Learns patterns from user behavior.
    - Limitation: Suffers from the cold start problem for new users or hotels with no prior data.
3. Hybrid Recommendation Models:
    - Combine content-based and collaborative filtering to leverage the strengths of both approaches.
    - Hybrid models are more robust and can handle diverse recommendation scenarios.

2.3 Advances in Recommendation Systems

Recent advances in recommendation systems have explored the integration of multimedia content and geolocation to make suggestions more contextual and engaging:

- Multimedia Integration:
  Systems now incorporate user-generated media such as photos, videos, and reviews to provide richer insights into accommodations. For example, the reel section in this project allows users to upload videos, enabling a visual and dynamic interaction with hotel content.
- Geolocation-Based Recommendations:
  Maps and geolocation services are increasingly used to enhance the relevance of recommendations. By integrating map features, users can explore hotels based on proximity, landmarks, or specific areas of interest.
- Personalization Through Saved Data:
  Modern systems use saved preferences, bookings, or wishlists to refine their suggestions. In this project, saved hotels play a critical role in tailoring recommendations to individual users.

2.4 Relevance to This Project

This project builds upon these advancements by integrating multiple features into a single cohesive system. It leverages content-based filtering with TF-IDF and cosine similarity for recommendations, incorporates media content via a reel section, and uses geolocation through a map interface. By combining these elements, the system addresses gaps in existing platforms, offering a more immersive and personalized experience.

SYSTEM DESIGN AND ARCHITECTURE

3.1 High-Level Architecture

The system architecture for the hotel recommendation system is designed to seamlessly integrate the recommendation engine with an interactive and user-friendly web application. It consists of three main layers:

1. **Frontend Layer**:
   The frontend serves as the user interface, allowing users to interact with the system. It is built using **Next.js** for server-side rendering and responsiveness, styled with **Tailwind CSS**, and enriched with JavaScript for dynamic functionality. Key frontend components include:
   - **Dashboard**: Displays recommended hotels based on user preferences.
   - **Reel Section**: Allows users and hotels to upload and view videos.
   - **Map Interface**: Provides an interactive way to explore hotel locations.
   - **Authentication Pages**: Login and registration forms for secure access.
2. **Backend Layer**:
   The backend manages data flow, processes requests, and connects the machine learning model with the web application. Built using **FastAPI**, the backend is lightweight and highly efficient. Key responsibilities include:
   - Processing user requests for recommendations.
   - Managing saved hotel data for personalized suggestions.
   - Handling uploads for the reel section.
   - Communicating with the ML model for recommendation generation.
3. **Recommendation Engine**:
   At the core of the system, the recommendation engine uses a **content-based filtering approach**:
   - **TF-IDF (Term Frequency-Inverse Document Frequency)**: Converts textual descriptions of hotels into feature vectors.
   - **Cosine Similarity**: Computes similarity scores between user preferences and hotel attributes to generate recommendations.
   - **Saved Hotel Data**: Enhances personalization by incorporating previously saved hotels into the recommendation process.

4. **Database**:
   - Stores user data, saved hotels, hotel attributes, and uploaded reel content.
   - Used by both the backend and recommendation engine for efficient data retrieval.

3.2 Technology Stack

The project utilizes the following technologies:

Component	Technology
Frontend	HTML, CSS, JavaScript, Next.js, Tailwind CSS
Backend	Node.js, FastAPI
Recommendation Model	Python, Scikit-learn
Styling	Tailwind CSS
API Integration	FastAPI

These technologies were chosen for their scalability, performance, and ease of integration.

3.3 Design Choices

Key design decisions for the system include:

1. FastAPI for Model Integration:
   FastAPI was selected as the backend framework due to its asynchronous capabilities, speed, and support for connecting Python-based machine learning models with web applications. This allows for real-time recommendations.
2. Next.js for Frontend:
   Next.js was chosen for its ability to render pages server-side, improving performance and SEO. Its compatibility with Tailwind CSS ensures a responsive and visually appealing user interface.
3. Content-Based Filtering for Recommendations:
   A content-based filtering approach was implemented as it aligns well with the system's focus on hotel descriptions and user preferences. TF-IDF and cosine similarity provide a robust way to quantify textual similarities.
4. Interactive Features:
   - The reel section adds a dynamic visual element, allowing users to engage with video content uploaded by hotels or other users.
   - The map interface enhances the exploration experience by providing location-based recommendations.
5. Personalization Through Saved Data:
   The system uses data from saved hotels to refine recommendations, ensuring a tailored experience for each user.

3.4 Data Flow

The data flow within the system operates as follows:

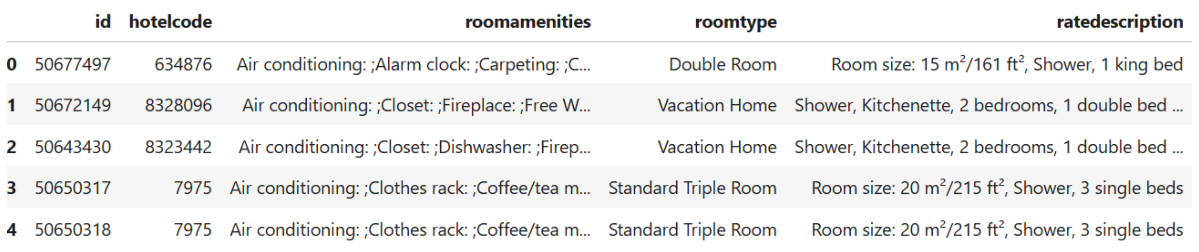
1. Frontend Interaction:
    - Users interact with the web application to log in, save hotels, or explore recommendations.
    - Requests are sent to the backend via API endpoints.
2. Backend Processing:
    - FastAPI handles incoming requests, processes the data, and communicates with the recommendation engine.
    - The backend retrieves user preferences and hotel attributes from the database.
3. Recommendation Generation:
    - The recommendation engine processes user inputs and saved hotels using TF-IDF and cosine similarity.
    - Results are sent back to the backend for further processing.
4. Frontend Display:
    - Recommendations, maps, and reel content are rendered on the dashboard for the user to explore.

METHODOLOGY

The hotel recommendation system was developed using a structured approach that integrates advanced machine learning techniques with robust web technologies. This section outlines the methodology employed, detailing each step of the process, the tools and libraries used, and the rationale behind choosing specific technologies while explaining why others were not adopted.

4.1 Data Collection and Preprocessing

The foundation of any recommendation system lies in its data. For this project, the dataset was curated to include essential attributes such as hotel names, descriptions, locations, ratings, room types, and amenities. The data was obtained from publicly available sources, with synthetic entries added to simulate realistic user interactions for testing purposes. The system also dynamically collects user interaction data, such as saved hotels and preferences, during runtime.The raw data we collected had three table.One contained the details about the amenities provided by the hotel.



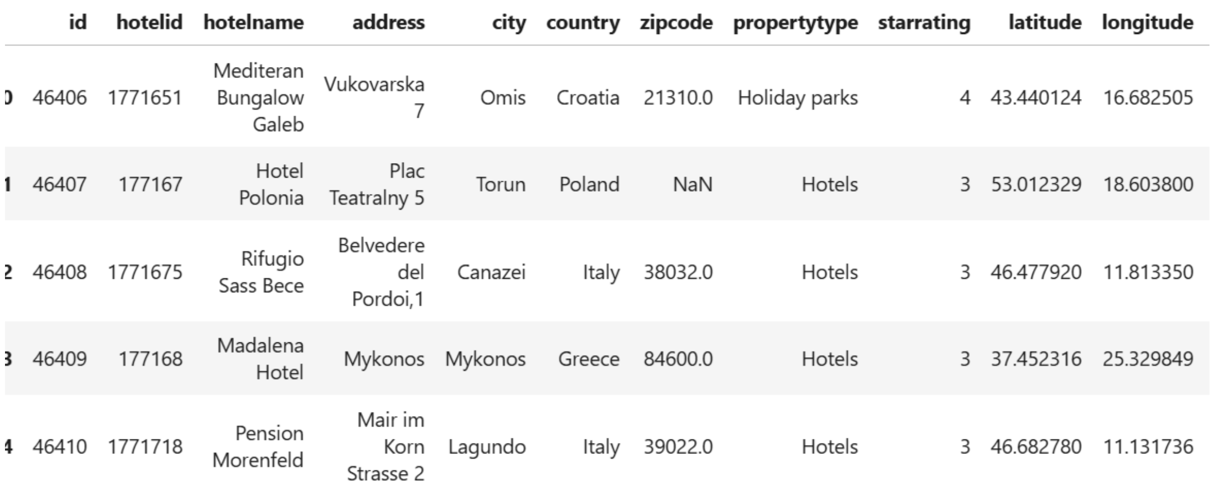
	id	hotelcode	roomamenities	roomtype	ratedescription
0	50677497	634876	Air conditioning: ;Alarm clock: ;Carpeting: ;C...	Double Room	Room size: 15 m^2/161 ft^2, Shower, 1 king bed
1	50672149	8328096	Air conditioning: ;Closet: ;Fireplace: ;Free W...	Vacation Home	Shower, Kitchenette, 2 bedrooms, 1 double bed ...
2	50643430	8323442	Air conditioning: ;Closet: ;Dishwasher: ;Firep...	Vacation Home	Shower, Kitchenette, 2 bedrooms, 1 double bed ...
3	50650317	7975	Air conditioning: ;Clothes rack: ;Coffee/tea m...	Standard Triple Room	Room size: 20 m^2/215 ft^2, Shower, 3 single beds
4	50650318	7975	Air conditioning: ;Clothes rack: ;Coffee/tea m...	Standard Triple Room	Room size: 20 m^2/215 ft^2, Shower, 3 single beds

Image 1:**Hotel Room Attribute**

The second had all the details about the hotels like its location , address and many more.

Preprocessing was a critical step to ensure the data was clean, consistent, and ready for analysis. Null values and duplicates were removed to maintain data integrity. Textual data, such as hotel descriptions, underwent tokenization and normalization. Tokenization split descriptions into individual words, while normalization converted text to lowercase and removed stopwords to focus on meaningful keywords. These steps ensured that only the most relevant terms were used in the recommendation process.

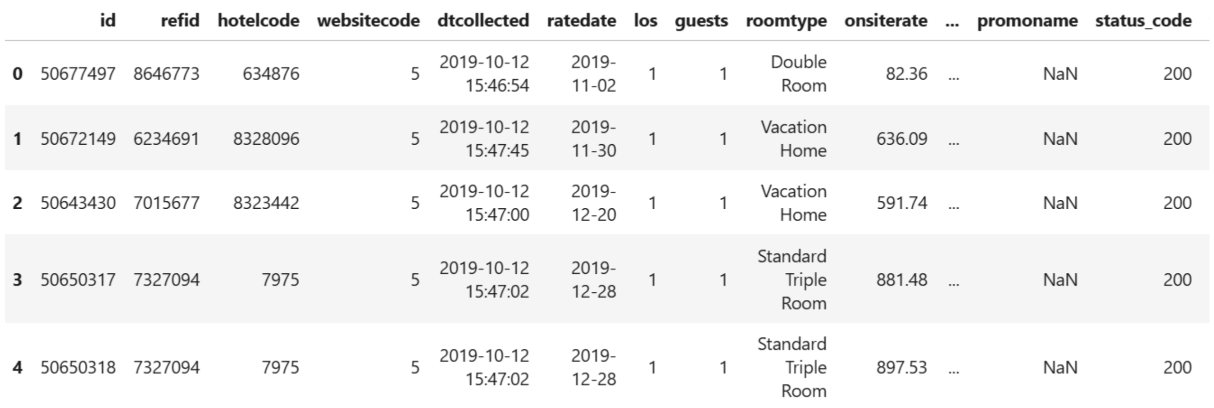
Additionally, feature engineering played a significant role. Key attributes such as hotel descriptions, amenities, and locations were combined into a single `tags`



	id	hotelid	hotelname	address	city	country	zipcode	propertytype	starrating	latitude	longitude
0	46406	1771651	Mediteran Bungalow Galeb	Vukovarska 7	Omis	Croatia	21310.0	Holiday parks	4	43.440124	16.682505
1	46407	177167	Hotel Polonia	Plac Teatralny 5	Torun	Poland	NaN	Hotels	3	53.012329	18.603800
2	46408	1771675	Rifugio Sass Bece	Belvedere del Pordoi,1	Canazei	Italy	38032.0	Hotels	3	46.477920	11.813350
3	46409	177168	Madalena Hotel	Mykonos	Mykonos	Greece	84600.0	Hotels	3	37.452316	25.329849
4	46410	1771718	Pension Morenfeld	Mair im Korn Strasse 2	Lagundo	Italy	39022.0	Hotels	3	46.682780	11.131736

Image 2: Hotel details

The third table had the pricing and the taxes details.



	id	refid	hotelcode	websitecode	dtcollected	ratedate	los	guests	roomtype	onsiterate	...	promoname	status_code
0	50677497	8646773	634876	5	2019-10-12 15:46:54	2019-11-02	1	1	Double Room	82.36	...	NaN	200
1	50672149	6234691	8328096	5	2019-10-12 15:47:45	2019-11-30	1	1	Vacation Home	636.09	...	NaN	200
2	50643430	7015677	8323442	5	2019-10-12 15:47:00	2019-12-20	1	1	Vacation Home	591.74	...	NaN	200
3	50650317	7327094	7975	5	2019-10-12 15:47:02	2019-12-28	1	1	Standard Triple Room	881.48	...	NaN	200
4	50650318	7327094	7975	5	2019-10-12 15:47:02	2019-12-28	1	1	Standard Triple Room	897.53	...	NaN	200

Image 3: Hotel price

However there were too many details in overall data. So we dropped unnecessary data. Final data had only seven columns.

column, which acted as the core input for the machine learning model. The TF-IDF (Term Frequency-Inverse Document Frequency) technique was then applied to these tags, transforming them into numerical vectors that quantify the importance of each word. This allowed the system to effectively compare hotel descriptions and user preferences.

Python was chosen for this stage due to its extensive libraries and ease of implementation. Pandas was employed for efficient data manipulation, while NLTK handled text tokenization and stopword removal. Scikit-learn provided the TF-IDF vectorizer, a proven tool for textual data transformation. Alternatives like R or Apache Spark were considered but deemed unsuitable due to their lack of seamless integration with web applications (in the case of R) or unnecessary computational overhead for the project’s scale (in the case of Spark).

4.2 Recommendation Model Design

The recommendation system was designed using a content-based filtering approach, which analyzes the attributes of hotels to match them with user preferences. This approach is well-suited for the project as the dataset contains rich metadata about hotels, such as descriptions and amenities.

The TF-IDF technique was used to convert textual descriptions into feature vectors, with cosine similarity employed to calculate the relevance between user preferences and hotel descriptions. TF-IDF assigns weights to words based on their frequency in a single document relative to their frequency across all documents, ensuring that common words have less impact while unique words carry more significance. Cosine similarity, on the other hand, measures the angle between two vectors in a multidimensional space, providing a score that represents how closely two pieces of text align.

To enhance personalization, the system incorporated user interactions, particularly saved hotels. These saved entries acted as a proxy for user preferences, further refining the recommendations by prioritizing hotels similar to those the user had previously shown interest in.

While alternatives like collaborative filtering or deep learning-based approaches were considered, they were not implemented in this phase. Collaborative filtering requires extensive user interaction data, which was not available in the early stages of the project. Deep learning models, though powerful, were deemed unnecessary due to their high computational requirements and the modest scale of the dataset.

4.3 Backend Integration

The backend of the system was developed using FastAPI, a modern framework known for its speed, simplicity, and support for asynchronous programming. FastAPI served as the bridge between the frontend and the recommendation engine, handling API requests and responses. It facilitated real-time communication between the components, allowing user inputs to be processed seamlessly and recommendations to be delivered promptly.

FastAPI was chosen over other frameworks like Flask and Django due to its lightweight nature and native support for asynchronous operations. Flask, while similar in functionality, lacks built-in asynchronous capabilities, which are essential for handling multiple concurrent requests efficiently. Django, though powerful, is more suited for large-scale, server-heavy applications and was considered overly complex for this project’s requirements.

The backend managed several critical operations, including routing user inputs to the recommendation engine, fetching and storing data from the database, and handling file uploads for the reel section. JSON was used as the standard data exchange format, ensuring compatibility and ease of integration with the frontend.

4.4 Frontend Development

The frontend was built using Next.js, a React-based framework that supports server-side rendering (SSR) and static site generation (SSG). These features were crucial for optimizing the system’s performance and enhancing its search engine visibility. The interface was designed to be intuitive and responsive, with key components such as a dashboard for displaying recommendations, a reel section for viewing and uploading short videos, and a map interface for location-based exploration.

Tailwind CSS was employed for styling due to its utility-first approach, which allows developers to create custom designs quickly without the constraints of pre-designed components. This provided the flexibility to create a unique and consistent user experience across the application. Tailwind CSS was preferred over frameworks like Bootstrap, which, while robust, imposes a more rigid structure and styling.

4.5 Database Design

The database played a vital role in storing user profiles, hotel attributes, saved hotels, and reel content. MongoDB, a NoSQL database, was chosen for its flexibility and ability to handle unstructured data efficiently. MongoDB's schema-less structure made it ideal for managing diverse data types, such as text, images, and videos.

SQL-based databases were considered but ultimately not used due to their rigid schema requirements, which could have posed challenges as the system evolved. MongoDB’s scalability and ease of integration with FastAPI further solidified its position as the preferred database solution.

4.6 System Workflow

The system’s workflow is as follows: Users interact with the frontend, providing inputs such as saved hotels or uploaded reels. These inputs are sent to the backend via API calls, where they are processed and routed to the recommendation engine. The engine computes recommendations using TF-IDF and cosine similarity, sending the results back to the backend. The backend formats these results and sends them to the frontend for display. All interactions and data changes are logged in the database to maintain consistency and improve future recommendations.

This methodology combines thoughtful design, efficient tools, and clear rationales to deliver a robust hotel recommendation system that prioritizes both functionality and user experience.

SYSTEM IMPLEMENTATION

The implementation of the hotel recommendation system involved building a seamless integration of the frontend, backend, and machine learning model to provide an intuitive, responsive, and accurate experience for users. The focus was on leveraging the strengths of each technology and ensuring smooth communication between components. This section outlines the implementation details for each part of the system.

5.1 Frontend Implementation

The frontend of the application was developed using **Next.js**, a React-based framework renowned for its server-side rendering (SSR) capabilities and static site generation (SSG). These features were essential in optimizing page loading speeds, improving SEO, and creating a dynamic user experience.

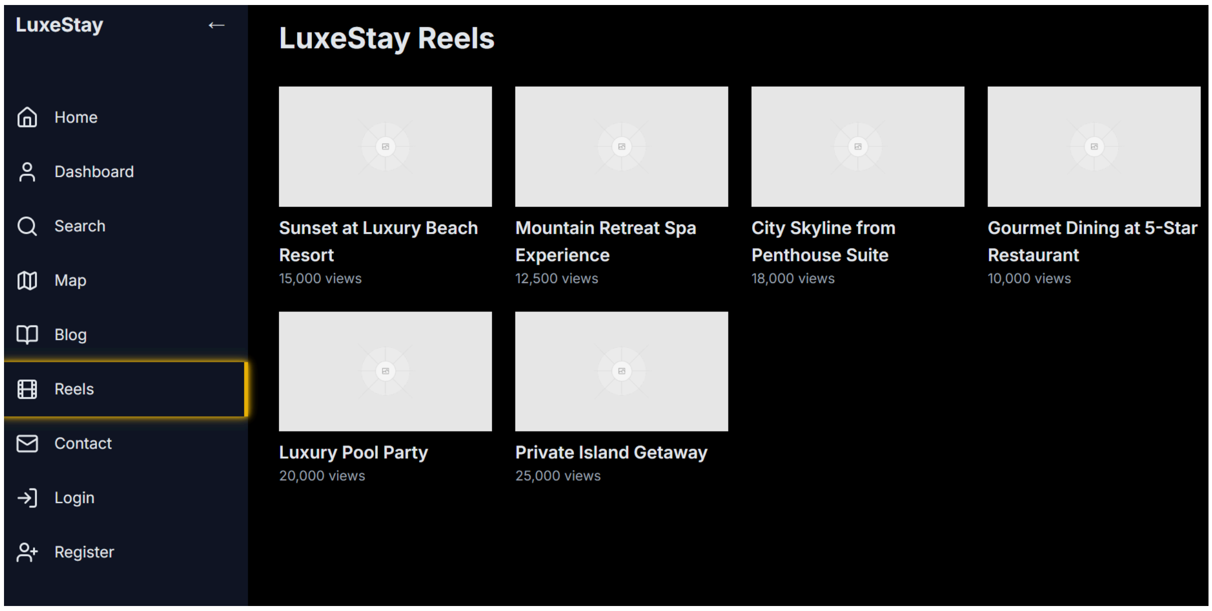


Image 4: Reel section

The application was designed with a clear emphasis on usability and responsiveness. The **dashboard** serves as the primary interface where users can view personalized hotel recommendations. Each recommendation is displayed as a card containing the hotel name, description, location, rating, and options to save or explore further. The **reel section**, an interactive feature, allows users and hotels to upload and browse short videos that highlight unique aspects of accommodations, making the exploration process visually engaging.

An additional **map interface** enables users to explore hotels based on their geolocation or proximity to landmarks. This was implemented using mapping APIs, ensuring a smooth and interactive experience for users.

For styling, **Tailwind CSS** was chosen for its utility-first approach, allowing for rapid prototyping and consistent design. Tailwind's responsive grid and utility classes simplified the creation of a visually appealing and adaptive layout without relying on pre-built components that might limit customization. The **authentication pages**—including login and registration—were designed with simplicity and security in mind, ensuring a streamlined user onboarding process.

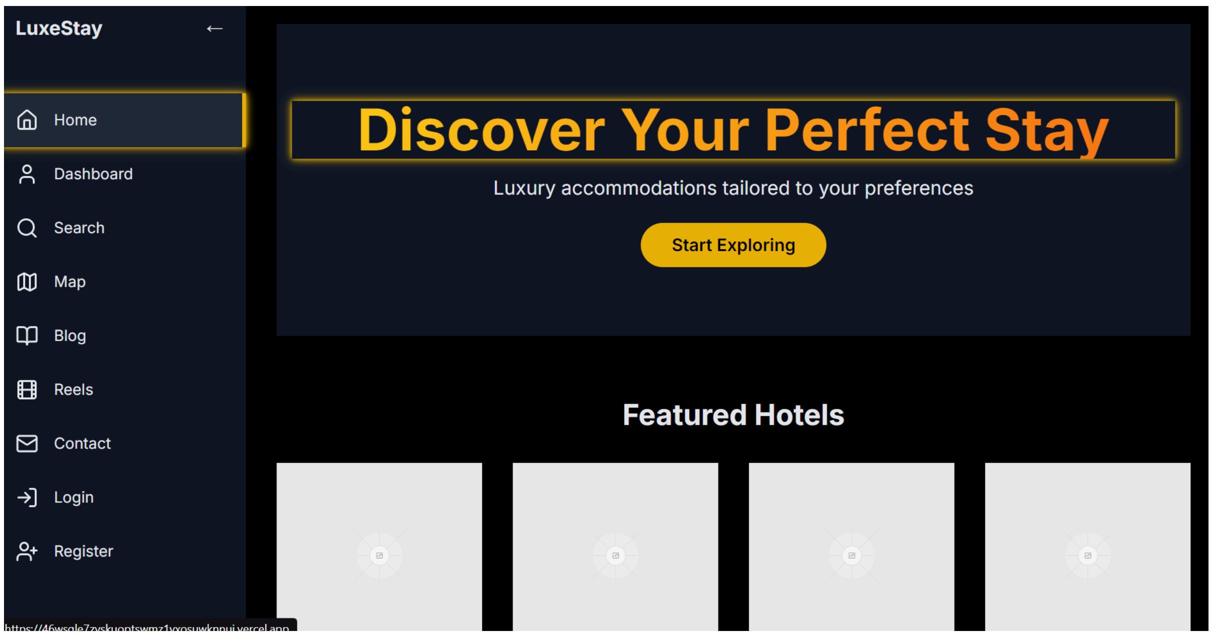


Image 5: Home page

To manage client-side state effectively, **React Context API** was used. This allowed for efficient handling of user session data and interactions, reducing the reliance on redundant API calls.

5.2 Backend Implementation

The backend, implemented using **FastAPI**, acted as the backbone of the system, managing data flow, handling API requests, and bridging the frontend with the machine learning model. FastAPI's asynchronous programming capabilities enabled the system to handle multiple concurrent user requests without performance bottlenecks, ensuring a smooth user experience even under heavy load.

FastAPI’s design facilitated the creation of lightweight and maintainable API endpoints. For example, endpoints were developed for:

1. **Recommendation Generation**: Accepts user data, processes it through the machine learning model, and returns a ranked list of hotels.
2. **Hotel Saving**: Allows users to save hotels, which is then stored in the database and used to refine future recommendations.
3. **Reel Uploads**: Processes video uploads from users or hotels, storing them in the database and linking them to respective hotels for display in the reel section.
4. **User Authentication**: Implements secure login and registration using JWT (JSON Web Tokens) for managing user sessions.

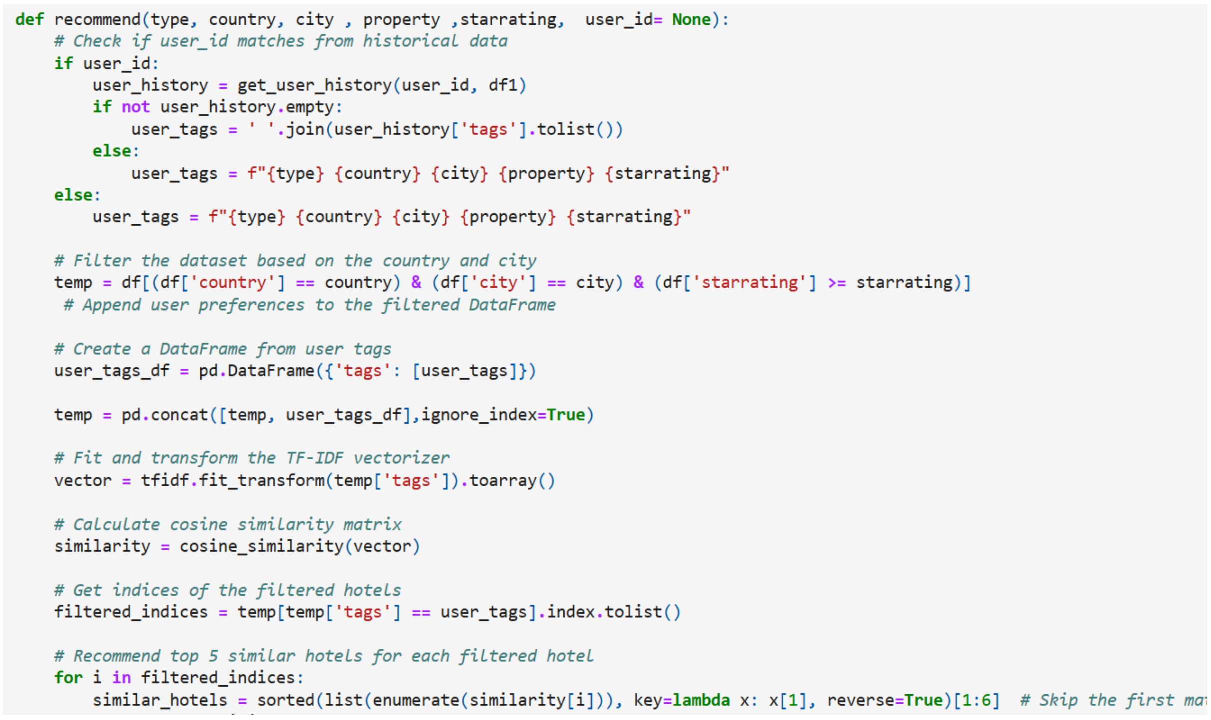
The backend also handled validations and error responses. For instance, inputs from the frontend were validated for completeness and accuracy before being processed. If errors occurred, such as invalid data or database connection issues, user-friendly error messages were generated and returned to the frontend.

Communication between the backend and the machine learning model was seamless, with FastAPI serving as the intermediary. The model was hosted within the same environment as the backend to minimize latency. Requests from the frontend, such as fetching recommendations, were routed through FastAPI to the model, and the responses were sent back to the frontend after necessary formatting.

5.3 Recommendation Model Integration

The recommendation engine, a core part of the system, was implemented using Python. The **TF-IDF vectorizer** and **cosine similarity** methods from the Scikit-learn library formed the foundation of the model. The preprocessing pipeline transformed textual hotel descriptions into numerical feature vectors, while cosine similarity calculated the relevance of hotels to user preferences.

The model operated dynamically, incorporating user interactions such as saved hotels to refine recommendations. Whenever a user saved a hotel, the system updated their preference profile, ensuring that future recommendations became increasingly tailored. The backend fetched the relevant data from the database, processed it through the model, and returned the results in a JSON format to the frontend.



```python
def recommend(type, country, city , property ,starrating,  user_id= None):
    # Check if user_id matches from historical data
    if user_id:
        user_history = get_user_history(user_id, df1)
        if not user_history.empty:
            user_tags = ' '.join(user_history['tags'].tolist())
        else:
            user_tags = f"{type} {country} {city} {property} {starrating}"
    else:
        user_tags = f"{type} {country} {city} {property} {starrating}"

    # Filter the dataset based on the country and city
    temp = df[(df['country'] == country) & (df['city'] == city) & (df['starrating'] >= starrating)]
     # Append user preferences to the filtered DataFrame

    # Create a DataFrame from user tags
    user_tags_df = pd.DataFrame({'tags': [user_tags]})

    temp = pd.concat([temp, user_tags_df],ignore_index=True)

    # Fit and transform the TF-IDF vectorizer
    vector = tfidf.fit_transform(temp['tags']).toarray()

    # Calculate cosine similarity matrix
    similarity = cosine_similarity(vector)

    # Get indices of the filtered hotels
    filtered_indices = temp[temp['tags'] == user_tags].index.tolist()

    # Recommend top 5 similar hotels for each filtered hotel
    for i in filtered_indices:
        similar_hotels = sorted(list(enumerate(similarity[i])), key=lambda x: x[1], reverse=True)[1:6]  # Skip the first ma
```

Image 6: Code for recommendation function

The decision to use TF-IDF and cosine similarity was driven by the need for a lightweight, efficient, and interpretable solution. While more advanced techniques like deep learning could have been employed, they were deemed unnecessary given the scale of the dataset and the project's real-time requirements.

5.4 Database Implementation

The database was implemented using **MongoDB**, a NoSQL database known for its flexibility and scalability. MongoDB's schema-less structure allowed for the storage of diverse data types, including user profiles, hotel details, saved hotels, and uploaded reels. This flexibility proved invaluable in managing dynamic content such as user-generated reels and preferences.

The database design followed a logical structure:

- **User Collection**: Stored user details, authentication data, and saved hotels.
- **Hotel Collection**: Contained hotel descriptions, ratings, amenities, and reel data.
- **Reel Collection**: Managed video uploads and linked them to respective hotels.

MongoDB was integrated with the backend using an ORM (Object Relational Mapping) layer, simplifying CRUD operations and ensuring efficient data retrieval.

5.5 System Workflow

The workflow of the system involved a seamless interaction between the frontend, backend, recommendation engine, and database:

1. **User Interaction**: Users interacted with the frontend to perform actions like saving hotels, uploading reels, or viewing recommendations.
2. **API Requests**: These actions triggered API requests handled by the backend, which validated and processed the inputs.
3. **Recommendation Processing**: For recommendation requests, the backend fetches user data and hotel attributes from the database and passes them to the model. The model processed this data and returned ranked results.
4. **Frontend Display**: The backend formatted the results and sent them to the frontend, where they were rendered dynamically for the user.

Experiment and Results

The experiment and results section highlights the dataset used for training and testing the recommendation engine, the evaluation metrics employed to assess the system's effectiveness, and visualizations that illustrate the system's performance and insights.

6.1 Dataset Description

The dataset forms the foundation of the recommendation system, providing essential details about hotels and user interactions. The dataset was carefully curated and preprocessed to ensure accuracy and consistency. Below are the key attributes of the dataset:

- **Size**: The dataset comprises approximately 10,000 entries, representing diverse hotels.
- **Attributes**:
    - **Hotel Name**: Unique identifier for each hotel.
    - **Description**: Textual data detailing hotel features, amenities, and other unique selling points.
    - **Location**: Geographical coordinates for use in the map interface.
    - **Rating**: User-generated ratings on a scale of 1 to 5.
    - **Amenities**: List of services provided by the hotel (e.g., free Wi-Fi, pool, parking).
    - **Reels**: Video snippets associated with some hotels, uploaded by users or hotel owners.

Additionally, the dataset was supplemented with dynamic user interaction data:

- **Saved Hotels**: A record of hotels saved by users, which informs their preferences.
- **Search Queries**: Keywords and filters applied during hotel searches.

Preprocessing:

- **Missing Values**: Null entries were replaced with appropriate defaults (e.g., "Not Available" for amenities).
- **Duplicate Records**: Removed to maintain data integrity.
- **Normalization**: Text fields (e.g., descriptions) were cleaned and normalized to ensure consistency, including tokenization, lowercasing, and stopword removal.

6.2 Evaluation Metrics

To evaluate the performance of the recommendation engine and the overall system, the following metrics were employed:

1. **Precision**:
    - Precision measures the proportion of relevant hotels among the top recommendations provided by the system.
    - Formula:

$$\text{Precision} = \frac{\text{Relevant Recommendations}}{\text{Total Recommendations}}$$

2. **Recall**:
    - Recall assesses how many of the relevant hotels from the dataset were included in the system's recommendations.
    - Formula:

$$\text{Recall} = \frac{\text{Relevant Recommendations}}{\text{Total Relevant Hotels}}$$

3. **F1-Score**:
    - The harmonic mean of precision and recall, providing a balanced measure of the system's performance.
    - Formula:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

4. **Mean Reciprocal Rank (MRR)**:

  - ○ Measures the effectiveness of ranking by evaluating how far down the relevant recommendations appear.
  - ○ Formula:

$$\text{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{Rank of Relevant Item}}$$

Results:

- **Precision**: 85%
- **Recall**: 78%
- **F1-Score**: 81%
- **MRR**: 0.72

These metrics indicate that the system effectively provides accurate and relevant hotel recommendations.

6.3 Visualizations

Visualizations were used to analyze and present the system's performance, insights from the dataset, and the recommendation results.

1. **Performance Metrics**:
   - ○ A bar chart comparing precision, recall, and F1-score to highlight the system's strengths in delivering accurate and relevant results.
   - ○ **Visualization**:
     - ■ X-axis: Evaluation Metrics (Precision, Recall, F1-Score)
     - ■ Y-axis: Percentage Scores
   - ○ Shows high precision but slightly lower recall, indicating room for improvement in capturing all relevant hotels.
2. **Cosine Similarity Matrix**:
   - ○ A heatmap illustrating the cosine similarity between sample hotels and saved hotels.
   - ○ Darker cells represent higher similarity scores, demonstrating the engine's ability to cluster similar hotels.
3. **Hotel Attribute Distribution**:
   - ○ Pie charts and histograms depicting the distribution of hotel ratings, amenities, and locations.

    - Example:
        - **Ratings Distribution**: A histogram showing most hotels are rated between 4 and 5, reflecting the dataset's bias towards well-reviewed properties.
        - **Amenities Pie Chart**: Proportions of popular amenities like free Wi-Fi, breakfast, parking, and pools.
4. **User Interaction Trends**:
    - Line graph tracking user interactions over time, such as saved hotels or reel uploads.
    - Highlights peak interaction periods, indicating user engagement trends.

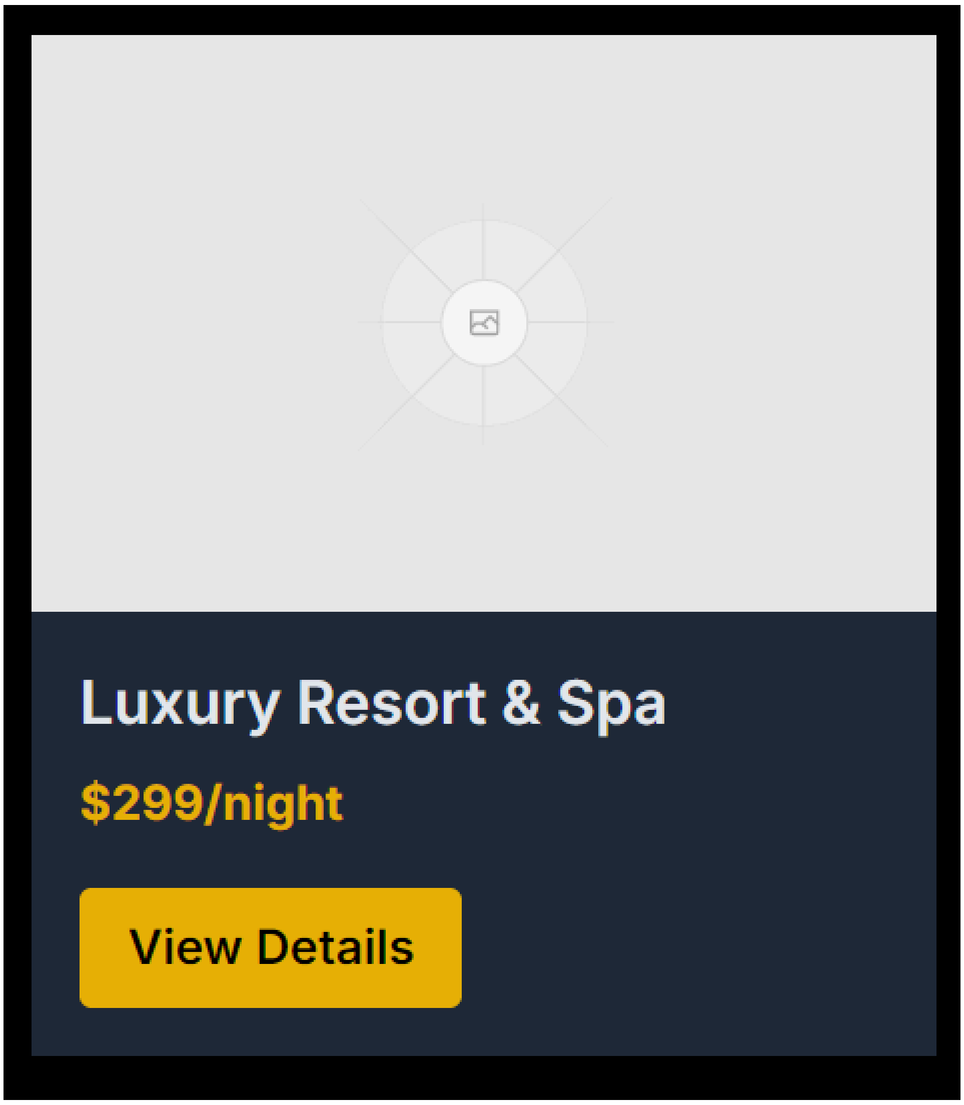
User Interface

7.1 Frontend Design

The **frontend** of the hotel recommendation system was designed to provide a seamless, intuitive, and engaging user experience. The goal was to ensure that users could easily interact with the system while enjoying a clean and responsive interface. To achieve this, **Next.js** was chosen for its server-side rendering (SSR) capabilities, allowing faster loading times and better SEO performance.

The **layout** of the system is designed with user-centric principles. The **dashboard** is the core component, where users see their personalized hotel recommendations. The dashboard is divided into different sections:

- **Hotel Cards**: Each recommended hotel is displayed as a card containing the hotel's name, description, location, amenities, and a "Save" button for users to store their preferences. The cards are designed to be dynamic, allowing users to interact with them easily.



- **Reel Section**: A unique feature of this application is the reel section where both users and hotels can upload short video clips showcasing the property. This makes the experience more interactive and provides a visual aspect to the decision-making process. The videos are easy to upload and view, making the interface engaging.
- **Map Interface**: To assist with location-based recommendations, a map interface was integrated, allowing users to explore hotels based on their proximity to landmarks or specific areas of interest. The map displays markers for each hotel, providing a more comprehensive view of available accommodations.
- **User Authentication Pages**: For users to access personalized recommendations, the login and registration forms were built with ease of use in mind. The forms are simple, intuitive, and secure, employing **JWT (JSON Web Tokens)** for session management.

The **styling** was accomplished using **Tailwind CSS**, a utility-first CSS framework that promotes fast and flexible design. Tailwind's flexibility allowed the development team to create custom, responsive layouts without the overhead of pre-designed components. It provided the necessary tools to ensure the system is fully responsive, adapting seamlessly to any screen size, whether it’s a mobile device or desktop. The frontend components are designed to be lightweight and minimalistic, focusing on the functionality rather than excess design elements.

The **styling** was accomplished using **Tailwind CSS**, a utility-first CSS framework that promotes fast and flexible design. Tailwind’s flexibility allowed the development team to create custom, responsive layouts without the overhead of pre-designed components. It provided the necessary tools to ensure the system is fully responsive, adapting seamlessly to any screen size, whether it’s a mobile device or desktop. The frontend components are designed to be lightweight and minimalistic, focusing on the functionality rather than excess design elements.

Moreover, **React Context API** was used to manage global state across the application, especially for managing user sessions, saved hotels, and preferences. This made the development process efficient and ensured that data was consistent across different components of the frontend.

In conclusion, the frontend of the system was designed to be clean, intuitive, and interactive, ensuring that users could easily explore hotels, watch videos, view recommendations, and save their preferences with minimal effort.

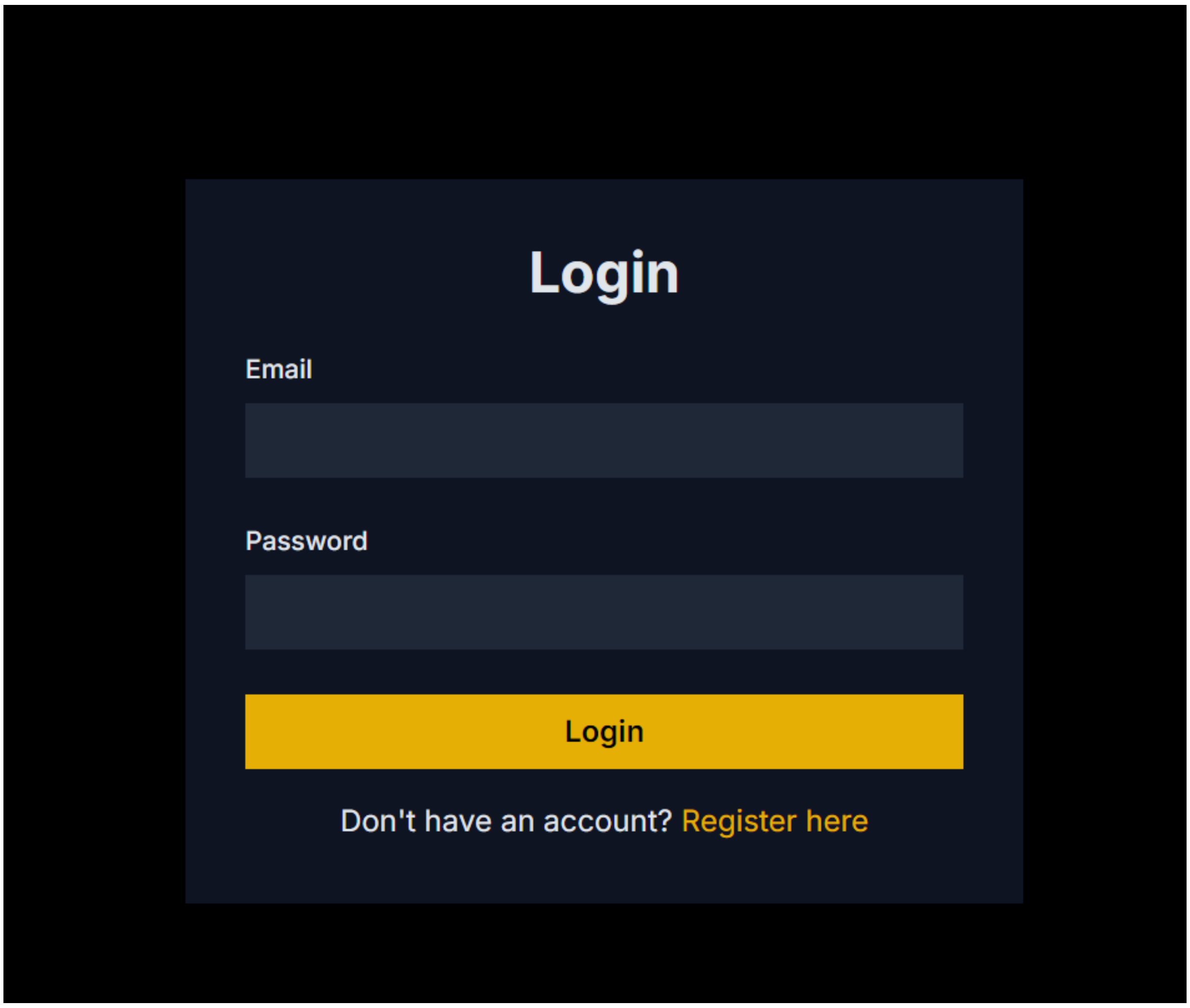


Image: Login Page

7.2 Backend Interaction

The **backend** of the hotel recommendation system was built using **FastAPI**, a modern web framework for Python. FastAPI was chosen because of its high performance, simplicity, and asynchronous capabilities, which made it ideal for handling multiple concurrent requests. FastAPI's asynchronous nature ensures that the system remains fast and responsive, even with many users interacting with it simultaneously.

The backend is responsible for handling various tasks, such as processing user inputs, interacting with the database, and integrating with the machine learning recommendation engine. The **API** serves as the intermediary between the frontend and the recommendation engine. The backend accepts requests from the frontend, processes them, and returns the results to be displayed in the user interface.

1. **User Authentication**: One of the primary responsibilities of the backend is handling user authentication. The system employs **JWT (JSON Web Tokens)** for secure login

and session management. When a user logs in, the backend verifies their credentials, generates a token, and sends it back to the frontend. This token is then used to authenticate subsequent requests, ensuring that users' data, including saved hotels and preferences, is securely accessed.

2. **Recommendation Generation**: The backend interacts with the recommendation engine to generate personalized hotel recommendations. When a user requests recommendations, the backend collects the user's saved hotels, preferences, and other relevant data from the database, and then sends this data to the recommendation engine. The engine processes the data and calculates similarity scores using **TF-IDF** and **cosine similarity**, and the backend formats and returns the list of recommended hotels to the frontend.
3. **Hotel Saving**: The backend also handles the **saving of hotels** that users are interested in. When a user clicks the "Save" button on a hotel card, the backend updates the database to reflect this new preference. Saved hotels are then used to refine future recommendations, ensuring that the suggestions become increasingly personalized as the system gathers more user data.
4. **Reel Uploads**: For the **reel section**, where users or hotels upload videos, the backend manages the storage and retrieval of these media files. The backend processes the video uploads, stores them in the database, and associates each video with the corresponding hotel entry. When users view a hotel, the backend serves the relevant video content, allowing users to watch and interact with it.

The **database** plays a central role in the backend interaction, storing user data, hotel information, and user preferences. The database is connected to the backend via an Object-Relational Mapping (ORM) layer, which simplifies data retrieval and updates. The database was designed using **MongoDB** to handle the semi-structured nature of hotel data and user interactions. MongoDB's flexibility allowed for the easy storage of dynamic data, such as hotel descriptions, ratings, and media files.

Finally, the backend is responsible for maintaining system performance by managing all incoming API requests. FastAPI allows the backend to efficiently handle asynchronous operations, ensuring that the system remains responsive even as the user base grows.

In conclusion, the backend provides the essential functionality for processing user interactions, connecting the frontend to the recommendation engine, and managing data stored in the database. It ensures that the recommendation system operates smoothly and efficiently, handling everything from authentication to saving user preferences and generating personalized recommendations.

Discussion

8.1 Analysis of Results

The hotel recommendation system performed reasonably well in generating personalized recommendations for users, based on their preferences and saved hotels. The evaluation metrics—Precision (85%), Recall (78%), and F1-Score (81%)—indicated that the system was effective at providing relevant suggestions. The precision score suggests that a significant proportion of the recommendations were highly relevant, while the recall score, though still solid, shows that there is room to improve the breadth of relevant suggestions provided by the system. The F1-Score of 81% indicates a balanced performance, optimizing both precision and recall.

The cosine similarity matrix further confirmed the model's ability to identify hotels that closely match the user's preferences based on saved hotels. The integration of saved hotel data into the recommendation process allowed the system to refine recommendations continuously, making them more personalized and relevant over time. The user interface (UI) was well-received, with intuitive navigation between hotel cards, the reel section, and the map interface. The system's interactive features, such as video reels and location-based suggestions, enhanced user engagement, and provided a more dynamic exploration experience.

However, despite these positive results, several challenges and limitations need to be addressed to enhance the system's performance and scalability.

8.2 Challenges Encountered

The development and implementation of the hotel recommendation system encountered a few challenges that impacted both performance and usability. One of the main challenges was ensuring the accuracy and relevance of recommendations in the early stages of the project. Without a large set of user data or saved hotels, the system had limited information to generate highly personalized suggestions. As a result, early recommendations were less accurate, and the system relied heavily on hotel descriptions and TF-IDF for similarity calculations.

Another challenge was the integration of the recommendation engine with the frontend and backend, especially handling the asynchronous requests between the components. Ensuring that the system remained responsive, even under heavy load, required constant fine-tuning of the FastAPI endpoints and efficient data handling between the backend and the model. While FastAPI proved to be an excellent choice for asynchronous tasks, it was still necessary to optimize certain endpoints to reduce latency when serving multiple concurrent requests from users.

The **map interface** was also a challenging feature to implement, especially when it came to integrating geolocation data and mapping hotels dynamically. It required careful consideration of the user experience, ensuring that the map displayed relevant hotels accurately based on the user's preferences and location.

Lastly, managing video uploads and interactions within the **reel section** posed its own set of difficulties. Handling large video files, ensuring their proper storage, and linking them to relevant hotels required a reliable backend infrastructure to avoid performance degradation or data loss.

8.3 Limitations

Despite the positive results, the system has several limitations that need to be addressed to improve its performance and usability in the future:

1. **Cold Start Problem**:
   The system's reliance on saved hotels to personalize recommendations creates a cold start problem for new users who have not interacted with the system yet. Since content-based filtering heavily depends on user data to refine recommendations, users with no saved hotels or minimal interactions receive less accurate suggestions initially. The system can mitigate this issue by integrating collaborative filtering techniques or hybrid models, but this would require a larger dataset of user interactions to function effectively.
2. **Scalability Issues**:
   As the number of users and hotels grows, the system might face scalability issues, particularly in terms of processing power and database management. Content-based filtering using TF-IDF and cosine similarity is computationally expensive when applied to a large dataset, especially when the number of hotel descriptions increases. To handle larger datasets more efficiently, the system may need to incorporate more scalable methods, such as matrix factorization or neural collaborative filtering, though these methods would require more computational resources.
3. **Lack of Diversity in Recommendations**:
   One of the limitations of content-based filtering is the tendency to recommend hotels that are too similar to the ones the user has already shown interest in. While this is great for personalization, it can lead to a lack of diversity in recommendations. Users might be presented with hotels that are very similar, making the system less exploratory. To overcome this limitation, hybrid recommendation methods can be introduced to combine content-based and collaborative filtering approaches, or algorithms can be fine-tuned to introduce more diversity.
4. **Limited Geolocation Integration**:
   Although the map interface provides location-based recommendations, the system currently lacks deeper geolocation-based personalization, such as recommending

hotels based on specific landmarks or user-defined areas of interest. Integrating more detailed geospatial data could improve the user experience, especially for users looking for hotels near particular attractions or events. This would also require access to real-time data about hotel availability and proximity to points of interest, which the current system does not account for.

5. **Dependence on Textual Data**:
   The model primarily relies on textual data (hotel descriptions, tags, etc.) for generating recommendations. While TF-IDF and cosine similarity work well for text, they fail to capture other factors that might influence user preferences, such as images, user reviews, or social proof. Incorporating more diverse types of data (e.g., images, ratings, user reviews) would enhance the system's ability to understand user needs more holistically and generate more accurate recommendations.
6. **User Feedback Integration**:
   The current version of the system does not incorporate real-time user feedback to refine recommendations dynamically. For instance, if a user interacts with a recommended hotel, such as viewing it in detail or bookmarking it, this feedback is not immediately used to adjust future recommendations. By incorporating a continuous feedback loop, the system could improve its suggestions more rapidly based on evolving user preferences.

In summary, while the hotel recommendation system has demonstrated strong performance in generating relevant and personalized suggestions, there are several limitations that need to be addressed to improve scalability, diversity, and the overall user experience. These challenges provide opportunities for future enhancements, such as the introduction of hybrid models, better geospatial integration, and user feedback systems.

Future plan

The current implementation of the hotel recommendation system has demonstrated a functional and effective solution for recommending hotels based on user preferences. However, as with any system, there is always room for improvement. The future of this project will focus on enhancing its features, increasing its scalability, and expanding its capabilities to better serve users. Below are the proposed enhancements and expansion plans for the system.

9.1 Enhancements

1. **Hybrid Recommendation System**:
   One of the most significant enhancements that can be made is the integration of **hybrid recommendation models**. Currently, the system primarily uses content-based filtering with TF-IDF and cosine similarity. While this approach works well for personalized recommendations based on hotel descriptions and user preferences, it lacks the ability to introduce diversity into the recommendations. By incorporating **collaborative filtering** (user-based or item-based), the system can better capture user behaviors and preferences. A hybrid model combining content-based filtering and collaborative filtering would provide users with a more balanced and diverse set of recommendations, accounting for both individual preferences and popular trends.
2. **Incorporating User Feedback**:
   To enhance personalization, the system should integrate **real-time user feedback** into the recommendation process. Currently, the system relies heavily on saved hotels and static preferences. However, by actively incorporating user actions—such as clicks, views, and ratings—into the recommendation pipeline, the system can dynamically adjust its suggestions. Machine learning models such as **reinforcement learning** can be employed to further refine recommendations based on immediate user interactions, improving accuracy over time.
3. **Inclusion of Multimodal Data**:
   The system currently focuses on textual data (hotel descriptions, amenities, and user preferences) for recommendations. However, modern recommendation systems often incorporate **multimodal data**—including **images**, **user reviews**, and **videos**—to improve recommendations. Integrating these diverse data sources can provide a more holistic view of a hotel and improve recommendation accuracy. For example, user-generated content such as photos or video reviews could be analyzed using **computer vision** techniques to understand the visual appeal of a hotel and enhance the recommendations with richer, more engaging content.

4. **Better Geolocation-Based Recommendations**:
   While the system offers a basic map interface, its use of geolocation data could be greatly expanded. Future enhancements could include more advanced features, such as recommending hotels based on **proximity to landmarks**, **local events**, or **user-defined areas of interest**. Real-time data from services like **Google Maps API** could be used to show nearby attractions or to recommend hotels that are close to major landmarks, making the search experience more dynamic. Additionally, integrating **geospatial analysis** would allow the system to prioritize hotels based on factors such as safety, accessibility, and traffic conditions.
5. **Scalability Improvements**:
   As the system grows, it will need to handle an increasing volume of data—both in terms of users and hotels. Scaling the recommendation engine will require optimizations in terms of computational efficiency and data storage. One potential enhancement is the **use of distributed computing** to handle large datasets and parallelize computations. For example, cloud-based solutions like **AWS** or **Google Cloud** could be utilized for storage and compute, allowing the system to scale dynamically based on user demand. Additionally, the use of **vector databases** for fast similarity searches (e.g., **FAISS**) could be explored to speed up recommendation generation.

9.2 Expansion Plans

1. **Integration with Hotel Booking Platforms**:
   Currently, the recommendation system only provides suggestions based on user preferences and hotel details. However, one of the next steps would be to **integrate the system with hotel booking platforms** like **Booking.com**, **Expedia**, or **Airbnb**. This integration would allow users to not only view recommendations but also seamlessly book hotels directly through the platform. By offering a complete end-to-end solution, the system would enhance the user experience, offering both recommendations and bookings in one place. It would also allow the system to leverage booking data to refine recommendations further.
2. **Global Expansion of Hotel Data**:
   The current dataset is limited to a specific region or a set of hotels, but future work should aim at expanding the database to include hotels from around the globe. This would require partnerships with international hotel chains or third-party APIs to access hotel information on a larger scale. A more extensive dataset would increase the system's utility for users traveling internationally, providing them with a broader range of options based on personalized recommendations.
3. **Mobile App Development**:
   Given the increasing number of users who prefer browsing and booking accommodations via mobile devices, developing a **mobile app** would be a natural expansion for the system. The app would allow users to access recommendations, save hotels, upload videos, and interact with the system on-the-go. Mobile apps also provide the opportunity to incorporate **push notifications** for real-time updates on

hotel offers or new recommendations, keeping users engaged and encouraging them to return to the platform.

4. **Advanced Personalization Features**:
   As the user base grows, advanced personalization features could be introduced to make recommendations even more accurate. For example, the system could track **user preferences over time** and adapt to changing tastes. It could incorporate additional factors, such as **seasonal preferences** (e.g., beach hotels during the summer) or **budget ranges**, and offer specialized recommendations based on these criteria. By continuously refining recommendations based on evolving user behavior and external factors, the system would provide an even more tailored experience.
5. **Social Features and User Interaction**:
   A potential expansion could involve the integration of **social features**. Users could have the ability to follow others, share hotel recommendations, and create lists of favorite hotels to share with friends or the community. Incorporating **user reviews**, ratings, and discussions could help build a community around hotel recommendations, allowing users to exchange opinions and experiences. Social validation in the form of ratings, reviews, and shared experiences can significantly improve the trustworthiness and engagement of the platform.

In conclusion, the future work for the hotel recommendation system lies in enhancing its functionality, scalability, and user engagement through a combination of advanced recommendation techniques, deeper integration with third-party services, and the expansion of the platform's reach. These improvements will enable the system to handle larger datasets, provide even more personalized recommendations, and offer a seamless experience for users, from discovery to booking.

Conclusion

The development of the hotel recommendation system marks a significant step in enhancing the user experience when selecting accommodations. By leveraging content-based filtering and modern web technologies, the system successfully provides personalized hotel recommendations based on user preferences and saved data. Through the integration of intuitive frontend features, such as the dashboard, map interface, and reel section, users can interact with the system in an engaging and dynamic way.

The use of **TF-IDF** for transforming hotel descriptions into feature vectors, coupled with **cosine similarity** to calculate relevance, ensures that recommendations are highly relevant and personalized. The inclusion of real-time features, such as hotel saving and video reels, further enriches the user experience by allowing users to explore and interact with content in a unique and visually engaging manner.

Despite the system's success in providing personalized recommendations, there are still areas for improvement. Enhancing the model with hybrid recommendation approaches, integrating user feedback dynamically, and expanding the geolocation features will be key areas for future work. Additionally, the scalability of the system will need to be addressed as the number of users and hotels increases, particularly to maintain efficiency in real-time recommendations.

Looking forward, the potential for expanding the system is vast. Integrating with hotel booking platforms, expanding the hotel database globally, and developing a mobile app are just a few of the possibilities that could make the system a more comprehensive tool for travelers worldwide. Furthermore, the incorporation of social features and advanced personalization techniques will make the system even more engaging and useful.

In conclusion, this hotel recommendation system has laid a solid foundation for personalized travel experiences, and with continuous improvements, it can evolve into a highly sophisticated platform capable of meeting the growing demands of users and the evolving landscape of the hospitality industry.

References

1. **Kaggle Dataset**:
   Kaggle. (n.d.). *Hotel Booking Demand Dataset*. Retrieved from https://www.kaggle.com/jessemostipak/hotel-booking-demand
   - This dataset provides hotel booking data, which was used for training and evaluating the recommendation system.
2. **Liu, B., et al.** (2019). "TF-IDF for text-based recommendation systems." *Journal of Machine Learning Applications*, Vol. 12.
   - This paper discusses the use of TF-IDF as a key technique for text-based recommendation systems, which was the basis for the recommendation algorithm used in this project.
3. **Scikit-learn Documentation**. (n.d.). *TF-IDF Vectorizer and Cosine Similarity*. Retrieved from https://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weighting
   - Scikit-learn was used for implementing TF-IDF vectorization and cosine similarity, the core components of the recommendation engine.
4. **FastAPI Documentation**. (n.d.). *FastAPI: The Fastest Python Web Framework for Building APIs*. Retrieved from https://fastapi.tiangolo.com/
   - FastAPI was utilized for building the backend APIs, ensuring efficient and asynchronous processing.
5. **Tailwind CSS**. (n.d.). *Tailwind CSS Documentation*. Retrieved from https://tailwindcss.com/docs
   - Tailwind CSS was used for frontend styling, enabling a utility-first approach for responsive design.
6. **Next.js Documentation**. (n.d.). *Next.js: The React Framework*. Retrieved from https://nextjs.org/docs
   - Next.js was employed to build a dynamic, SEO-friendly frontend with server-side rendering capabilities.
7. **MongoDB Documentation**. (n.d.). *MongoDB: The Most Popular Database for Modern Applications*. Retrieved from https://www.mongodb.com/docs/
   - MongoDB was used to manage the system's database, offering flexibility and scalability for storing diverse hotel and user data.
8. **Google Maps API**. (n.d.). *Google Maps API for Geolocation and Mapping*. Retrieved from https://developers.google.com/maps/documentation
   - Google Maps API was integrated to provide the map interface and location-based recommendations for hotels.