Dreamscape Destinations

A Journey from your dreams to reality

1. Group Name: SE PROJECT TEAM 02

2. Group Members:

- Veeresh Sakali
- Susmith Meesa
- Harshath Budida
- Harshitha Thokala
- Gopi Krishna Kummari
- Pooja Sree Poka
- Sashidhar Chary Viswanathula
- Balaji Valeti

Deliverable 4 – Project Phase 2
CSCE 5430 (Spring 2024)

Implementation Phase 2:

This is 2nd phase of real time implementation of the code. As we have discussed in deliverable 3 that initially our target was to generate all front end in the first phase and back end in the second phase of implementation, but as per the TA suggestions and peers review we have changed our plan and decided to go hands in hand with both the front end which is on React JS and the back end which is on Nodes JS.

Requirements:

1. Change of plan:

Initially we thought of making the front end for phase 1, back end for the phase 2 and optimization for the phase 3, but after our requirements submission we got peer feedback and teacher feedback which led us to alter our plan a little bit. So, now our target is to make front end along with the back end and database of functional requirements and that complete the remaining set of requirements in the phase 2 and phase 3 is still allocated for the web optimization, code optimization and SEO. The 2nd biggest change is requirement is the change of our database. In initial plan we planned to use the SQLite3 database but looking at our back end which is purely in the Nodes.JS, we find out that the best fit database with the Nodes is MongoDB. We have also conducted a thorough search on the database compatibility and found that, as Nodes.JS is a browser language and MongoDB is also a browser-based database so they best fit with each other (Vohra, D. (2015). Using MongoDB with Node.js. In: Pro MongoDB™ Development. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-1598-2_5). So, we have setted up a MongoDB account and created a cluster for our project.

2. Front end:

- **Blogs/Feedback reactions:**

In the frontend development of DreamEscape's blog feature, we implemented a dynamic and interactive user interface using React. Our primary focus was on creating a seamless user experience by dynamically rendering blog posts fetched from the backend server. Leveraging React's useQuery hook from Apollo Client, we established real-time data fetching, ensuring that users always have access to the latest blog content without needing to refresh the page.

One significant achievement was the integration of user authentication functionality, enhancing the platform's security and user experience. Authenticated users are granted access to additional features, such as creating new blog posts. We implemented conditional rendering to display the post creation form exclusively to authenticated users, guiding them through the content creation process seamlessly. Our team also prioritized the development of an interactive user interface, enabling users to engage with blog posts effortlessly. The PostForm component facilitated the creation of new posts, providing users with a straightforward form submission process. Additionally, we integrated features like post liking and deletion (for post authors) to foster user engagement and interaction within the blog community.

To ensure a consistent and pleasant user experience across various devices and screen sizes, we adopted responsive design principles. Utilizing Tailwind CSS utility classes, we crafted a visually appealing and mobile-friendly layout for the blog page, optimizing readability and usability on both desktop and mobile devices. Furthermore, robust error handling mechanisms were implemented to provide informative feedback to users in case of failed operations. By utilizing try-catch blocks and error logging, we ensured graceful handling of exceptions, preventing application crashes and enhancing overall reliability.

- **Profile Page:**

The recent update to DreamEscape's blogging platform introduces several enhancements to the user profile page, enriching the user experience and providing additional functionality for users to interact with their content and customize their profiles. One of the standout features of this update is the integration of user posts directly onto the profile page. Now, users can conveniently view all their published blog posts within a dedicated section of their profile. This feature enhances user engagement by allowing individuals to showcase their contributions to the platform and easily access their authored content in one centralized location.

Furthermore, users now have visibility into the engagement metrics of their blog posts, including likes and comments. By displaying this information alongside each post on the profile page, users gain valuable insights into the reception and interaction with their content. This fosters a sense of connection and feedback loop between content creators and their audience, encouraging continued content creation and community engagement.

In addition to showcasing posts and engagement metrics, the profile page now offers enhanced customization options for users. With the ability to update profile pictures, location, and bio directly from the profile page, individuals can personalize their profiles to better reflect their identity and interests. This empowers users to curate their online presence and establish a unique persona within the blogging community.

The inclusion of these features not only enhances the functionality of the profile page but also contributes to a more immersive and interactive user experience across the DreamEscape platform. By providing users with comprehensive tools to manage their content, engage with their audience, and personalize their profiles, DreamEscape reinforces its commitment to fostering a vibrant and engaging blogging community. These updates represent a significant step forward in empowering users to express themselves creatively, connect with like-minded individuals, and cultivate meaningful relationships within the platform.

- **Booking:**

The latest update to DreamEscape's booking feature introduces a significant enhancement by integrating the "Travel Advisor" API from RapidAPI. This API provides access to a vast repository of travel destinations, allowing users to discover and explore the latest travel options tailored to their preferences and requirements. With this integration, DreamEscape empowers users to access up-to-date information on a diverse range of travel places, enhancing their ability to plan and customize their travel experiences seamlessly.

The "Travel Advisor" API offers comprehensive search functionality based on various parameters, including travel type, pickup place, destination place, start date, end

date, and the number of passengers. This enables users to specify their travel criteria with precision, ensuring that the recommendations provided align closely with their preferences and needs. By leveraging the extensive database of travel destinations available through the API, DreamEscape enables users to access a wealth of options and make informed decisions when planning their journeys.

Upon initiating a search, DreamEscape presents users with a curated list of available travel options that match their specified criteria. This includes a diverse array of destinations, accommodations, transportation options, and activities, allowing users to explore various possibilities and tailor their travel plans according to their preferences. The intuitive interface empowers users to browse through the available options effortlessly, facilitating an immersive and personalized booking experience.

Furthermore, DreamEscape enables users to customize their travel itineraries based on the recommendations provided by the "Travel Advisor" API. Whether users are seeking a relaxing beach getaway, an adventurous mountain expedition, or a cultural city tour, they can easily select and customize their preferred travel options to create a personalized itinerary that suits their interests and preferences.

By integrating the "Travel Advisor" API into its booking feature, DreamEscape reaffirms its commitment to providing users with comprehensive and tailored travel solutions. This update expands the platform's capabilities, enabling users to access a wealth of travel information, discover new destinations, and plan their journeys with ease. With DreamEscape, users can embark on unforgettable travel experiences, confident in the knowledge that they have access to the latest and most relevant travel options available.

- **Detail Page for places:**

The addition of a dedicated page showcasing top-of-the-line travel destinations along with their unique features and offerings. This new page serves as a comprehensive resource for users seeking detailed information about premium travel destinations, allowing them to explore and discover exciting travel opportunities from the comfort of their own homes. With this enhancement, DreamEscape aims to provide users with curated insights into the most coveted travel destinations, facilitating informed decision-making and seamless booking experiences.

The newly added destination detail page offers users a rich and immersive experience, presenting detailed information about each featured destination in a visually compelling format. Users can delve into the unique features, attractions, accommodations, and activities offered by each destination, gaining valuable insights to help them envision their dream vacation. By showcasing top-of-the-line destinations, DreamEscape caters to discerning travelers seeking unparalleled luxury and exclusivity in their travel experiences.

In addition to providing comprehensive destination information, the new page also offers users the convenience of booking their desired destination directly from the platform. By integrating booking functionality seamlessly into the destination detail page, DreamEscape streamlines the booking process, allowing users to secure their travel arrangements with ease and efficiency. Whether users are drawn to luxurious resorts, exotic locales, or cultural landmarks, they can book their desired destination with confidence, knowing that they are embarking on a premium travel experience curated by DreamEscape.

Furthermore, the destination detail page is designed to inspire and captivate users, enticing them to explore and discover new travel opportunities. Through stunning imagery,

engaging descriptions, and immersive multimedia content, DreamEscape creates an immersive browsing experience that transports users to their desired destinations virtually. This immersive approach not only informs users about the features and attractions of each destination but also ignites their wanderlust and fuels their desire to embark on unforgettable travel adventures.

- **Contact Us form:**

We have also added a contact us form as per the peer review. The contact us form is become necessary of time as it gives the customers liberty to ask any sort of questions and it will help them to clear all the confusions.

3. Back end:

- **Blogs/Feedback:**

In the backend development of DreamEscape's blog feature, we focused on designing a robust GraphQL schema to define the data structure and operations supported by the server. Through meticulous schema design, we ensured efficient data retrieval and manipulation, laying the foundation for seamless integration with the frontend.

A key achievement was the successful integration of a database management system (DBMS) to persistently store and retrieve blog post data. Leveraging MongoDB as the backend database, we established a scalable and reliable storage solution, accommodating the platform's growing data requirements effectively. To secure the backend API endpoints and protect sensitive operations, we implemented authentication and authorization mechanisms. User authentication was facilitated using JSON Web Tokens (JWT), verifying user identity, and controlling access to privileged operations like post creation and deletion.

Furthermore, we prioritized data validation and sanitization to maintain data integrity and mitigate security risks. By implementing input validation and sanitization techniques, we guarded against common vulnerabilities such as SQL injection and cross-site scripting (XSS) attacks, ensuring the integrity and security of user-generated content. Our team also optimized query performance to enhance the scalability and responsiveness of the backend server. Employing indexing strategies and query optimization techniques, we minimized query execution time, improving overall system efficiency, and user experience, especially during peak usage periods.

Lastly, robust error handling and logging mechanisms were implemented to monitor server-side errors and exceptions actively. By recording diagnostic information and tracking application behavior, we facilitated efficient troubleshooting and debugging efforts, ensuring the reliability and stability of the backend infrastructure.

- **Queries:**

The QUERY_POSTS GraphQL query serves a pivotal role in fetching a list of blog posts from the backend server. This query is essential for displaying a collection of posts on the blog page, allowing users to explore recent content. By optionally accepting a username parameter, it enables filtering posts based on specific authors, enriching the browsing experience. The retrieved post data includes crucial information such as post title,

content, creation date, author details, comment count, and likes. Furthermore, by fetching comments associated with each post, QUERY_POSTS facilitates comprehensive post interactions, fostering community engagement and discussion on the platform.

As a counterpart to QUERY_POSTS, the QUERY_POST GraphQL query focuses on retrieving a single blog post based on its unique identifier (ID). This query enables users to access detailed information about a specific post, including its title, content, creation date, author details, comment count, and likes. By providing a means to fetch comments associated with the post, QUERY_POST empowers users to delve deeper into the content, read discussions, and contribute their insights. This query enhances the user experience by facilitating seamless navigation between posts and promoting meaningful interactions within the blogging community.

The QUERY_USER GraphQL query is instrumental in fetching detailed information about a user profile based on a provided username. This query enables users to explore and connect with other members of the blogging community by retrieving essential user attributes such as user ID, email, location, description, avatar, friend count, and a list of friends. Additionally, QUERY_USER fetches posts authored by the user, along with associated comments and likes, providing valuable insights into the user's contributions and interactions within the platform. By facilitating user discovery and engagement, this query contributes to building a vibrant and interconnected blogging community on DreamEscape.

QUERY_ME represents a crucial GraphQL query tailored to the currently authenticated user's perspective. By fetching the authenticated user's details, including user ID, username, email, location, description, avatar, friend count, and list of friends, this query personalizes the user experience and facilitates self-discovery within the platform. Furthermore, QUERY_ME retrieves posts authored by the user, along with associated comments and likes, empowering users to manage their content and engage with their audience effectively. With its focus on the user's identity and contributions, QUERY_ME fosters a sense of ownership and belonging, enhancing user retention and satisfaction on DreamEscape.

Designed to retrieve basic details of the currently authenticated user, QUERY_ME_BASIC offers a streamlined view of the user's profile information. By fetching essential attributes such as user ID, username, email, location, description, avatar, friend count, and a simplified list of friends, this query optimizes data retrieval for use cases where comprehensive user details are not immediately required. QUERY_ME_BASIC provides a lightweight solution for accessing user information, ensuring efficient performance, and minimizing unnecessary data transfer. This query enhances the overall responsiveness and usability of DreamEscape's blogging platform, contributing to a seamless and enjoyable user experience.

- **Mutations:**

Mutations collectively form the backbone of DreamEscape's blogging platform, enabling essential user interactions such as authentication, registration, profile management, social networking, content creation, interaction with posts, and engagement with the blogging community. By leveraging GraphQL mutations, DreamEscape delivers a seamless and feature-rich blogging experience, fostering user satisfaction and platform growth.

The LOGIN_USER mutation facilitates user authentication by verifying the provided email and password against the stored credentials in the backend system. Upon successful authentication, it returns an authentication token and essential user details, such as the user ID and username. This token is crucial for subsequent authenticated requests, enabling secure access to protected resources on the platform.

The ADD_USER mutation is responsible for registering new users on the DreamEscape platform. It accepts user registration details such as username, email, password, location, and description. Upon successful registration, it returns an authentication token along with detailed user information, including the user ID, username, email, location, description, and avatar. This mutation plays a vital role in expanding the platform's user base and facilitating user engagement.

The UPDATE_USER mutation allows users to update their profile information, such as avatar, location, and description. It accepts an input object containing the updated user details and the user's ID. Upon successful update, it returns an authentication token along with the updated user information. This mutation empowers users to customize their profiles and maintain accurate and up-to-date information on the platform.

The ADD_FRIEND mutation enables users to add other users as friends within the blogging community. It accepts the ID of the user to be added as a friend. Upon successful addition, it returns detailed information about the user, including the user ID, username, friend count, and a list of friends. This mutation fosters social connections and networking among users, enhancing the sense of community on DreamEscape.

Conversely, the REMOVE_FRIEND mutation allows users to remove friends from their social network. It accepts the ID of the friend to be removed. Upon successful removal, it returns updated information about the user, including the user ID, username, friend count, and an updated list of friends. This mutation provides users with control over their social connections and helps maintain a curated network of meaningful relationships.

The ADD_POST mutation facilitates the creation of new blog posts by users. It accepts the title and content of the post as input parameters. Upon successful creation, it returns detailed information about the newly created post, including the post ID, title, content, creation date, author details, comment count, and likes. This mutation empowers users to share their thoughts, experiences, and insights with the blogging community, enriching the platform's content and engaging users.

The DELETE_POST mutation allows users to delete their own blog posts from the platform. It accepts the ID of the post to be deleted. Upon successful deletion, it returns a confirmation message. This mutation provides users with control over their published content and helps maintain the quality and relevance of posts on DreamEscape.

The ADD_COMMENT mutation enables users to add comments to existing blog posts. It accepts the ID of the post to which the comment will be added and the content of the comment. Upon successful addition, it returns updated information about the post, including the updated comment count and a list of comments with detailed information. This mutation encourages user engagement and fosters discussions around blog posts, enhancing the platform's interactivity and community spirit.

The LIKE_POST mutation allows users to like or favorite blog posts. It accepts the ID of the post to be liked. Upon successful liking, it returns updated information about the post, including the updated like count and a list of users who liked the post. This mutation

enables users to express appreciation for posts they find interesting or valuable, providing feedback to content creators and encouraging further engagement within the community.

- **Authentication:**

The AuthService class serves as a critical component in managing user authentication within DreamEscape's blogging platform, encapsulating a suite of methods to handle authentication-related operations seamlessly. At the heart of this class lies the getProfile() method, designed to retrieve and decode user profile data stored within the authentication token. Leveraging the jwt-decode library, this method facilitates the extraction of essential user information, including usernames, user IDs, and any additional data embedded within the token payload.

One of the core functionalities offered by the AuthService class is the ability to determine whether a user is currently logged in. This functionality is achieved through the loggedIn() method, which verifies the presence of a valid authentication token in the local storage. By performing a dual check to ensure the existence of a token and its non-expiration, this method effectively evaluates the user's authentication status, returning true if the user is logged in and false otherwise.

Additionally, the isTokenExpired(token) method plays a crucial role in assessing the validity of authentication tokens. By parsing the token payload to extract the token expiration time (exp), this method compares it with the current time to ascertain whether the token has expired. Robust error handling mechanisms are implemented to gracefully manage decoding errors or invalid tokens, ensuring reliable token validation.

The getToken() method serves as a gateway for retrieving the user's authentication token stored in the local storage under the id_token key. This method facilitates seamless token retrieval for subsequent authentication and authorization processes, enabling secure access to protected resources within the blogging platform.

Upon successful authentication, the login(idToken) method stores the provided authentication token in the local storage, paving the way for authenticated user sessions. Subsequently, it redirects the user to the blog page (/blog), granting access to authenticated content and features seamlessly.

Conversely, the logout() method offers a robust mechanism for logging users out of their sessions. By clearing the authentication token and associated profile data from the local storage, this method effectively terminates the user's session, ensuring secure logout procedures. Following token removal, the method reloads the page to the root URL (/), resetting the application state and prompting users to log in anew if required.

Through the centralized management of authentication logic provided by the AuthService class, DreamEscape's blogging platform ensures a streamlined and secure authentication experience for users. This modular approach enhances code maintainability, promotes adherence to security best practices, and fosters a seamless user experience throughout the authentication lifecycle.

- **Contact Us:**

We have built contact us form on top of a 3rd party website which is “https://formspree.io/”. This website can generate, tracking and answering of the queries forms and it also has a feature of sending the emails. So, as soon as customer send us a form we can answer to that form via their provided email.

4. Database:

The MongoDB database connection established using Mongoose serves as the persistent data storage solution for our project. MongoDB is a NoSQL database that offers flexibility and scalability, making it well-suited for storing unstructured or semi-structured data. In our project, MongoDB allows us to efficiently manage various types of data, such as user profiles, blog posts, travel destinations, and booking information.

We opted for MongoDB due to its schema-less nature, which accommodates the evolving nature of our project's data requirements. This flexibility enables us to store and retrieve data without strict schema constraints, facilitating rapid development and iteration. Additionally, MongoDB's support for nested documents and arrays aligns well with the hierarchical structure of our project's data, such as user profiles with associated posts or booking details.

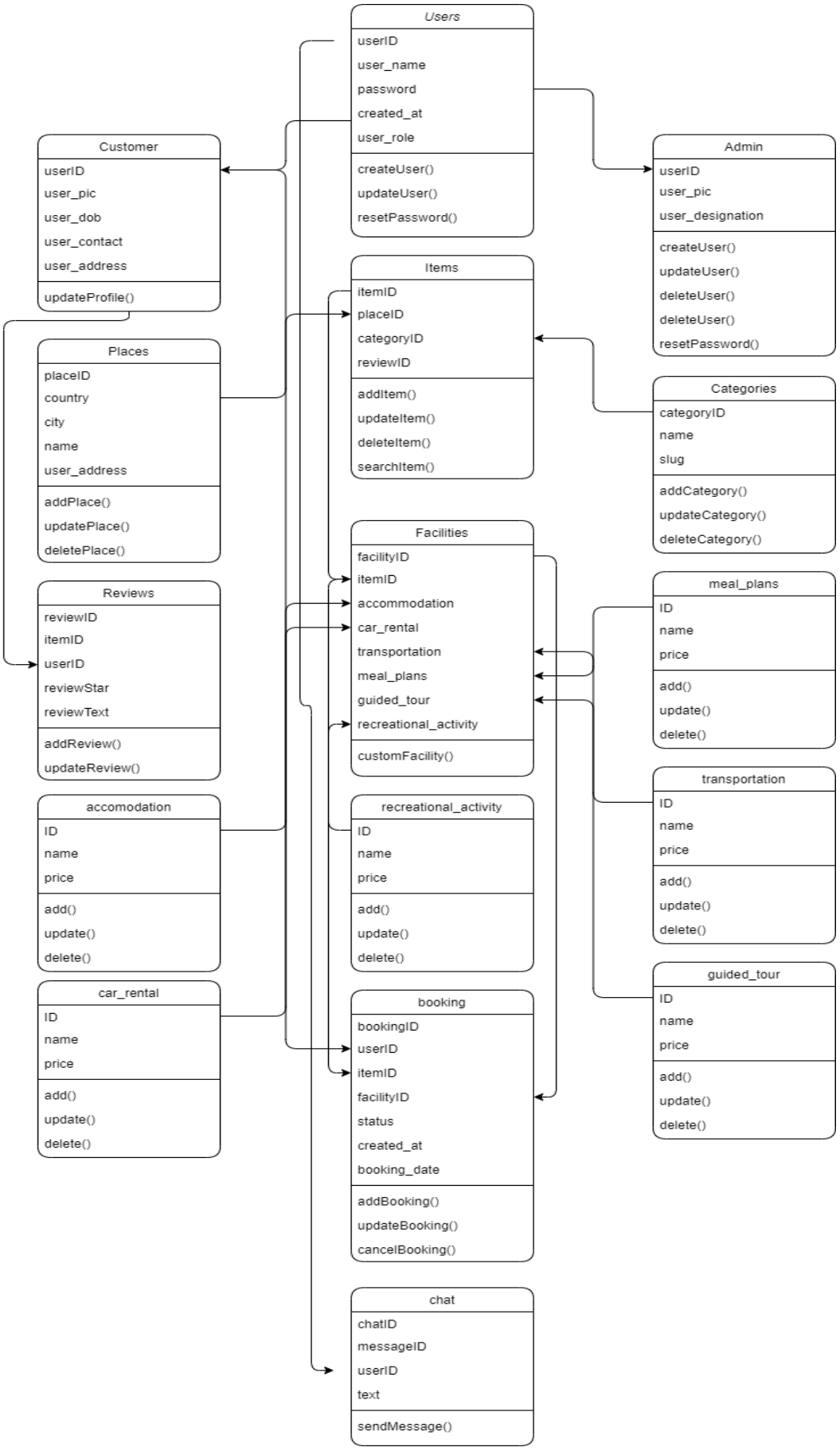
By using Mongoose, a MongoDB object modeling tool, we enhance our development workflow by providing schema validation, data mapping, and other utilities that streamline database interactions within our Node.js application. Through Mongoose, we can define schemas that enforce data consistency and integrity while leveraging MongoDB's scalability and performance benefits. Overall, MongoDB, coupled with Mongoose, empowers our project with a robust and scalable database solution tailored to our dynamic data storage needs.

UML Diagrams:

This phase most of the part was on front end but we have created whole UML:

1. Class Diagram:

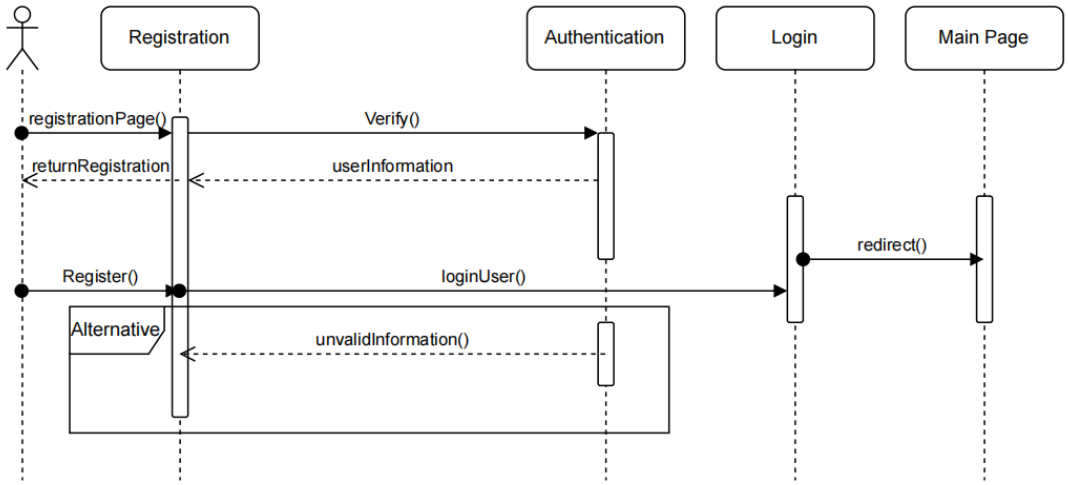
The class diagram as shown below, a travel booking system with users, places, items, reviews, bookings, and chat functionalities. Users can create profiles, update information, and book items. Places can be added, updated, or deleted with reviews. Items can be reviewed and booked. Bookings can be added, updated, or cancelled. There is also a chat function.


Users
userID
user_name
password
created_at
user_role
createUser()
updateUser()
resetPassword()
Customer
userID
user_pic
user_dob
user_contact
user_address
updateProfile()
Admin
userID
user_pic
user_designation
createUser()
updateUser()
deleteUser()
deleteUser()
resetPassword()
Items
itemID
placeID
categoryID
reviewID
addItem()
updateItem()
deleteItem()
searchItem()
Places
placeID
country
city
name
user_address
addPlace()
updatePlace()
deletePlace()
Categories
categoryID
name
slug
addCategory()
updateCategory()
deleteCategory()
Facilities
facilityID
itemID
accommodation
car_rental
transportation
meal_plans
guided_tour
recreational_activity
customFacility()
Reviews
reviewID
itemID
userID
reviewStar
reviewText
addReview()
updateReview()
meal_plans
ID
name
price
add()
update()
delete()
accomodation
ID
name
price
add()
update()
delete()
recreational_activity
ID
name
price
add()
update()
delete()
transportation
ID
name
price
add()
update()
delete()
car_rental
ID
name
price
add()
update()
delete()
booking
bookingID
userID
itemID
facilityID
status
created_at
booking_date
addBooking()
updateBooking()
cancelBooking()
guided_tour
ID
name
price
add()
update()
delete()
chat
chatID
messageID
userID
text
sendMessage()

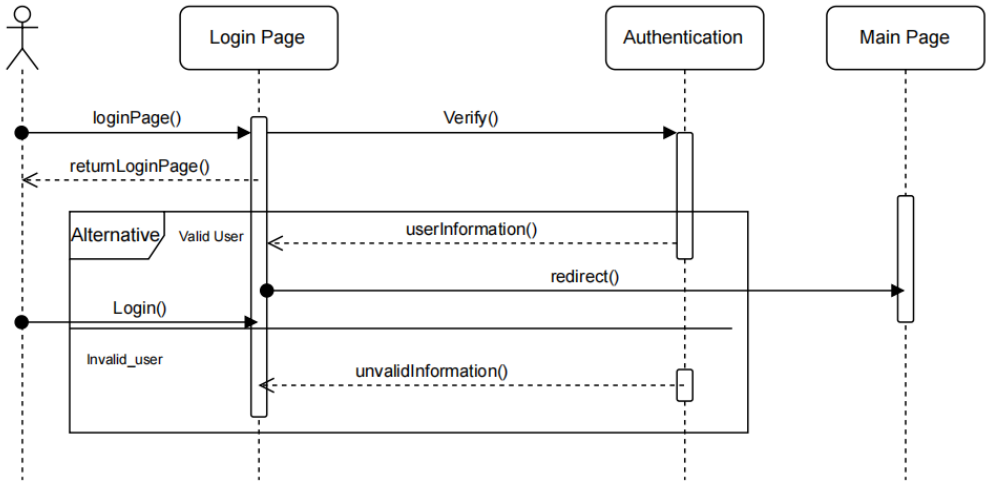
2. Sequence Diagram:

Some Sequence diagrams are from deliverable 3 as these functions are updated as the back-end development in this phase.

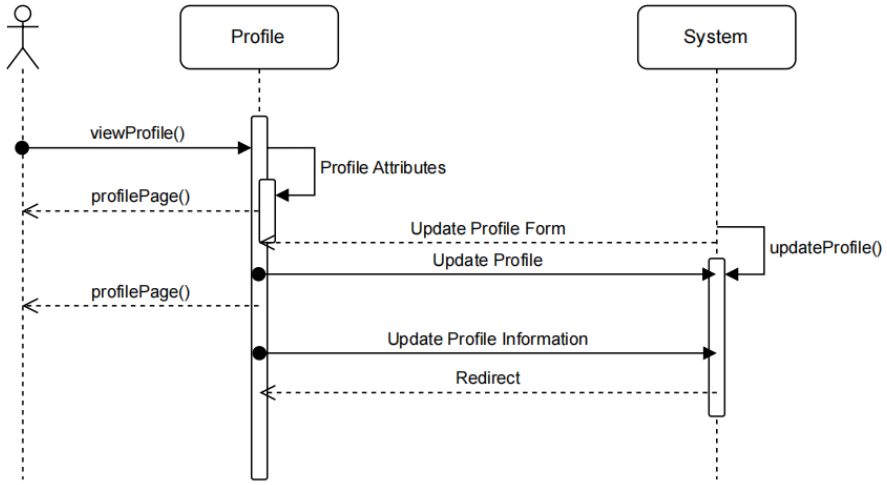
- **Sign Up:**



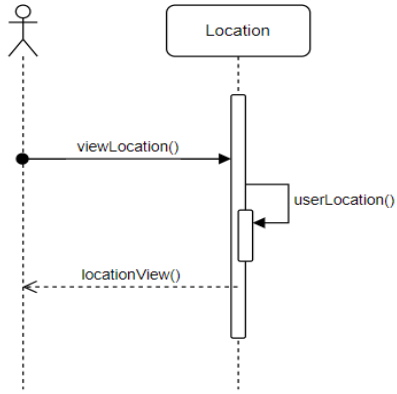
- **Login:**



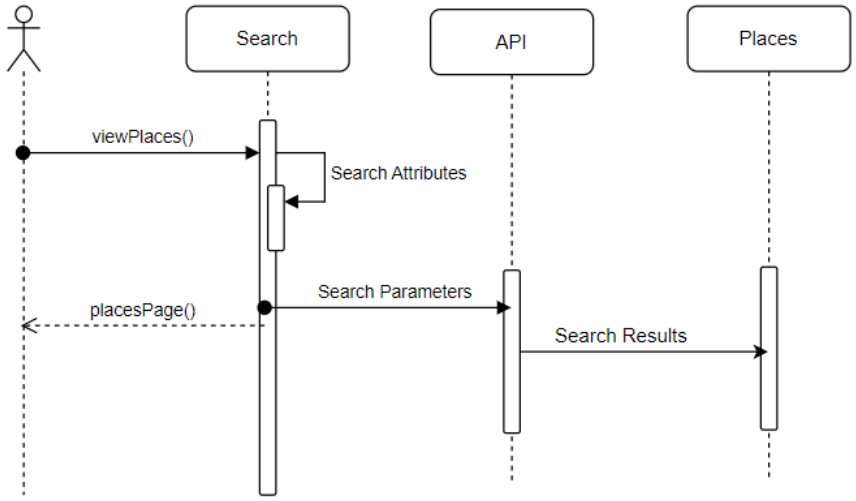
- **Update Profile:**



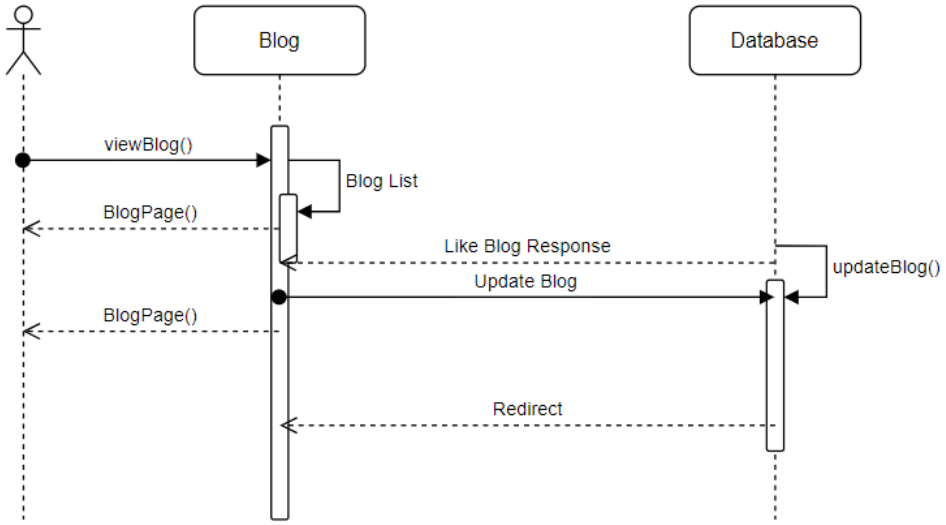
- **User Location:**



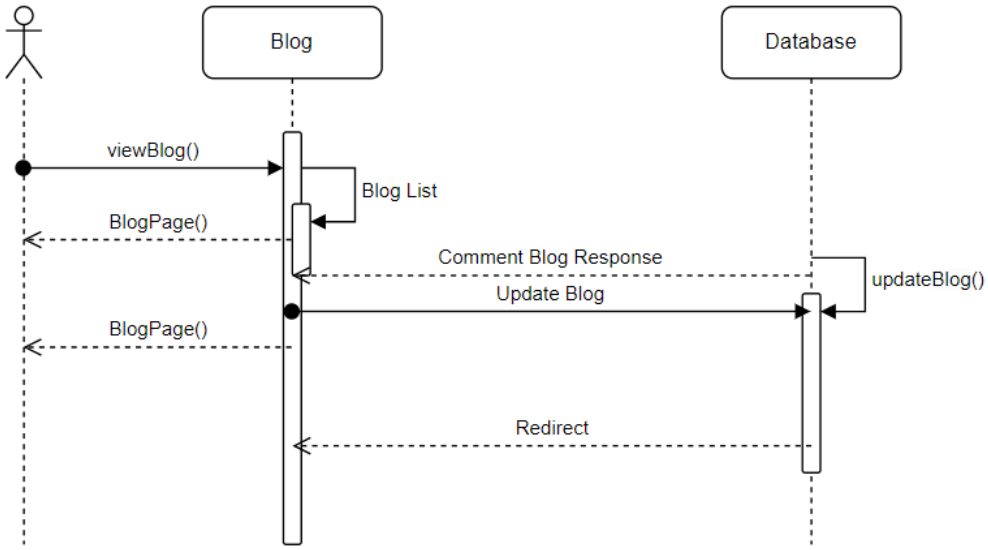
- **Search Nearby Place:**



- **Blog Like:**

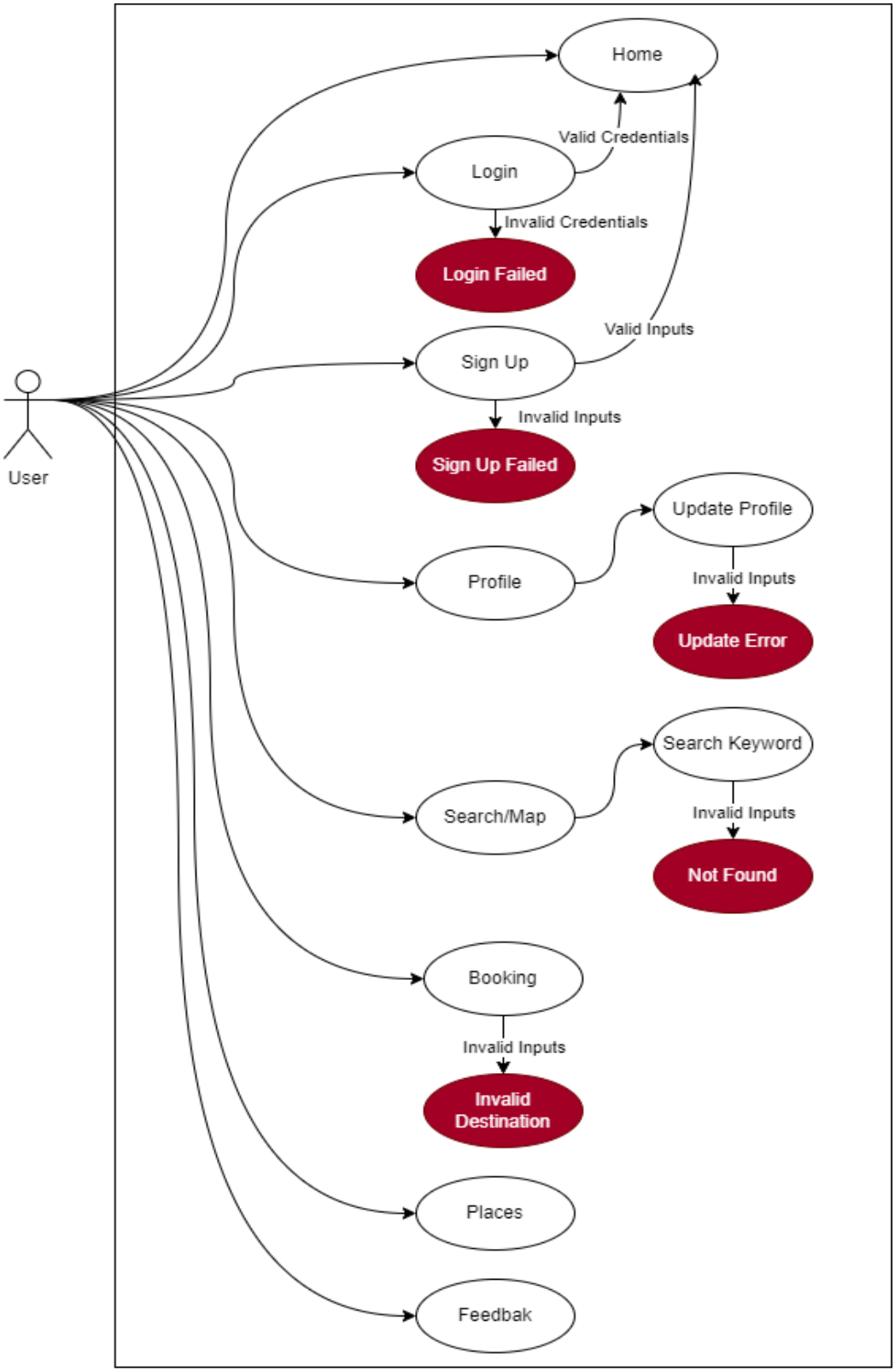


- **Blog Comment:**



3. Use Case Diagram:

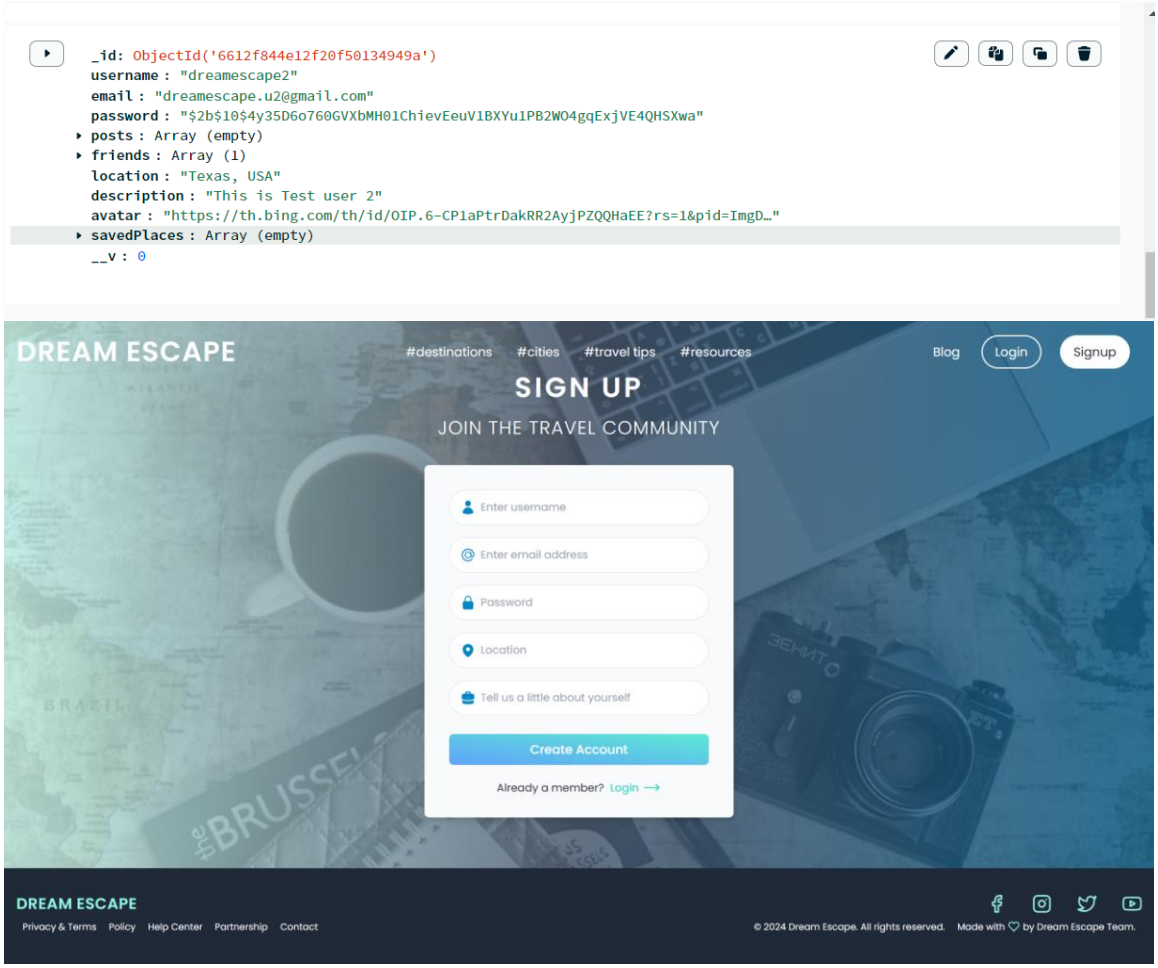
This Use case diagram shown below illustrates that a user can perform actions such as logging in, searching, and updating their profile. Login can be successful or unsuccessful. Upon successful login, the user can search using keywords or update their profile. Update profile can also be successful or unsuccessful. If the user enters invalid inputs during search or update profile, the system will display an error message.


Home
Valid Credentials
Login
Invalid Credentials
Login Failed
Valid Inputs
Sign Up
Invalid Inputs
Sign Up Failed
User
Update Profile
Profile
Invalid Inputs
Update Error
Search Keyword
Search/Map
Invalid Inputs
Not Found
Booking
Invalid Inputs
Invalid Destination
Places
Feedbak

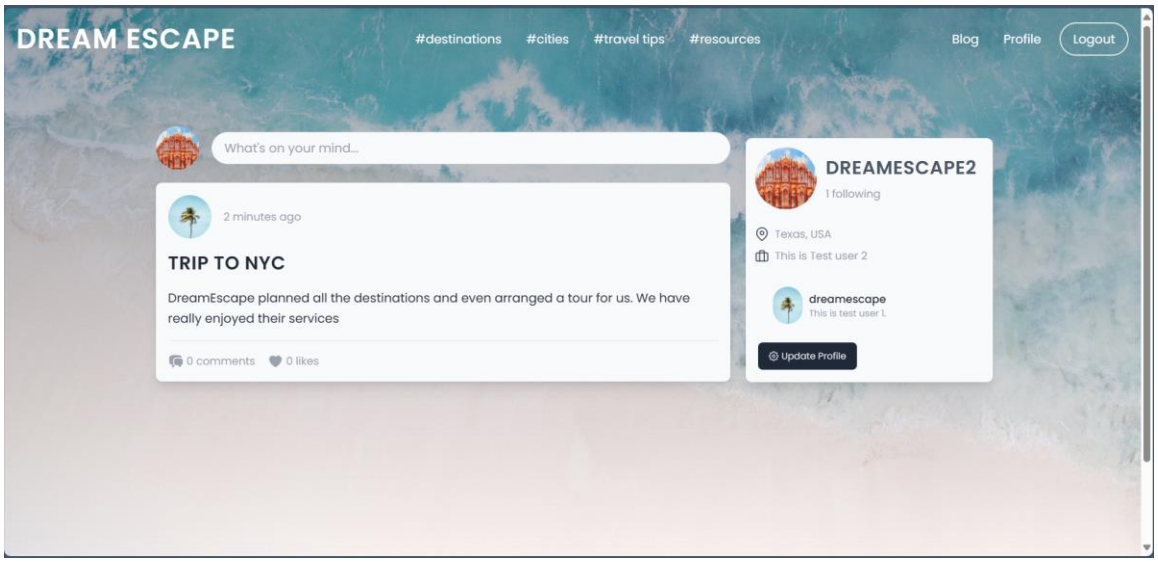
Test Cases:

This phase mostly depends on the back end along with the new features that we added:

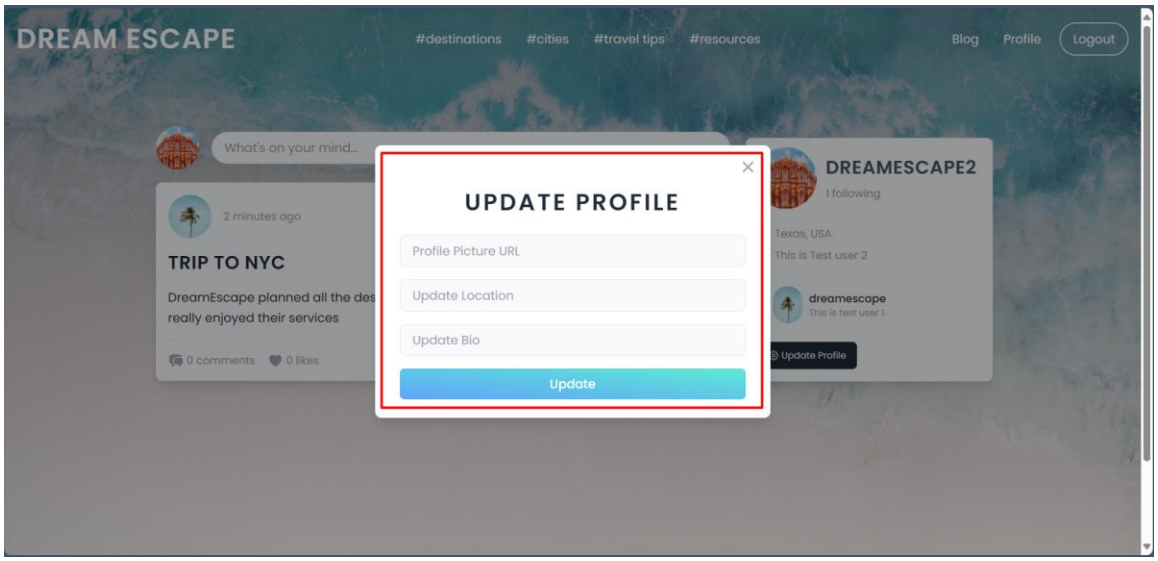
1. User Creation in the database:



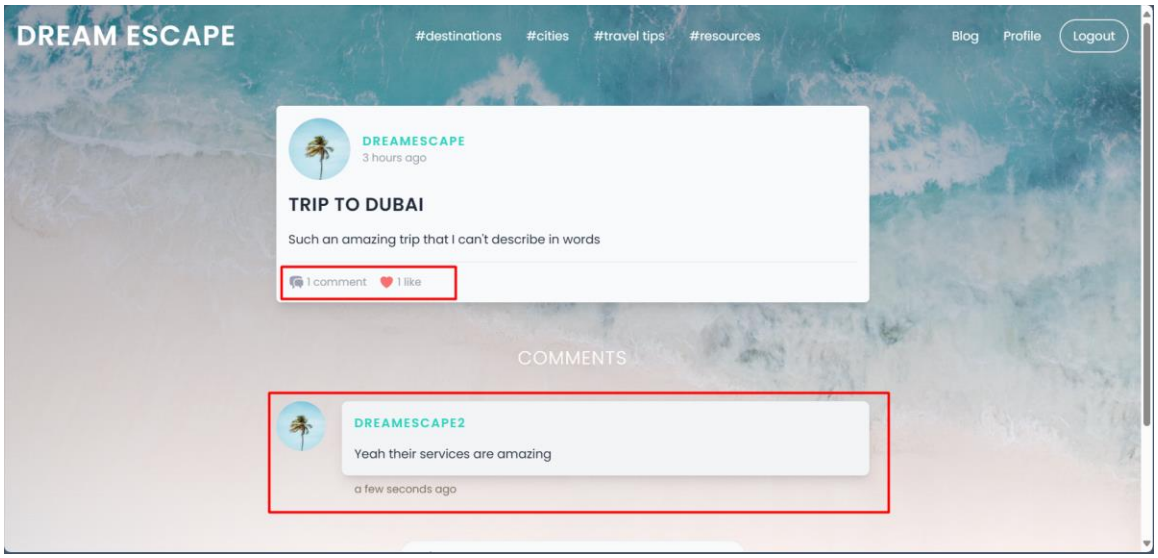
2. User Profile:



3. Profile Update:

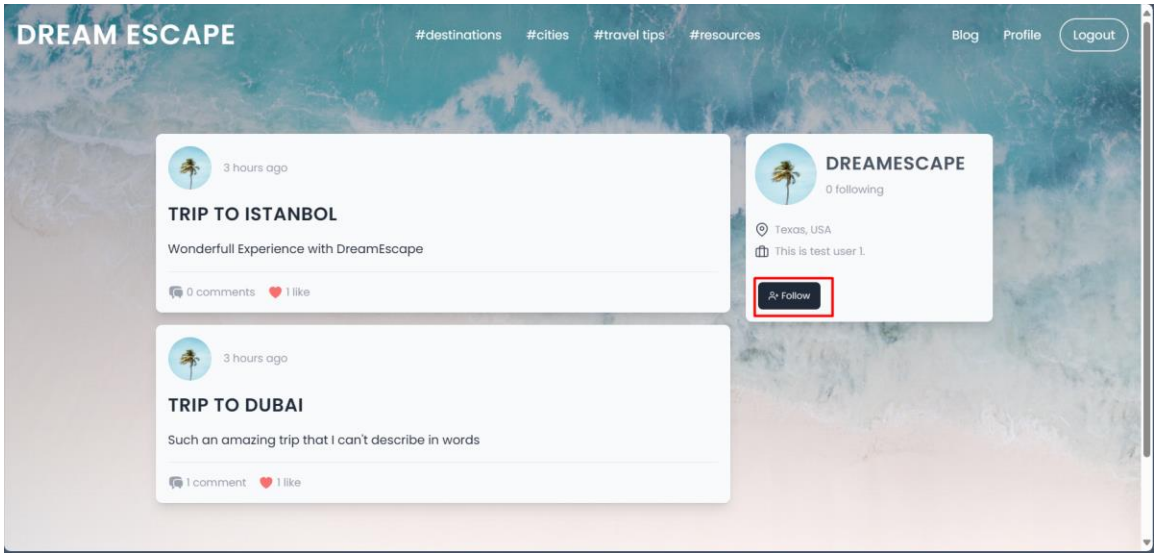


4. Like and comment a Blog:



5. Follow a user:

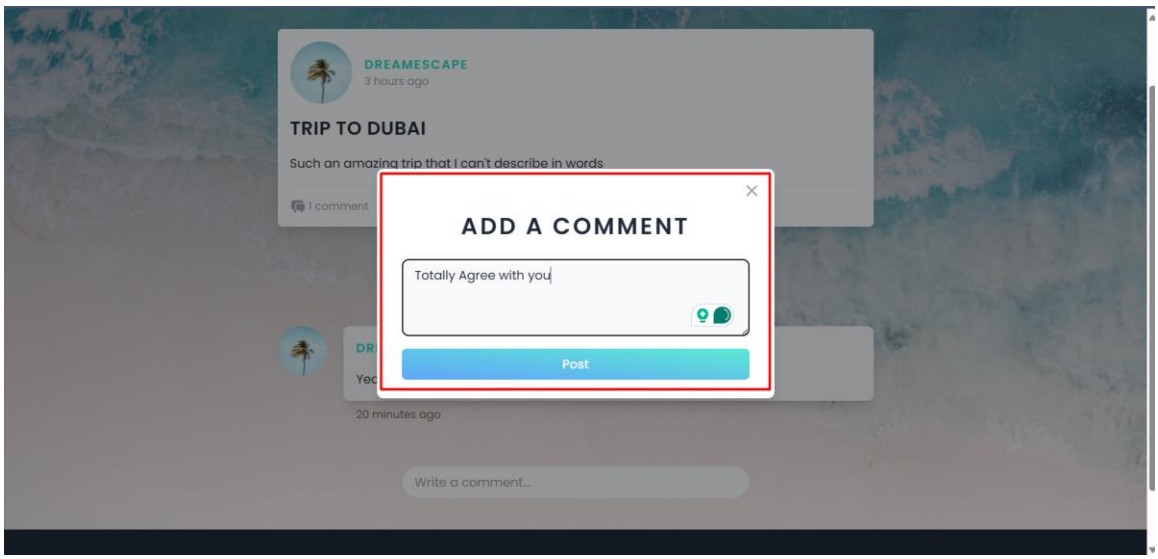
:

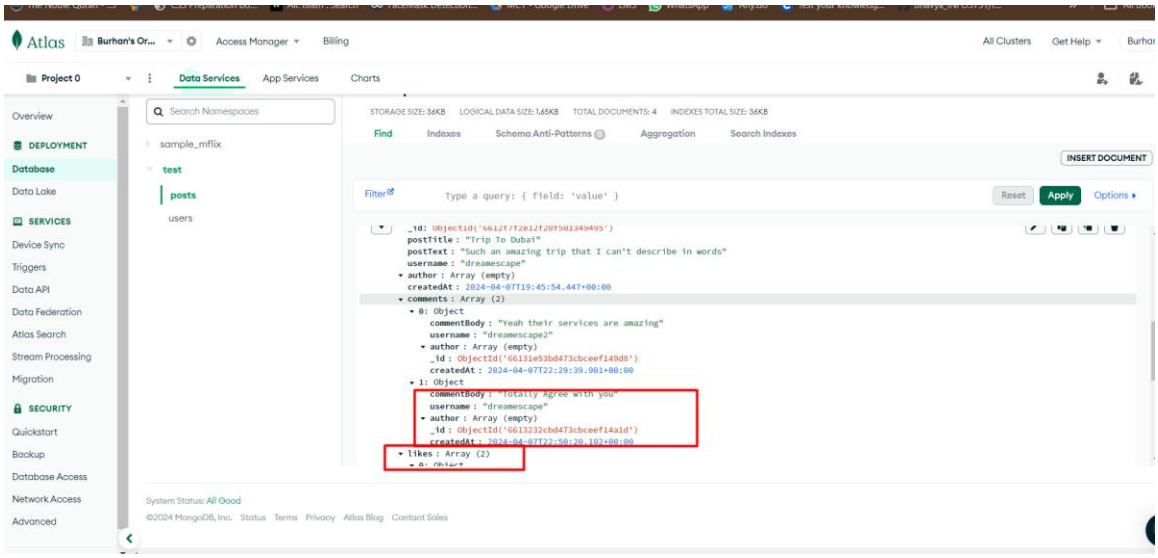


6. Posts Database:


```
_id: ObjectId('6612f7f2e12f20f501349495')
postTitle : "Trip To Dubai"
postText : "Such an amazing trip that I can't describe in words"
username : "dreamescape"
author : Array (empty)
createdAt : 2024-04-07T19:45:54.447+00:00
comments : Array (1)
  0: Object
    commentBody : "Yeah their services are amazing"
    username : "dreamescape2"
    author : Array (empty)
    _id : ObjectId('66131e53bd473cbceef149d8')
    createdAt : 2024-04-07T22:29:39.901+00:00
likes : Array (1)
  0: Object
    username : "dreamescape2"
    createdAt : 2024-04-07T19:47:38.004+00:00
    _id : ObjectId('6612f85ae12f20f5013494a9')
__v : 1
```


7. Add like/comment:





8. API Integeration:

Installation User Manual:

- **Prerequisites:**

  User must need to install Nodes.JS from the official website https://nodejs.org/en/download. Install as per the user operating system either it is Windows, Mac, or Linux. After downloading and installing the nodes, please make sure the NPM which is nodes package manager, is also installed correctly and to check this please run this command.

  npm list -g

- **Installation:**

Once nodes are configured, you must need to install some modules that are being used in this project. To do this, open the client folder in the CMD and run following command.

npm install

Once all modules are installed successfully now it's time to open the server folder inside CMD and repeat the process again. By this all required modules for the front end and back end will be installed.

Program Run User Manual:

To run the program, user need to open both client and server folders in two separate CMD. After that in both cmd please write this command

npm start

This will start two ports. On port 3001 the server is running which is back end of our project and on port 3000 the front end will start running and you can access this by this url in your local browser

http://localhost:3000/

Report Code Inspection Feedback:

Initially our target was to just create the front end as the phase 1 of the development but with the peer review and teacher review, we must change our plan as they prescribed the best practice will be to make both front end and back end alongside and this will save a lot of end hustle as this can potentially reduce development bugs. We have welcomed these reviews and made changes in our development phases requirements and as you can see in this phase, we have created all 3 pillars of website: front end, back end, and database.

Report Reflection:

Our team was good at developing the front-end part i.e. the signup page and all the elements were well accomplished in the signup page. We faced few difficulties in the backend part, much care was needed to manage dependencies between modules for Node JS and React JS. Modifications to the codebase and project schedules were required to respond to criticism. Initially we planned to use the SQLite3 database but by looking at our backend, we thought that MongoDB would best fit and changed from SQLite3 to MongoDB. This was a little challenging to switch from SQlite3 to MongoDB.

Collaboration:

Member name	Contribution description	Overall Contribution (%)
Veeresh Sakali	I have mainly worked on the front-end part especially on updating the Profile Page and Sign-up page. Also, helped other teammates working under front-end.	12.5%
Susmith Meesa	I have worked on class diagram which acts as a blueprint to understand the relation between	12.5%

	classes and attributes, I also have assisted in back-end part.	
Harshath Budida	I have worked on Back-end part mainly on GraphQL queries, Mutations and Authentication model.	12.5%
Harshitha Thokala	I have worked on booking page and Details page for places which comes under front-end part and was a part of documentation making.	12.5%
Gopi Krishna Kummari	I have worked on front-end part Blogs/feedback reactions and Contact form and helped in documentation process.	12.5%
Pooja Sree Poka	I have worked on back-end part mainly on Blogs/Feedback and Contact form and was a helping hand in documentation.	12.5%
Sashidhar Chary Viswanathula	I have worked on sequence diagram which is useful for understanding the flow control and communication.	12.5%
Balaji Valeti	I have worked on use case diagram which is a successful method for defining and arranging functional needs and documented the test cases.	12.5%