Sight See Tourist Map – Report 2

Software Engineering – Spring 2025

Cover Page

Project Name: Sight See Tourist Map
Course Name: Software Engineering
Submission Date: April 2nd, 2025
GitHub Repository: https://github.com/sublimebovine/Sight-See.git

Team Members:

- Rayyan Barbhuiya - rb1269scarletmail.rutgers.edu
- Abhinav Bollu - ab2445@scarletmail.rutgers.edu
- Aaryan Handa - ah1351@scarletmail.rutgers.edu
- Sean Johnson - sbj33@scarletmail.rutgers.edu
- Saahil Patel - sp2183@scarletmail.rutgers.edu
- Nirshanth Kiritharan - nk833@scarletmail.rutgers.edu
- Sachit Nigam - sn789scarletmail.rutgers.edu
- Adiel Torres - amt332scarletmail.rutgers.edu

Table of Contents

1. Domain Model

- Overview of entities
    - User
        - user_id, name, email, password_hash, role (visitor, registered, admin), preferences
        - represents individuals interacting with the app. Roles determine access to reviews, saved locations, and admin moderation
    - Attraction
        - attraction_id, name, description, location, category, average_rating, is_sponsored
        - places of interest gathered from Google Places API and user submissions. Sponsored listings are specially marked and promoted.
    - Review
        - review_id, user_id, attraction_id, rating, comment, confidence_score, status (approved, pending, rejected)
        - reviews left by registered users. Each is assessed for credibility and moderated before publication.
    - WhetherReport
        - report_id, attraction_id, temperature, conditions, source (API/user), timestamp

        - weather conditions retrieved from OpenWeather API or submitted by users. Used to filter attraction availability.
    - Map
        - map_id, search_query, attraction_ids, last_rendered
        - stores search data used to generate map views for caching and offline mode support.

- Overview of services (business logic)
    - Authentication Service
        - handles registration, login, role verification, and session management.
    - Search Service
        - processes keyword/location queries.
        - interfaces with Google Places API to return matching attractions.
        - applies filters (weather, budget, rating, category).
    - Review Moderation Service
        - validates review content for spam or abuse.
        - assigns a confidence score (based on history, keyword checks).
        - forwards reviews to admins if flagged.
    - Weather Service
        - pulls real-time forecasts from OpenWeather.

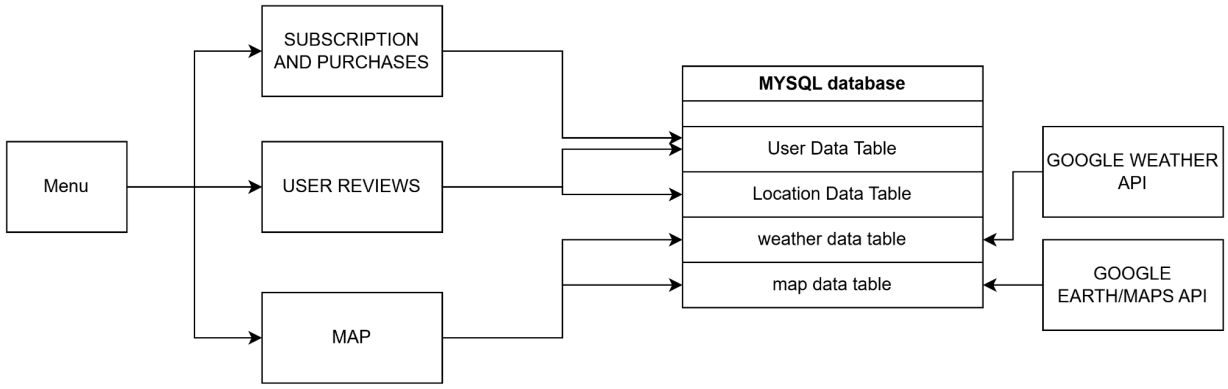
        - stores weather data locally for caching and filtering.
        - validates user-submitted conditions.
    - Map Rendering Service
        - generates interactive maps using Google Maps API.
        - handles fallback rendering from cached data if APIs fail.
    - Admin Service
        - enables review approval/rejection, sponsored listing management, and general moderation.

Entity service relationships

Entity	**Services**	**Access Type**
User	Authentication, Review Moderation	Read/Write
Attraction	Search, Map Rendering, Admin	Read/Write (Admin Only)
Review	Review Moderation, Admin	Read/Write
WeatherReport	Weather Service	Read/Write
Map	Map Rendering	Read/Write

2. Database Schema

ER diagram



SQL table definitions

User Data- Account and login information as well as subscriptions, user history and review confidence

Location Data-unique location data to our software containing Sight See specific recommendations and information

Weather data- Sight See weather information collected from google weather API and reported by users

Map data- Map references list to populate the interactive map.

Data source and population plan (auto/API/user input)

User Data will be populated by recordings of user activity and self reported preferences and information

Location data will be input by location managers with Sight See approval and updated with user comments and reviews

Weather data is taken from google weather API with confirmation and input from users to supplement this

Map data will be populated using google maps as well as location manager input.

Role based access description (visitor, registered, admin)

Visitors can access the online map to find locations but cannot write reviews or contribute location information whereas a registered user can read and write reviews as well as contribute location data and a admin can directly access the mysql database as well as api keys and may remove or reduce the visibility of other users activity,

3. API Specification

Overview of REST API design

Our API will mostly be used to access data and validate the users identity and credentials as well as manage our use of external api calls and regulate user activity.

API endpoints table (URI, method, inputs, response, status codes)

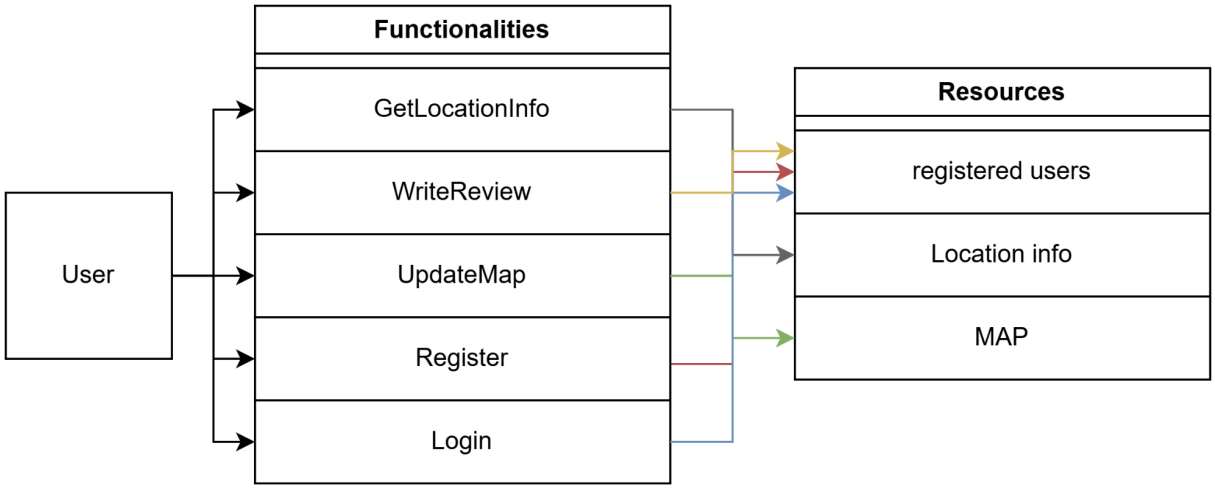
URI	Method	Inputs	Response	Status codes
`https://api.SightSee.com/locations/3`	LocationInfo()	Authorization key from client after login,User id, selected location	Location info appropriate to user status including reviews and weather	200 (OK) 404(Not found) 400(Bad request)
`https://api.SightSee.com/MAP/3`	MapUpdate()	User id, map segment	Allows api call to google maps to refresh map	200 (OK) 404(Not found) 400(Bad request)
`https://api.SightSee.com/registeredusers/3`	Login()	Password, username	Allows user to log into account	200 (OK) 404(Not found) 400(Bad request)
`https://api.SightSee.com/registeredusers/3`	Register()	Password, username, email	Allows user to register account	200 (OK) 404(Not found) 400(Bad request)

`https://api.SightSee.com/registeredusers/3`	WriteReview()	Authorization key from client after login,user id	Allows user write review	200 (OK) 404(Not found) 400(Bad request)

Input validation and error handling

Error handling for our api will return 404 if a service is down and 400 if the user inputs a bad request in either case an error report prompt will appear allowing the user to input what they attempted to do and we will review the nature of the error.

API use case mapping



4. Third-Party APIs

APIs used and their purpose

- Google Maps API
    - We are using the google maps API for searching for attractions and mapping their details. This API will be the one responsible for the interactive map that will provide users with details and directions to the user-specified attractions.
- Google Places API
    - The Google Places API will primarily be used for presenting the user with detailed information about a certain location, such as user reviews, ratings, and traffic data. We will also be using it to handle user-generated ratings.

- OpenWeather API
    - This API will be used to display current and forecasted weather conditions at a given location, which will further aid the user in planning for a trip. We will create a weather-based filtering system so that users can avoid planning a trip during unfavorable weather.

Documentation URLs

- **Google Maps API:** https://developers.google.com/maps/documentation/javascript/overview

- **Google Places API:** https://developers.google.com/maps/documentation/places/web-service/overview

- **OpenWeather API:** https://openweathermap.org/api

API Integration

- We intend on creating backend processes that connect to the server, which will directly query the Google Maps API for map rendering and attraction listings based on user search criteria. The Google Places API integrations allow for the fetching and submission of reviews and ratings from registered users. This will trigger the moderation workflow, which will be monitored by the admin team.

Finally, the OpenWeather API is integrated to retrieve weather conditions to filter attractions based on ideal travel conditions.

Notes on API limitations or fallback handling

- The project will use the flexibility and reliability of these robust APIs to create a cohesive user experience. However, these APIs do have usage quotas and rate limitations, which will temporarily cause a service outage in the event that they are exceeded. For the scenario, we intend on using cached location recommendations and locally stored reviews will serve as fallback content to maintain continuous service availability. This will allow for the continuity and UX of the service to be intact despite the rate limitation.

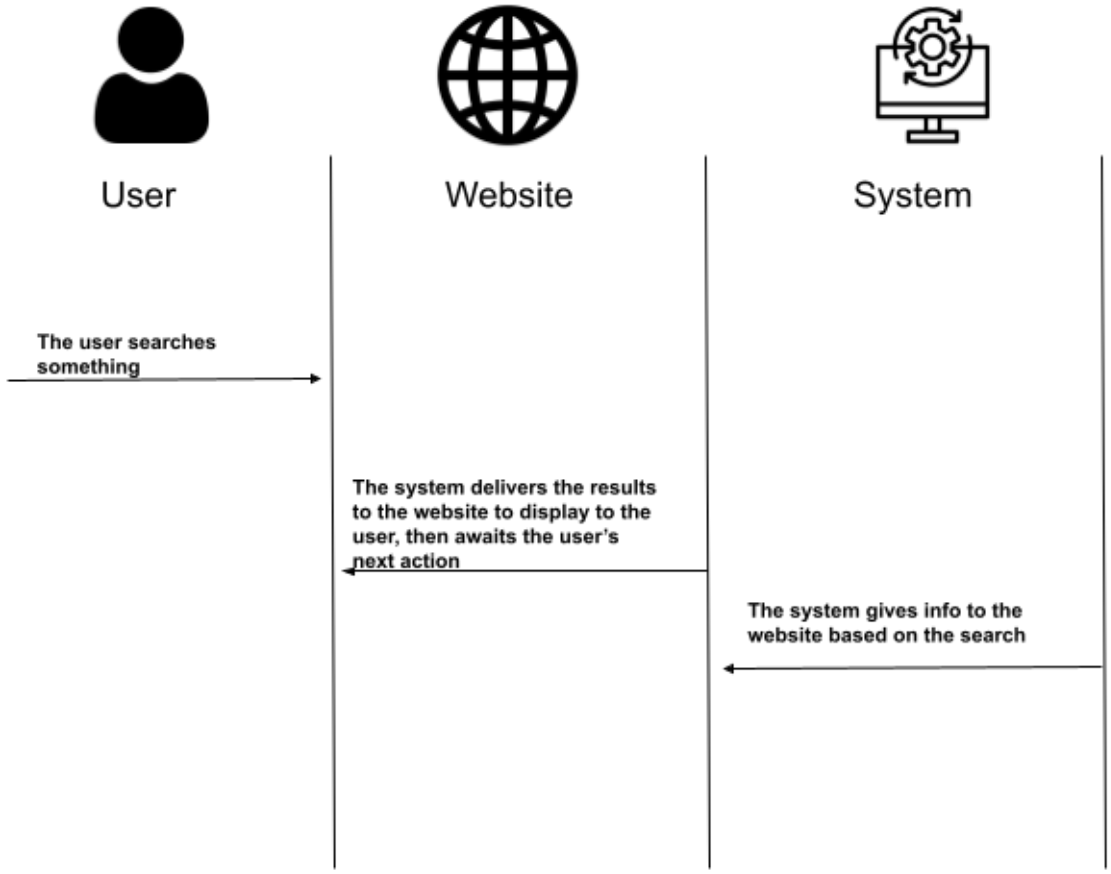
5. System Sequence Diagram

Use Case: Searching for Attractions:

Flow and Message Content:

1. The user searches for something (e.g. restaurants in Miami)
2. The system gets info on this search using google places API
3. The user selects a search result
4. The system displays information on this result (could use OpenWeather API or Google Places depending on what was searched)
5. The user is allowed to save this result

Diagram:

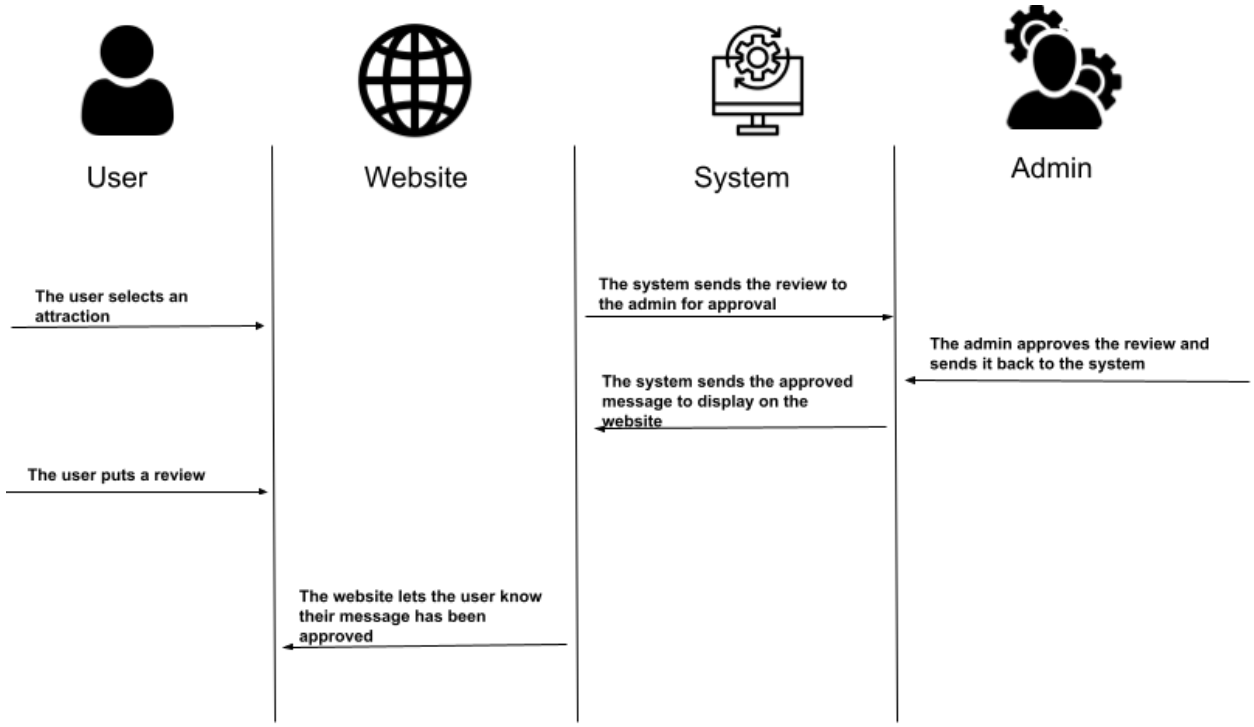


Use Case: Submitting a Review:

Flow and Message Content:

1. The user selects an attraction
2. The user enters a rating and/or text review
3. The system sends review to admin to check
4. If review is approved, it is posted

Diagram:


User
Website
System
Admin
The user selects an attraction
The system sends the review to the admin for approval
The admin approves the review and sends it back to the system
The system sends the approved message to display on the website
The user puts a review
The website lets the user know their message has been approved

6. Team Member Contribution

Who designed which APIs

- Abhinav - Responsible for designing, implementing, and integrating REST APIs. This includes managing interactions with third-party APIs such as Google Maps, Google Places, and OpenWeather. Primary tasks involve developing backend endpoints, API validation, error handling, and implementing fallback mechanisms for API limitations.
- Sean and Rayyan - Responsible for designing and implementing the SQL database, and integrating it properly with both the frontend and backend. This includes usernames and passwords, account preferences, location information, and weather information.
- Nirshanth and Sachit - Reposnsible for testing frontend and backend services. Basically making sure the website works as it should with no bugs, and identifying and problems that arise.

Who is responsible for db, testing, integration