Sight See Tourist Map – Report 3 (Project Backend Implementation)

Software Engineering – Spring 2025

Cover Page

Project Name: Sight See Tourist Map
Course Name: Software Engineering
Submission Date: May 5nd, 2025
GitHub Repository: https://github.com/LakersRutgers/sublimebovine.github.io

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

NOTE!

IN order to run code you must replace the API keys
These are private so we have removed them from the repository and are giving them to you here privately

Replace line 8 in index with
<script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyAGM_vPWCvl3L0h1DyB0vOk-aLA9dXp7Gs&libraries=places"></script>

As one line
Replace line in 1 .env

REACT_APP_OPENWEATHER_API_KEY=40f45cbd25778f84596c05c8eb1729c4

As one line
https://github.com/LakersRutgers/sublimebovine.github.io

Also to run the code you will need to load the sql database
The github contains a file called main.sql which contains our database. Load it into a mysql database called login_db

For an account to use you can use username:test password:test or make your own

1. API Implementation Overview

The backend APIs for Sight See were implemented using Node.js and Express with

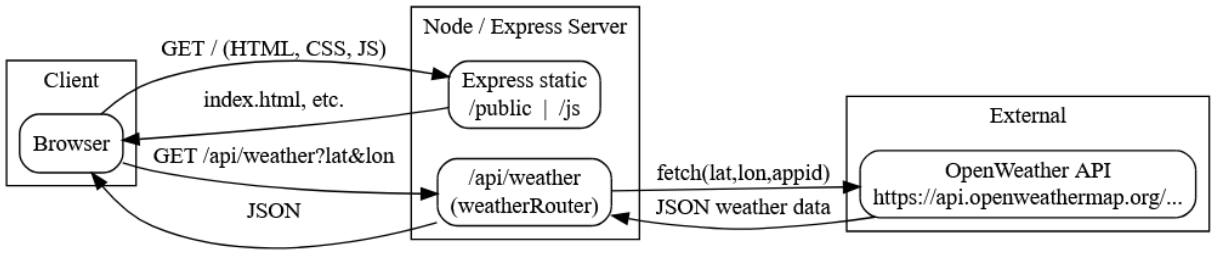
MySQL as the database. Below are four key APIs:

- Add Review API
    - Method: POST

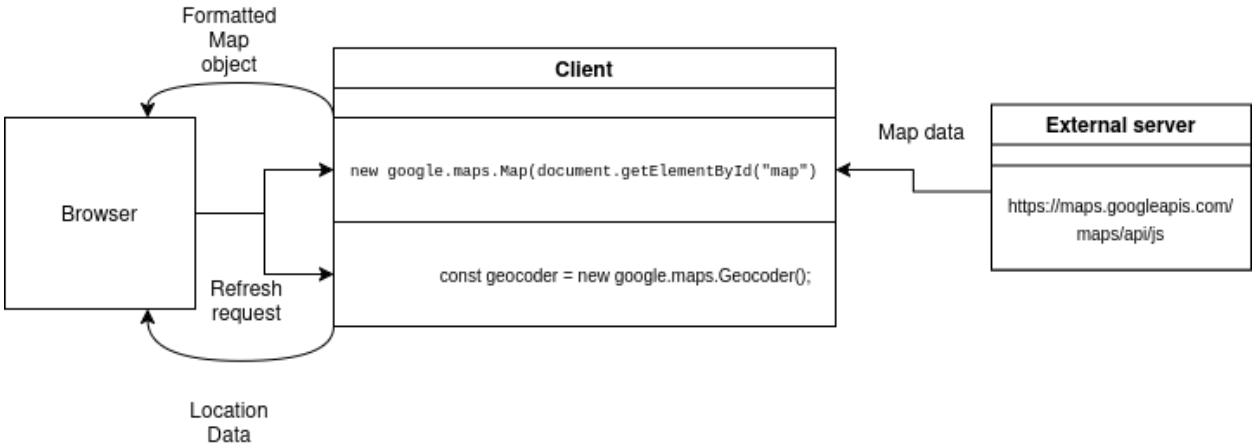
    - URI: /addReview
    - Input: username, text, rating, code, name
    - Response: 201 Created / 403 Forbidden / 500 Error
    - Database Command: INSERT INTO reviews (User, Text, Rating, Code, Name) VALUE
    - Validation: Requires user to be premium
- Get Reviews API
    - Method: GET
    - URI: /reviews
    - Input: None
    - Response: JSON array of reviews
    - Database Command: SELECT * FROM reviews ORDER BY id DESC;
- Register User API
    - Method: POST
    - URI: /register
    - Input: username, email, password
    - Response: 201 Created / 400 Bad Request / 409 Conflict
- Weather API Integration
    - Method: GET
    - URI: /api/weather?lat={lat}&lon={lon}

2. Server-Side Routing Diagram

- Weather API


Node / Express Server
GET / (HTML, CSS, JS)
Client
Express static
/public | /js
index.html, etc.
Browser
GET /api/weather?lat&lon
/api/weather
(weatherRouter)
fetch(lat,lon,appid)
JSON weather data
External
OpenWeather API
https://api.openweathermap.org/...
JSON

- Maps API


Formatted
Map
object
Client
Map data
External server
Browser
new google.maps.Map(document.getElementById("map")
https://maps.googleapis.com/
maps/api/js
const geocoder = new google.maps.Geocoder();
Refresh
request
Location
Data

3. Steps to Run Server Code in NodeJS

- Prerequisites:
    - Node.js v18+
    - MySQL installed
    - Github repo cloned locally
- Run Steps:
    - 1. Clone the repository:
        - Git clone https:github.com/sublimebovine/Sight-See.git
    - 2. Navigate to server directory:
        - Cd Sight-See
    - 3. Install dependencies:
        - Npm install bcrypt body-parser mysql2 express
    - 4. Configure DB credentials in .env:
        - DB_HOST=localhost
        - DB_USER=root
        - DB_PASSWORD=yourpassword (default this is root)
        - DB_NAME=login_db
    - 5. Start the server:
        - Node server.js

4. HTTP Messages: Client Requests & Server Responses

FOR REVIEWS:

1. GET /reviews

Purpose: Retrieves all reviews from the database to display them on the reviews page.

Triggered by: loadReviews(name) function in reviews.html.

Client Request:

Type: GET

Initiated using fetch('/reviews') in the browser.

Server Response:

Status: 200 OK on success

Body: JSON array of review objects from the reviews table.

Each review includes:

User: username of reviewer

Text: review content

Rating: numeric rating

Code: location code (optional)

Name: location name

```
app.get('/reviews', (req, res) => {
 connection.query('USE login_db', (err) => {
   if (err) {
     console.error('Error switching database:', err);
```

```
      return res.status(500).send('Error switching database.');

    }

    connection.query('SELECT * FROM reviews ORDER BY id DESC', (err,
results) => {

      if (err) {

        console.error('Error fetching reviews:', err);

        return res.status(500).send('Error fetching reviews.');

      }

      res.json(results);

    });

  });

});
```

2. POST /addReview

Purpose: Submits a new review to the server.

Triggered by: Submission of the #reviewForm form in reviews.html.

Client Request:

Type: POST

Content-Type: application/json

Body:

{

"name": "Location name",

"code": "Optional code",

"text": "Review text",

"rating": 5

}

Server Response:

Status: 200 OK with message "Review added successfully." if successful.

Status: 500 Internal Server Error with message "Error adding review." if database insert fails.

```
  try {
    const res = await fetch('/addReview', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    });


    if (res.ok) {
      alert('Review submitted!');
      //loadReviews("sean");
      form.reset();
    } else {
      alert('Failed to submit review.');
    }
  } catch (err) {
    console.error('Error submitting review:', err);
```

```
  }
```

For Weather

```
try {
  await fetch('/api/forecast', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      lat,
      lon,
      forecast: JSON.stringify(weatherData),
      timestamp: new Date().toISOString()
    })
  });
} catch (error) {
  console.error('Error saving weather data:', error);
}
```

FOR USERS

1. POST: /login

Purpose: Retrieves the login info by username to compare the entered password to the one in the database.

Triggered by: Submission of loginForm in login.html

Client Request:

Type: POST

Initiated using fetch('/login) in the browser.

Server Response:

Status: 500 on error, 401 on invalid credentials

Server message: Success message or incorrect message

```
try {
    const res = await fetch('/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
      body: data
    });

    const text = await res.text();

    if (res.ok) {
```

```
        messageDiv.textContent = text;

        messageDiv.classList.remove('error');

        messageDiv.classList.add('success');



        // Save login flag and redirect

        localStorage.setItem('loggedIn', 'true');

        setTimeout(() => {

          window.location.href = 'index.html';

        }, 1000);

      } else {

        messageDiv.textContent = text;

        messageDiv.classList.remove('success');

        messageDiv.classList.add('error');

      }

    }
```

Get response

```
    app.get('/users', (req, res) => {
```

```
 db.query('SELECT username FROM users', (err, results) => {
   if (err) return res.status(500).send('Database error.');
   res.json(results);
 });
});
```

2. POST: /register

Purpose: Adds a user to the database if there is not a user with the same name.

Triggered by: Submission of registerForm in register.html

Client Request:

Type: POST

Initiated using fetch('/register) in the browser.

Server Response:

Status: 500 on error, 400 when username is already taken

Server message: Success message on successful registration

```
      app.post('/login', (req, res) => {
        const { username, password } = req.body;

        db.query('SELECT * FROM users WHERE username = ?', [username],
(err, results) => {
          if (err) return res.status(500).send('Database error.');
          if (results.length === 0) return res.status(401).send('Invalid
credentials.');

          const user = results[0];
            if (err) return res.status(500).send('Error hashing
password.');
            bcrypt.compare(password, user.password, (err, result) => {
              if (result) {
                res.send('Login successful!');
              } else {
```

```
                res.status(401).send('Incorrect password.');
              }
            });


        });
      });
```

3. POST: /submit_email

Purpose: Adds email value into users table in SQL database.

Triggered by: Submission of submitEmail in premium.html

Client Request:

Type: POST

Initiated using fetch('/submit_email) in the browser.

Server Response:

Status: 500 on error, 400 on duplicate or missing email.

Server message: Redirects to either thankyou.html or duplicate.html depending on the conditions

```
app.post('/submit_email', (req, res) => {
  const { email } = req.body;

  if (!email) {
    return res.status(400).send('Email is required.');
  }

  const insertEmailQuery = 'INSERT INTO emails (email) VALUES (?)';
  db.query(insertEmailQuery, [email], (err) => {
    if (err) {
      if (err.code === 'ER_DUP_ENTRY') {
        return res.status(400).redirect('/duplicate.html')
      } else {
```

```
        console.error('Database error:', err);
        return res.status(500).send('Database error.');
      }
    }
    res.redirect('/thankyou.html');
  });
});
```

5. Third-Party API Interactions

- OpenWeather
    - Endpoint used:
        - GET

          https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lo

          n}&appid={API KEY}
    - Inputs:
        - latitude, longitude
    - Reponse:
        - JSON weather data parsed and returned to the frontend
- Google Maps API
- <script src="https://apis.google.com/js/api.js"></script>
    - Endpoint used:
        - We used the below api wrapper to load the map

        - map = new google.maps.Map(document.getElementById("map"), {
    - Reponse:
        - Map object

6. Security and Validation Measures

- Authentication:
    - User must login to access review and save features
- Authorization:
    - Only premium users may write reviews
- SQL Injection Prevention:
    - All queries use parameterized inputs
- Input Validation:
    - Ratings are limited to integers 1-5
    - Text field sanitized
- Status Codes:
    - 201 for created resources
    - 403 for unauthorized review submissions
    - 400 for bad input

- Encryption
    - All user passwords are hashed to protect them.

7. Team Member Contributions

Team Member	Contributions
Abhinav Bollu	Wrote the logic and api routes for the openweather api
Sean Johnson	Maps API, Maps Locations, Login & Registration(with rayyan), Github setup, testing environment,set up and managed Mysql database, presented demo, Set up mysql and server for other team members
Nirshanth Kiritharan	Testing and enforcing general + premium review perms
Rayyan Barbhuiya	Created login and register pages and related scripts with Sean, integrated them with the main page, set up server.js, set up the main page including Maps API.
Saahil Patel	Created the premium account feature and the email storing database. Also worked on frontend html scripting and helped with testing/debugging of functions.
Aaryan Handa	Designed the Webpage for the premium page and the html frontend part for the Login page. Helped with testing and debugging the frontend part. Me and Saahil worked on storing the email in the database and linking those together.
Sachit Nigam	Testing/optimization and creating the reviews page and questions table on it. Also creating the “Reviews” physical

	button on the homepage of the map (in reviews.html/server.js) as well as making sure a review is saved in the database. I had also presented the “Reviews” page breakdown and explanation on the slides.
Adiel Torres	Helped adjust OpenWeather API routes and integrated them with the front end. Developed a button that triggered API calls and updated the website with current conditions.

8. Future Backend Improvements

- Token based Authentication:
    - Replace session based login with JWT to improve security and scalability
- Logout Functionality:
    - Implement /logout route to invalidate session data or clear client tokens
- Admin Access Controls:
    - Add an admin user type with privileges to delete reviews or manage users
- API Rate Limiting:
    - Prevent abuse of third party APIs with request throttling
- Review Editing:
    - Allow premium users to edit or delete their own reviews
- API Caching

    - Store recent weather data temporarily to reduce API calls and speed up responses
- Weather Based Recommendations:
    - Suggest occasions based on current and forecasted weather conditions

9. Known Bugs or Limitations

- No Session Persistence:
    - Current login does not maintain sessions across browser refreshers or tabs
- No Email Verification:
    - Accounts are created without verifying email ownership
- Weather API Overlap:
    - Repeated requests can sometimes exceed free tier API limits
- Hardcoded Review Filtering:
    - Review retrieval logic does not currently allow location specific filtering
- Minimal Error Feedback:
    - User facing error messages are limited or generic in placed (for example review submission errors)