CUSTOMER LOCATION PINING SYSTEM

Email: just.andrew.ckw@gmail.com

setup Instructions

Clone the repository

unawarexi/Location-Pinning-System: repo for customer location pining system using `javascript` `react` `google Map Api` `express` `mongoDB` and more... (github.com)

- git clone <repository-url>
- cd <repository-directories>

Install dependencies

- npm install

Run the project

- npm run dev for both

Build the project for production

- npm run build

Project Structure

Frontend

- React Application: Built using create-react-app.
- Components: Modular components for reusability.
- Styling: Tailwind CSS for responsive design.
- Lazy Loading: react-lazyload for efficient image loading.
- Caching: Service workers for offline capabilities and faster load times.

Backend

- API Endpoints: For retrieving and storing location data.
- Database: To store location pins and related information.

System Functionality Demonstration

Auth, Landing, Create, View, Details, Update, Delete Page

- Displays a gallery of images representing capabilities of locations in different continents.
- Each customer is displayed in the view customer page
- On click customer you get to view a complete detail and their Location distance
- Allows users to pin locations on a map and get the calculated distances using user IP address.
- Efficient location tracking across various continents and more accurately based on zip codes.
- Swift response to user interactions.

Responsive Design

- Ensures the application works seamlessly across different devices and screen sizes.
- Optimized Image Loading
- Uses lazy loading to improve performance.
- Caches images to reduce load times on subsequent visits.

Detailed Overview of Frontend Topics Covered

Project Initialization

- Setting up the React project using create vite@latest.
- Initial project structure and setup.

Component and folder Creation

- Building the main LandingPage component.
- Modular approach for easier maintenance and scalability.
- Folders include: component, pages, error, Apis, animation, assets, layout, container

Styling with Tailwind CSS

- Integrating Tailwind CSS into the project.
- Applying responsive design principles.
- Ensuring consistent and modern design across the application.

Lazy Loading and Caching

- Implementing react-lazyload for images.
- Adding service workers for offline capabilities and faster load times.

Dynamic Content Rendering

- Creating an array of objects for images and descriptions.
- Mapping over the array to dynamically render content.

SVG Icon Integration

Converting and integrating SVG icons for a more uniform quality look

- Using SVGs for UI elements like links and buttons.

Location Pinning System

- Designing a system to allow users to pin locations on a map.
- Ensuring accurate and efficient location tracking.
- Longitude and latitude calculation based on unique selected user
- Displays all retrieved data as required
- . env for Api keys

Optimized Image Layout

- Ensuring images are laid out without unnecessary grid spaces.
- Adjusting CSS styles for a seamless gallery experience.
- Descriptions for Images
- Adding meaningful descriptions to each image.
- Enhancing the user experience with informative content.

Testing and Debugging

- Thorough testing to ensure all features work as expected.
- Debugging and fixing any issues that arise.

Documentation

- Writing detailed documentation for project setup and usage.
- Documenting the functionality and features of the system.

Customer Location Pinning System using Google Maps API

Backend Project Overview

- Objective: Allow users to pin and manage locations on a map.
- Technologies: Google Maps API, Node.js, Express, MongoDB, JWT, bcrypt.

1. Project Setup

- Environment Setup
- Install Node.js and npm.
- Initialize a new Node.js project.
- Install necessary packages: Express, Mongoose, Google Maps API, JSON Web Token (JWT), bcrypt, dotenv.

1i. Directory Structure

- Create necessary folders: controllers/, models/, routes/, middleware/, config/.
- Create essential files: index.js, .env.
- Setting up nodemon and package.json script for instant reloading

2. Google Maps Integration

- Google Maps API Key
- Obtain an API key from Google Cloud Platform.
- Store the API key in the .env file.
- Frontend Integration
- Include the Google Maps JavaScript API in the frontend structure.
- Initialize the map and enable pinning functionalities.

3. CRUD Functionalities

3i. Create Location

Endpoint: POST /api/customers/createCustomers

Functionality

- Extract location data from customer request body.
- Validate data.
- Create a new location document in the database.
- Return the created location details.
- Read Locations

3ii. **Retrieve registered locations**

Endpoint: GET /api/customers/getAllCustomers

Functionality

- Fetch all customer documents from the database.
- Return a list of customer addresses and possible locations.

3iii. Update Location

Endpoint: PUT /api/customers/:id

Functionality

- Extract customer unique ID from URL parameters.
- Extract updated data from request body.
- Find and update the address and zip codes by ID.
- Return the updated location document.

3iv . Delete Location

Endpoint: DELETE /api/customers/:id

Functionality

- Extract customer ID from URL parameters.
- Find and delete the data by ID.
- Return a success response.

4. User Authentication

Registration

Endpoint: POST /api/users/register

Functionality

- Extract user data from request body.
- Validate and hash the password.
- Create a new user document in the database.
- Return user details (excluding password).

Login

Endpoint: POST /api/users/login

Functionality

- Extract email and password from request body.
- Validate credentials.
- Generate a JWT token.
- Return the token and user details.

5. Middleware Tokenization

- Token Verification Middleware
- Extract token from Authorization header.
- Verify the token using the secret key.
- Attach decoded user information to the request object.
- Proceed to the next middleware or return an error if invalid.

6. Route Protection

- Protecting Routes
- Apply the token verification middleware to routes requiring authentication.
- Expires in every hour to ensure unnecessary alter of customers data

7. Database Models

User Model

Schema

- Define fields: email, password.
- Ensure email uniqueness.

Customer model

Schema

- Defines fields states : names, address, and email
- Ensures uniqueness of zipcodes for proper tracking

8. Configuration

- Environment Variables
- Create a .env file:
- PORT
- MONGODB_URI

- JWT_SECRET
- GOOGLE_MAPS_API_KEY (frontend. env)
- Database Connection
- Configure the database connection.
- Server Setup
- Initialize the Express server.
- Set up the routes.

9. security practices

- Error Handling
- Implement error handling for all routes and controllers.
- Security Best Practices
- Ensure passwords are hashed.
- Validate and sanitize inputs.