Documentation for the Map Capture and 3D Visualization Application

---

Overview:

This project is a **ReactJS-based** full-stack(MERN) application that allows users to interact with Google Maps, capture map regions, and visualize them in 3D using **BabylonJS**. The app also features a **Node.js** backend with **MongoDB** for data persistence and includes an endpoint for processing frequently captured map regions with caching for performance optimization.

Frontend (ReactJS):

The frontend consists of a **ReactJS** app that integrates with Google Maps to allow users to select and capture visible map regions. These regions are then applied as textures to a 3D cuboid rendered using **BabylonJS**. The app features a responsive user interface with styled components for user interaction.

Key Components:

1. **App.js**:
    - Main container of the app, responsible for rendering the **MapComponent** and the **Canvas3D** component.
    - Handles fetching saved map captures from the backend and dynamically displaying them.
    - Allows users to apply captured images as textures to a 3D cuboid.
2. **MapComponent.js**:
    - Implements the Google Maps API to display an interactive map.
    - Captures the current visible region on the map, generates a static image using the Google Maps Static API, and sends the image along with map center coordinates to the backend.
    - Includes a button to trigger map capture.
3. **Canvas3D.js**:
    - Uses **BabylonJS** to render a 3D cuboid.
    - Applies the captured map image as a texture on the cuboid.
    - Supports dynamic updates when the user selects a different saved map capture.

CSS Styling:

The UI includes a responsive layout with modern styles, making the map full-screen and arranging the saved captures list with flexbox for a neat presentation. Buttons and captures are visually enhanced with hover effects.

Backend (Node.js/Express):

The backend is built using **Node.js** and **Express.js**, providing APIs to store and retrieve map captures in **MongoDB**. The backend includes advanced features such as the ability to identify the top three most frequently captured regions and implements caching for performance.

Key API Endpoints:

1. **POST `/api/captures`**:
   - Saves the captured map data (image URL and center coordinates) into MongoDB.
   - Returns the newly saved map capture.
2. **GET `/api/captures`**:
   - Retrieves all saved map captures from MongoDB.
   - Returns a list of map captures, including the image URL and coordinates.
3. **GET `/api/top-regions`**:
   - Analyzes the map captures stored in the database to identify the top three most frequently captured regions.
   - Returns the top three regions based on capture frequency.

Caching Mechanism:

- **In-Memory Caching** using **NodeCache**: Frequently accessed map data is cached in memory to reduce redundant database queries and improve performance. This is particularly useful for retrieving frequently captured regions or map data.

MongoDB:

The MongoDB database stores captured map details, including:

- **imageUrl**: The URL of the static map image.
- **center**: Latitude and longitude coordinates of the map center.
- **createdAt**: The timestamp of when the capture was created.

Capture Schema:

```
const captureSchema = new mongoose.Schema({
  imageUrl: String,
  center: {
    lat: Number,
    lng: Number
  },
  createdAt: { type: Date, default: Date.now },
});
```

Top Regions Algorithm:

The backend processes the stored map data by grouping captures based on proximity (clustering them using latitude and longitude) and counts the occurrences of similar regions. It then returns the top three regions based on frequency.

Caching Strategy:

- **NodeCache** is used to store frequently accessed map captures, including the top three regions.
- The cache reduces database load by serving cached data for frequent requests, improving response times.

How to Run the Application:

- **Backend Setup**:
  - Install dependencies: `npm install`.
  - Set up the environment variables in a `.env` file, including:
    ```
    MONGODB_URI=mongodb+srv://your_mongo_uri
    PORT=5000
    ```
  - Run the backend server: `npm start`.
- **Frontend Setup**:
  - Install frontend dependencies: `npm install` (inside the `client` directory).
  - Ensure you have the correct Google Maps API key in the **MapComponent.js** file.
  - Run the React frontend: `npm start`.
- **Accessing the Application**:
  - Frontend runs on: `http://localhost:3000`
  - Backend runs on: `http://localhost:5000`

Dependencies:

- **React.js**: Frontend framework.

- **Babylon.js**: For 3D visualization.
- **Google Maps API**: For map interaction and capture.
- **Node.js & Express.js**: Backend framework.
- **MongoDB**: NoSQL database for storing map captures.
- **NodeCache**: Caching library for performance optimization.

Conclusion:

This application is a complete example of integrating **Google Maps**, **3D visualization**, and a **full-stack MERN architecture**. The application features dynamic map capture, 3D rendering with **BabylonJS**, optimized data access using caching, and an API endpoint for identifying popular map regions.