TSAW DRONES

PROBLEM STATEMENT 1

To implement this we need a web Application using React.js, Node.js, Express.js, MongoDB

**Backend (Node.js & Express.js)**

1. **Setup Node.js and Express.js Server:**
   * Initialize a Node.js project and set up Express.js for handling HTTP requests.
2. **Database Integration (MongoDB):**
   * Use MongoDB to store the coordinates (latitude and longitude) and any other relevant data.
3. **API Endpoints:**
   * Create an API endpoint to receive the two points (latitude and longitude).
   * Develop an API to calculate the path between the two points and store it in the database.
4. **Path Calculation:**
   * Use a geographical library (like Turf.js) to calculate the path between the two points.

Code Example (Simplified)

Backend - Node.js/Express.js

JAVASCRIPT CODE:

const express = require('express');

const app = express();

const port = 3000;

app.use(express.json());

app.post('/api/path', (req, res) => {

res.json({ message: 'Path calculated and stored.' });

});

app.listen(port, () => {

console.log(`Server running on port ${port}`);

});

Frontend - React.js

JAVASCRIPT CODE:

import React, { useEffect, useState } from 'react';

import axios from 'axios';

function App() {

const [path, setPath] = useState(null);

useEffect(() => {

// Fetch path data from backend

axios.get('/api/path')

.then(response => {

setPath(response.data);

// Animate avatar here

});

}, []);

return (

<div>

{/\* Map and Avatar components go here \*/}

</div>

);

}

export default App;

PROBLEM STATEMENT 2

Developing a Quiz Web Application involves multiple components, including backend and frontend. Below is a simplified implementation in both Node.js/Express.js for the backend and React.js for the frontend.

Backend (Node.js, Express.js, MongoDB):

1.Initialize Node.js Project:

CODE:

mkdir quiz-app-backend

cd quiz-app-backend

npm init -y

2.Install Dependencies:

CODE:

npm install express mongoose body-parser nodemailer multer

3Create Server (server.js):

JAVASCRIPT CODE:

const express = require('express');

const bodyParser = require('body-parser');

const mongoose = require('mongoose');

const nodemailer = require('nodemailer');

const multer = require('multer');

const app = express();

const port = 3000;

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

mongoose.connect('mongodb://localhost/quizdb', { useNewUrlParser: true, useUnifiedTopology: true });

app.post('/api/register', (req, res) => {

});

app.post('/api/start-test', (req, res) => {

});

app.post('/api/submit-answer', (req, res) => {

});

app.listen(port, () => {

console.log(`Server is running on port ${port}`);

});

4.Set Up MongoDB:

* Install MongoDB and start the server.
* Create a database named quizdb.

5.Define MongoDB Models (models/User.js, models/Question.js):

javascript

CODE:

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

});

const questionSchema = new mongoose.Schema({

});

const User = mongoose.model('User', userSchema);

const Question = mongoose.model('Question', questionSchema);

module.exports = { User, Question };

6.Implement Route Logic:

* Implement registration logic.
* Randomly select questions for the test.
* Capture and store images during the test.
* Implement automatic test submission and warnings.

**Frontend (React.js):**

1.Initialize React.js Project:

CODE:

npx create-react-app quiz-app-frontend

cd quiz-app-frontend

2.Install Dependencies:

CODE:

npm install axios react-router-dom

3.Create Components (src/components):

* RegistrationForm.js
* Quiz.js
* Question.js
* Result.js

4.Define Routes (src/App.js):

jsx

CODE:

import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import RegistrationForm from './components/RegistrationForm';

import Quiz from './components/Quiz';

import Result from './components/Result';

function App() {

return (

<Router>

<Switch>

<Route path="/register" component={RegistrationForm} />

<Route path="/quiz" component={Quiz} />

<Route path="/result" component={Result} />

</Switch>

</Router>

);

}

export default App;