HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>AstroScope</title>
  <style>
    body {
      margin: 0;
      font-family: Arial, sans-serif;
    }
    .navbar {
      position: fixed;
      width: 100%;
      background: rgba(255, 255, 255, 0.05);
      backdrop-filter: blur(5px); /* Reduced blur effect */
      -webkit-backdrop-filter: blur(5px); /* Reduced blur effect */
      box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
      display: flex;
      justify-content: space-between;
      align-items: center;
      padding: 10px 20px;
      box-sizing: border-box;
    }
    .navbar h1 {
      margin: 0;
      font-size: 1.5em;
      color: white;
    }
    .navbar ul {
      list-style: none;
      margin: 0;
      padding: 0;
      display: flex;
    }
    .navbar li {
      margin-left: 20px;
    }
    .navbar a {
      text-decoration: none;
      color: white;
      font-weight: bold;
    }
    .navbar a:hover {
      text-decoration: underline;
    }
    .intro-text {
      position: absolute;
      top: 50%;
      left: 50%;
      transform: translate(-50%, -50%);
      text-align: center;
      color: white;
      opacity: 0;
      transition: opacity 2s ease-in-out;
      transition-delay: 5s; /* 5 second delay */
    }
    .intro-text h1 {
      font-family: "Lexend Deca", sans-serif;
      text-transform: uppercase;
      font-size: 50pt;
      margin: 0;
```

```
}
.intro-text h2 {
  font-family: "Lexend Deca", sans-serif;
  text-transform: uppercase;
  font-size: 40pt;
  margin: 0;
}
.intro-text p {
  font-family: "Lexend Deca", sans-serif;
  letter-spacing: 0.5em;
  font-weight: bold;
  font-size: 1.2em;
}
.intro-text button {
  margin-top: 20px;
  padding: 10px 20px;
  font-size: 1em;
  font-weight: bold;
  color: white;
  background: rgba(255, 255, 255, 0.1);
  border: none;
  backdrop-filter: blur(5px);
  -webkit-backdrop-filter: blur(5px);
  border-radius: 10px;
  cursor: pointer;
}
.fade-out {
  opacity: 0;
}
.search-container {
  position: absolute;
  top: 20%;
  left: 20px;
  width: 300px;
  background: rgba(255, 255, 255, 0.1);
  backdrop-filter: blur(10px);
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  transition: top 0.5s ease-in-out;
  overflow: visible;
  padding: 20px;
  box-sizing: border-box;
  z-index: 1002; /* Increased z-index to ensure it's always on top */
}
.search-container.top {
  top: 10px; /* Adjust this value as needed */
}
.search-input {
  position: relative;
  width: 100%;
}
.search-input input {
  width: 100%;
  padding: 10px 40px 10px 10px; /* Added right padding for icon */
  border: 2px solid white;
  border-radius: 10px;
  outline: none;
  background: transparent;
  color: white;
  box-sizing: border-box;
}
.search-input input::placeholder {
  color: rgba(255, 255, 255, 0.7);
  }
  .autocom-box {
    position: absolute;
    top: 100%;
    left: 0;
    width: 100%;
    background: rgba(255, 255, 255, 0.1);
    backdrop-filter: blur(10px);
    border-radius: 0 0 10px 10px;
    max-height: 200px;
    overflow-y: auto;
    display: none;
    z-index: 1000; /* Ensure it's above other elements */
    margin-top: 5px; /* Add some space between input and suggestion list */
  }
  .autocom-box li {
    list-style: none;
    padding: 10px;
    cursor: pointer;
    color: white;
    border-radius: 5px; /* Slightly rounded corners for list items */
  }
  .autocom-box li:hover {
    background-color: rgba(255, 255, 255, 0.2);
  }
  .icon {
    position: absolute;
    right: 30px; /* Adjusted to keep icon inside the input */
    top: 50%;
    transform: translateY(-50%);
    cursor: pointer;
    color: white;
  }
  .icon i {
    font-size: 20px;
  }
  .info-boxes {
    position: absolute;
    top: calc(10px + 80px); /* Adjust based on search box height */
    left: 20px;
    right: 20px;
    display: flex;
    gap: 20px;
    z-index: 1001; /* Ensure the boxes are on top */
    transition: top 0.5s ease-in-out;
  }
  .info-box {
    flex: 1;
    height: calc(100vh - 150px); /* Adjust based on your layout */
    background: rgba(255, 255, 255, 0.1);
    backdrop-filter: blur(10px);
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    padding: 20px;
    box-sizing: border-box;
    color: white;
    overflow-y: auto;
  }
</style>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css"/>
<script type="importmap">
  {
    "imports": {
        "three": "https://cdn.jsdelivr.net/npm/three@0.161/build/three.module.js",
        "jsm/": "https://cdn.jsdelivr.net/npm/three@0.161/examples/jsm/",
        "GLTFLoader": "https://cdn.jsdelivr.net/npm/three@0.161/examples/jsm/loaders/GLTFLoader.js"
      }
    }
  </script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.9.1/gsap.min.js"></script>
  <script src="script.js"></script>
  <script src="suggestions.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/xlsx/0.17.0/xlsx.full.min.js"></script>
</head>
<body>
  <div class="navbar">
    <h1>AstroScope</h1>
    <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">About</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </div>
  <div class="intro-text" id="introText">
    <b><h1>Explore</h1></b>
    <h2>Near Earth-Objects</h2>
    <p>Search for the near earth satellites</p>
    <button id="exploreButton">Explore Now</button>
  </div>

  <!-- Search Box with Autocomplete -->
  <div class="search-container" id="searchBox">
    <div class="search-input">
      <input type="text" id="searchInput" placeholder="Search for satellites..." />
      <div class="icon"><i class="fas fa-search"></i></div>
    </div>
    <ul class="autocom-box">
      <!-- Autocomplete suggestions will be injected here -->
    </ul>
  </div>

  <div class="info-boxes" id="infoBoxes" style="display: none;">
    <div class="info-box left-box"></div>
    <div class="info-box right-box"></div>
  </div>

  <script type="module" src="index.js"></script>
  <!-- Remove these two lines as they're not needed and might cause conflicts -->
  <!-- <script src="js/suggestions.js"></script> -->
  <!-- <script src="js/script.js"></script> -->

  <script>
    // Fade in text after 5 seconds
    window.onload = function() {
      const introText = document.getElementById('introText');
      introText.style.opacity = 1;
    };

    // Handle explore button click
    document.getElementById('exploreButton').onclick = function() {
      document.getElementById('introText').style.display = 'none';

      // Trigger 360 rotation and search box slide-in
      window.dispatchEvent(new CustomEvent('exploreClicked'));
    };
  </script>
```

```
</body>
</html>
```

JS code:

```
import * as THREE from "three";
import { OrbitControls } from 'jsm/controls/OrbitControls.js';
import { GLTFLoader } from 'jsm/loaders/GLTFLoader.js';

import getStarfield from "./src/getStarfield.js";
import { getFresnelMat } from "./src/getFresnelMat.js";

const w = window.innerWidth;
const h = window.innerHeight;
const scene = new THREE.Scene();

// Initial Camera Setup (Earth centered)
const camera = new THREE.PerspectiveCamera(75, w / h, 0.1, 1000);
camera.position.set(0, 0, 3); // Start zoomed out and centered on Earth

const renderer = new THREE.WebGLRenderer({ antialias: true });
renderer.setSize(w, h);
document.body.appendChild(renderer.domElement);

renderer.toneMapping = THREE.ACESFilmicToneMapping;
renderer.outputColorSpace = THREE.LinearSRGBColorSpace;

const controls = new OrbitControls(camera, renderer.domElement);
controls.enableDamping = true;
controls.dampingFactor = 0.1;
controls.rotateSpeed = 0.1;
controls.enableZoom = false;  // Disable zoom scrolling

// Earth Group (Initially centered)
const earthGroup = new THREE.Group();
earthGroup.rotation.z = -23.4 * Math.PI / 180;  // Earth Tilt
scene.add(earthGroup);

const radius = 1;
const widthSegments = 64;
const heightSegments = 64;
const geometry = new THREE.SphereGeometry(radius, widthSegments, heightSegments);

// Earth Surface Material
const loader = new THREE.TextureLoader();
const earthMaterial = new THREE.MeshPhongMaterial({
  map: loader.load("./textures/Earth_Diffuse_6K.jpg"),
  specularMap: loader.load("./textures/Earth_Glossiness_6K.jpg"),
  bumpMap: loader.load("./textures/Earth_Bump_6K.jpg"),
  bumpScale: 0.04,
});
const earthMesh = new THREE.Mesh(geometry, earthMaterial);
earthGroup.add(earthMesh);

// Lights and Clouds on Earth
const lightsMat = new THREE.MeshBasicMaterial({
  map: loader.load("./textures/Earth_Lights_6K.jpg"),
  blending: THREE.AdditiveBlending,
});
const lightsMesh = new THREE.Mesh(geometry, lightsMat);
earthGroup.add(lightsMesh);

const cloudsMat = new THREE.MeshStandardMaterial({
  map: loader.load("./textures/Earth_Clouds_6K.jpg"),
  transparent: true,
  opacity: 0.8,
  blending: THREE.AdditiveBlending,
  alphaMap: loader.load('./textures/Earth_Clouds_6K.jpg'),
});
const cloudsMesh = new THREE.Mesh(geometry, cloudsMat);
cloudsMesh.scale.setScalar(1.003);
earthGroup.add(cloudsMesh);

// Halo and Glow Effects
const haloGeometry = new THREE.SphereGeometry(radius * 1.1, widthSegments, heightSegments);
const haloMaterial = new THREE.ShaderMaterial({
  uniforms: {
    viewVector: { type: "v3", value: camera.position },
    c: { type: "f", value: 0.8 },
    p: { type: "f", value: 2.0 }
  },
  vertexShader: `
    varying vec3 vNormal;
    void main() {
      vNormal = normalize(normalMatrix * normal);
      gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
    }
  `,
  fragmentShader: `
    uniform vec3 viewVector;
    uniform float c;
    uniform float p;
    varying vec3 vNormal;
    void main() {
      float intensity = pow(c - dot(vNormal, viewVector), p);
      gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0) * intensity;
    }
  `,
  side: THREE.BackSide,
  blending: THREE.AdditiveBlending,
  transparent: true
});
// const haloMesh = new THREE.Mesh(haloGeometry, haloMaterial);
// haloMesh.scale.setScalar(1.03);
// earthGroup.add(haloMesh);

const fresnelMat = getFresnelMat();
const glowMesh = new THREE.Mesh(geometry, fresnelMat);
glowMesh.scale.setScalar(1.01);
earthGroup.add(glowMesh);

// Starfield
const stars = getStarfield({ numStars: 2000 });
scene.add(stars);

// Sunlight - Day-Night Cycle Rotation
const sunLight = new THREE.DirectionalLight(0xffffff, 5.0);
sunLight.position.set(-2, 0.5, 1.5);
scene.add(sunLight);

let sunAngle = 0;
function updateSunPosition() {
  sunAngle += 0.001; // Control day-night cycle speed
  const x = Math.cos(sunAngle) * 5;
  const z = Math.sin(sunAngle) * 5;
  sunLight.position.set(x, 1, z);
```

```
}

// Initial Search Box Hidden
const searchBox = document.getElementById('searchBox');
searchBox.style.transition = 'left 0.5s';
searchBox.style.left = '-700px';  // Off-screen initially

function showSearchBox() {
  searchBox.style.left = '20px'; // Slide in after animation
}

// Satellite group (as before)
const satelliteLoader = new GLTFLoader();
const satelliteGroup = new THREE.Group();
earthGroup.add(satelliteGroup);

satelliteLoader.load('./simple_satellite_low_poly_free/scene.gltf', (gltf) => {
  const satelliteModel = gltf.scene;
  satelliteModel.scale.set(0.02, 0.02, 0.02);

  const numSatellites = 5;
  for (let i = 0; i < numSatellites; i++) {
    const satelliteClone = satelliteModel.clone();
    satelliteClone.position.set(
      Math.random() * 2 - 1,
      Math.random() * 2 - 1,
      Math.random() * 2 - 1
    ).normalize().multiplyScalar(radius * 1.5);

    satelliteClone.lookAt(earthMesh.position);
    satelliteGroup.add(satelliteClone);
  }
}, undefined, (error) => {
  console.error('An error occurred:', error);
});

// Asteroid group (as before)
const asteroidGeometry = new THREE.DodecahedronGeometry(0.02);
const asteroidMaterial = new THREE.MeshStandardMaterial({ color: 0x808080 });
const asteroidGroup = new THREE.Group();
earthGroup.add(asteroidGroup);

for (let i = 0; i < 10; i++) {
  const asteroidMesh = new THREE.Mesh(asteroidGeometry, asteroidMaterial);
  asteroidMesh.position.set(
    Math.random() * 2 - 1,
    Math.random() * 2 - 1,
    Math.random() * 2 - 1
  ).normalize().multiplyScalar(radius * 1.7);

  asteroidMesh.lookAt(earthMesh.position);
  asteroidGroup.add(asteroidMesh);
}

// Detect user interaction
let userIsInteracting = false;
controls.addEventListener('start', () => {
  userIsInteracting = true;
});
controls.addEventListener('end', () => {
  userIsInteracting = false;
});
```

```
// Camera 360° Turn and Zoom-in Animation After Clicking "Explore Now"
function animateCameraAroundEarth(callback) {
  const rotationDuration = 3000; // 3 seconds
  const initialPos = { x: 0, z: 3 };
  const finalPos = { x: -2, z: 1.5 }; // End up on the right side, zoomed in
  let startTime = null;

  function rotateAndZoom(time) {
    if (!startTime) startTime = time;
    const elapsed = time - startTime;
    const progress = Math.min(elapsed / rotationDuration, 1);  // Normalize progress

    // Camera rotates and zooms in around the Earth
    const angle = progress * Math.PI * 2; // 360° rotation
    const dist = THREE.MathUtils.lerp(initialPos.z, finalPos.z, progress);
    camera.position.x = Math.cos(angle) * dist;
    camera.position.z = Math.sin(angle) * dist;

    camera.fov = THREE.MathUtils.lerp(75, 75, progress);  // Zoom effect
    camera.lookAt(new THREE.Vector3(0, 0, 0)); // Look at the shifted Earth on the right
    camera.updateProjectionMatrix();

    if (progress < 1) {
      requestAnimationFrame(rotateAndZoom);
    } else {
      callback(); // Call after the camera animation is complete
    }
  }

  requestAnimationFrame(rotateAndZoom);
}

// Halo Fade-Out Effect
function fadeOutHalo() {
  const fadeDuration = 2000;
  const startTime = Date.now();

  function fade() {
    const elapsed = Date.now() - startTime;
    const fraction = elapsed / fadeDuration;
    haloMesh.material.opacity = 1.0 - fraction;

    if (fraction < 1.0) {
      requestAnimationFrame(fade);
    }
  }
  fade();
}

// Main Animation Loop
function animate() {
  requestAnimationFrame(animate);

  // Rotate Earth and simulate day-night cycle
  if (!userIsInteracting) {
    earthGroup.rotation.y += 0.001;
  }
  updateSunPosition();

  controls.update();
  renderer.render(scene, camera);
}
```

```
animate();

// Handle window resize
function handleWindowResize() {
  camera.aspect = window.innerWidth / window.innerHeight;
  camera.updateProjectionMatrix();
  renderer.setSize(window.innerWidth, window.innerHeight);
}
window.addEventListener('resize', handleWindowResize, false);

// "Explore Now" Button Event - Handles All Animations
window.addEventListener('exploreClicked', () => {
  const introText = document.getElementById('introText');
  introText.style.transition = 'opacity 2s';
  introText.style.opacity = 0;

  setTimeout(() => {
    // Start Camera 360 turn and zoom-in after text fades out
    animateCameraAroundEarth(() => {
      showSearchBox(); // Show the search bar after camera finishes moving
    });

    // Fade out halo during the camera animation
    fadeOutHalo();
  }, 2000); // Delay for intro text fade out
});

// Search Feature
const searchInput = document.querySelector("#searchInput");
const autocomBox = document.querySelector(".autocom-box");
const searchIcon = document.querySelector(".icon");
const infoBoxes = document.getElementById("infoBoxes");

let satellites = []; // This will store our satellite data

// Function to read XLSX file
function readXlsxFile(file) {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.onload = (e) => {
      try {
        const data = new Uint8Array(e.target.result);
        const workbook = XLSX.read(data, {type: 'array'});
        const firstSheetName = workbook.SheetNames[0];
        const worksheet = workbook.Sheets[firstSheetName];
        const jsonData = XLSX.utils.sheet_to_json(worksheet);
        resolve(jsonData);
      } catch (error) {
        console.error('Error processing XLSX file:', error);
        reject(error);
      }
    };
    reader.onerror = (error) => reject(error);
    reader.readAsArrayBuffer(file);
  });
}

// Function to load satellite data
async function loadSatelliteData() {
  try {
    const response = await fetch('comet.xlsx');
    const blob = await response.blob();
    satellites = await readXlsxFile(blob);
    console.log('Satellite data loaded:', satellites.slice(0, 5)); // Log first 5 satellites
    console.log('Total satellites loaded:', satellites.length);
  } catch (error) {
    console.error('Error loading satellite data:', error);
  }
}

// Call this function when the page loads
document.addEventListener('DOMContentLoaded', loadSatelliteData);

function handleSearch() {
  const inputValue = searchInput.value.trim();

  console.log("Search input value:", inputValue);

  const matchedSatellite = satellites.find(sat =>
    sat['Satellite Name'].toLowerCase() === inputValue.toLowerCase()
  );

  console.log("Matched satellite:", matchedSatellite);

  if (matchedSatellite) {
    // Always keep search box at top
    document.querySelector(".search-container").classList.add('top');

    // Show and update info boxes
    infoBoxes.style.display = 'flex';

    const leftBox = document.querySelector('.left-box');
    const rightBox = document.querySelector('.right-box');

    leftBox.innerHTML = `
      <h2>Satellite Data</h2>
      <p><strong>Name:</strong> ${matchedSatellite['Satellite Name'] || 'N/A'}</p>
      <p><strong>Discovered:</strong> ${matchedSatellite.Discovered || 'N/A'}</p>
      <p><strong>Diameter:</strong> ${matchedSatellite.Diameter || 'N/A'}</p>
      <p><strong>Mass:</strong> ${matchedSatellite.Mass || 'N/A'}</p>
      <p><strong>Close Approach:</strong> ${matchedSatellite['Close Approach'] || 'N/A'}</p>
      <p><strong>Impact Risk:</strong> ${matchedSatellite['Impact Risk'] || 'N/A'}</p>
      <p><strong>Mission:</strong> ${matchedSatellite.Mission || 'N/A'}</p>
      <p><strong>Facts:</strong> ${matchedSatellite.Facts || 'N/A'}</p>
      <p><strong>Composition:</strong> ${matchedSatellite.Composition || 'N/A'}</p>
    `;

    rightBox.innerHTML = '<h2>Additional Information</h2><p>Add any extra details or visualizations here.</p>';
  } else {
    // Keep search box at top, but hide info boxes if no match found
    infoBoxes.style.display = 'none';
    console.log("No match found, boxes hidden");
  }
}

if (searchInput && autocomBox) {
  searchInput.addEventListener('input', function() {
    const query = this.value.toLowerCase();
    autocomBox.innerHTML = ''; // Clear previous results

    console.log("Input query:", query);
    console.log("Number of satellites:", satellites.length);

    if (query && satellites.length > 0) {
      const filteredSatellites = satellites.filter(satellite =>
        satellite['Satellite Name'] && satellite['Satellite Name'].toLowerCase().includes(query)
      );

      console.log("Filtered satellites:", filteredSatellites);

      if (filteredSatellites.length > 0) {
        autocomBox.style.display = 'block';
        filteredSatellites.forEach(satellite => {
          const li = document.createElement('li');
          li.textContent = satellite['Satellite Name'];
          li.addEventListener('click', function() {
            searchInput.value = satellite['Satellite Name'];
            autocomBox.style.display = 'none';
            handleSearch(); // Trigger the search function to show details
          });
          autocomBox.appendChild(li);
        });
      } else {
        autocomBox.style.display = 'none';
      }
    } else {
      autocomBox.style.display = 'none';
    }
  });
}

// Add click event listener to the search icon
searchIcon.addEventListener('click', handleSearch);

// Add keypress event listener to the search input for Enter key
searchInput.addEventListener('keypress', function(e) {
  if (e.key === 'Enter') {
    handleSearch();
  }
});

// Close autocomplete box when clicking outside
document.addEventListener('click', (e) => {
  if (!searchInput.contains(e.target) && !autocomBox.contains(e.target)) {
    autocomBox.style.display = 'none';
  }
});
```