# Version\_3:

**Map.html:**  
Certainly! Let's break down the HTML and associated JavaScript in the provided code:

### HTML Structure:

1. Document Type and Language:

- `<!DOCTYPE html>` declares the document type as HTML5.

- `<html lang="en">` specifies the document language as English.

2. Head Section:

- Title and Meta Tags:

- `<title>Total\_Schools</title>` sets the title of the webpage.

- `<meta charset="UTF-8">`, `<meta http-equiv="X-UA-Compatible" content="IE=edge">`, and `<meta name="viewport" content="width=device-width, initial-scale=1.0">` define character set and viewport settings for better responsiveness.

- Bootstrap and CSS/JS Libraries:

- Links to Bootstrap CSS and JavaScript (`bootstrap.min.css` and `bootstrap.bundle.min.js`) for styling and JavaScript functionality.

- `<link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css" />` and `<script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>` link to Leaflet CSS and JavaScript for interactive maps.

- `<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>` includes Chart.js for data visualization.

- `<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">` provides Font Awesome for icons.

3. Body Section:

- Navbar (`nav`):

- Implements a Bootstrap navbar (`navbar-expand-md navbar-dark bg-dark`) with a toggle button (`navbar-toggler`) for mobile responsiveness.

- Includes a brand link (`<a class="navbar-brand" href="#">Geo-Portal</a>`) and dropdown menu (`<li class="nav-item dropdown">`) with various links (`<a class="nav-link active" href="#" onclick="Home\_button()">Home</a>`).

- Sidebar and Content (`d-flex flex-row position-relative`):

- Sidebar (`left\_panel`):

- Contains various form elements and dropdowns (`<select>` and `<input>`) for selecting maps, seasons, and filtering data by state and district.

- Uses Bootstrap classes (`form-group`, `dropdown`, `dropdown-toggle`, `dropdown-menu`) for styling and dropdown functionality.

- Main Content Area:

- Leaflet Map (`<div id="map" class="map" style="flex-grow: 1; height: 90vh;"></div>`):

- Placeholder for the Leaflet map with initial styling and dimensions (`flex-grow: 1; height: 90vh;`) to occupy most of the viewport height (`90vh`).

- Chart Container (`<div id="chart-container" class="chart-container" style="display:none; flex: 1;">`):

- Container for Chart.js visualization (`<canvas id="barChart"></canvas>`).

- Initially hidden (`style="display:none;"`) and set to occupy remaining space (`flex: 1;`) when displayed.

4. JavaScript Files:

- Includes multiple JavaScript files (`Script.js`, `State.js`, `District.js`, `Legend.js`, `Charts.js`, `Click.js`) at the end of the body section for functionality like map interactions, state and district data handling, legends, charts, and click events.

**Script.js:**  
This JavaScript code initializes and enhances a Leaflet map embedded in an HTML page, providing interactive mapping functionalities along with dynamic data handling:

Key Components and Functionality:

1. Tile Layers Initialization:

- Three tile layers (`osm`, `osmHOT`, `openTopoMap`) from OpenStreetMap and OpenTopoMap are defined with specific URLs and attribution details. These layers provide different map styles (standard, humanitarian, and topographic).

2. Map Initialization:

- `map` object is initialized using Leaflet's `L.map` function, centered at `[20.5937, 81.9629]` with an initial zoom level of `5`. The `osm` layer is set as the default base layer.

3. Map Controls and Interactivity:

- Max Bounds: Defines geographical bounds (`bounds`) to restrict panning beyond specified coordinates (`southWest` and `northEast`).

- Max Zoom: Limits maximum zoom level based on the defined bounds to prevent overzooming.

- Layer Control: Enables switching between different base map layers (`baseMaps`) using `L.control.layers`.

- Scale Control: Adds a scale control (`L.control.scale`) at the bottom left corner of the map.

4. Event Handling:

- Home\_button Function: Resets the map view to the default center and zoom level (`[20.5937, 81.9629], 5`), removes any added layers (`wmsLayer` and `legend`), and resets dropdown selection. It also closes any open pop-ups and hides the chart if visible.

- Mousemove Event: Displays latitude and longitude coordinates (`e.latlng.lat.toFixed(4)` and `e.latlng.lng.toFixed(4)`) dynamically as the mouse moves over the map.

5. Dropdown Event Listener:

- Detects changes in the dropdown selection (`layers`). Based on the selected value (`'Health'` or `'District'`), it removes current layers, resets the map view, adds respective layers (`addHealthLayer()` or `addDistrictLayer()`), and fetches corresponding state data (`fetchStates()`).

6. Chart Toggle Function:

- `toggleChart()` toggles the visibility of a chart container (`chart-container`) when clicking on a button (`toggleChartButton`). It dynamically changes the button's text and icon (`Show Chart` or `Hide Chart`).

7. Custom Controls:

- Coordinate Display Control: Creates a custom control (`coordinateDisplayControl`) that dynamically displays mouse coordinates (`'Lat: ' + e.latlng.lat.toFixed(4) + ' , Long: ' + e.latlng.lng.toFixed(4)`).

**State.js:**

1. Initial Setup and Layers:

- Initializes a WMS tile layer (`L.tileLayer.wms`) from a GeoServer endpoint (`http://localhost:8080/geoserver/India/wms`). This layer (`Schools\_States`) is for displaying state boundaries with specific styling options.

- Calls `fetchNationalLevelData()` to fetch national-level data for further processing.

2. Fetch States Function:

- Fetches state data (`State` property) from a GeoServer WFS endpoint (`http://localhost:8080/geoserver/India/wfs`). Converts the fetched data into JSON format.

- Processes the data to extract unique state names (`states`), which are then populated dynamically into a dropdown menu (`#stateDropdownMenu`).

- Each dropdown item (`<a>` tag) is created with an event listener that updates the `selectedState` variable on click, triggers filtering (`filterByState()`), zooms the map to the selected state (`zoomToState(state)`), and adds a health legend (`addHealthLegend()`).

3. Map Interaction Functions:

- zoomToState(stateName): Fetches and sets the map bounds for the selected state (`stateName`) using a WFS request to GeoServer. Alerts the user if no data is available for the selected state.

- filterByState(): Filters map data (`wmsLayer`) based on the selected state (`selectedState`). If `wmsLayer` doesn't exist, it creates it using `L.tileLayer.wms` and sets parameters for filtering. Also calls `fetchDistricts(selectedState)` to fetch district-level data.

4. Dropdown and User Interaction:

- Uses `DOMContentLoaded` event listener to ensure JavaScript executes after the HTML document is fully loaded.

- Event listeners and functions handle dropdown interactions (`stateDropdown` and `stateDropdownMenu`) and search functionality (`stateSearchInput`).

5. Data Fetching and Chart Generation:

- fetchNationalLevelData(): Fetches national-level data (`State` and `ud\_2122\_1` properties) from GeoServer WFS endpoint. Processes the data to aggregate school counts (`ud\_2122\_1`) by state (`aggregateDataByState()`), then generates a national-level bar chart (`generateBarChart(Data)`).

- fetchStateLevelData(): Fetches state-level data (`District` and `ud\_2122\_1` properties) based on the selected state (`selectedState`). Processes the data and generates or updates a state-level bar chart (`generateBarChart(stateData)` and `updateBarChart(stateData)`).

6. Chart Generation and Update:

- generateBarChart(data): Generates a bar chart using Chart.js library (`new Chart(ctx, {...})`). Configures the chart with labels (`labels`), dataset (`datasets`) for school counts (`ud\_2122\_1`), and styling options (`backgroundColor`, `borderColor`, etc.).

- updateBarChart(stateData): Updates an existing bar chart (`barChart`) with new data (`stateData`).

**District.js:**

1. Initialization and Variables:

- districtLayer: Initializes a GeoJSON layer (`L.geoJSON()`) without any initial data and adds it to the `map` object.

- selectedDistrict: Initializes a variable to store the currently selected district (`null` initially).

2. Highlight Style Definition:

- highlightStyle: Defines a style object (`highlightStyle`) with properties (`color`, `weight`, `opacity`) for highlighting a selected district on the map.

3. Fetch Districts Function:

- fetchDistricts(selectedState): Fetches district data (`District` property) from a GeoServer WFS endpoint (`http://localhost:8080/geoserver/India/wfs`). It filters the data based on the selected state (`selectedState`).

- Processes the fetched data to extract unique district names (`districts`) and populates them dynamically into a dropdown menu (`#districtDropdownMenu`).

- Each dropdown item (`<a>` tag) is created with a click event listener that updates the `selectedDistrict` variable and calls `zoomToDistrict(District)` to zoom the map to the selected district.

4. Zoom to District Function:

- zoomToDistrict(districtName): Uses a WFS GetFeature request to fetch the bounds of the selected district (`districtName`) from GeoServer.

- Clears any existing layers (`districtLayer.clearLayers()`) and adds a new GeoJSON layer (`L.geoJSON(data, { style: highlightStyle })`) with the specified `highlightStyle` to visually highlight the selected district on the map.

- Fits the map to the bounds of the selected district for better visibility (`map.fitBounds(bounds)`).

5. Dropdown and Search Interaction:

- Uses `DOMContentLoaded` event listener to execute JavaScript once the HTML document is fully loaded.

- truncateText(text, maxLength): Helper function to truncate text if it exceeds a specified maximum length (`maxLength`).

- Event Listeners: Handles dropdown menu interactions (`districtDropdownMenu`) and search input (`districtSearchInput`).

- Updates the dropdown button text (`districtDropdownButton.text(truncatedText)`) with the selected district.

- Filters dropdown items based on user input (`searchTerm`) using regular expressions (`regex.test(text)`).

- Displays a message (`noDataItem`) if no matching items are found (`matchingItems.length === 0`).

**Legend.js:**

1. Initialization:

- legend: Initializes a variable (`legend`) to store the legend control for the map. It's initially set to `null`.

2. Function to Add Legend (`addHealthLegend`):

- addHealthLegend(wmsUrl, layerName): This function creates and adds a legend control to the map (`map`). It takes two parameters:

- `wmsUrl`: URL of the GeoServer WMS service (`"http://localhost:8080/geoserver/India/wms"` in this example).

- `layerName`: Name of the layer for which the legend is generated (`"India:Total\_Schools"` in this example).

3. Legend Control Creation:

- Creates a `legend` control using `L.control({ position: 'bottomright' })`, positioning it at the bottom-right corner of the map.

4. onAdd Method of Legend Control:

- legend.onAdd = function (map): This method is called when the legend control is added to the map.

- Creates a `<div>` element (`div`) with the class `legend-container`.

- Adds a title (`Legend`) to the legend container (`div.innerHTML += '<div class="legend-title">Legend</div>'`).

5. Constructing Legend URL:

- Constructs the URL for the legend image (`legendUrl`) using parameters:

- `wmsUrl`: Base URL of the GeoServer WMS service.

- `'REQUEST=GetLegendGraphic'`: Request type to get the legend image.

- `'VERSION=1.0.0'`: Version of the WMS service.

- `'FORMAT=image/png'`: Format of the legend image.

- `'LAYER=' + layerName`: Specific layer for which the legend is requested (`layerName`).

6. Adding Legend Image:

- Appends the legend image (`<img>`) to the legend container (`div.innerHTML += '<div class="legend"><img src="' + legendUrl + '" alt="Legend"></div>'`).

7. Returning the Div:

- Returns the populated `div` containing the legend HTML elements.

8. Adding Legend to the Map:

- legend.addTo(map): Finally, adds the constructed legend control (`legend`) to the `map`.

**Click.js:**This code snippet is a JavaScript application using Leaflet and GeoServer to interact with geospatial data related to schools in India. It provides functionalities to dynamically fetch and display information based on user interactions such as clicking on the map and selecting options from dropdown menus. Here's an explanation of its key parts:

1. Initialization and Variables:

- highlightStyle: Defines the style (`color`, `weight`, `opacity`) for highlighting districts on the map.

- selectedState: Stores the currently selected state.

- selectedDistrict: Stores the currently selected district.

- districtLayer: Initializes an empty GeoJSON layer (`L.geoJSON()`) which will be used to display district boundaries on the map.

- wmsLayer: Initially `null`, this variable will hold the WMS layer for displaying school data.

- legendAdded: Flag to track whether the legend has been added to the map (`false` initially).

2. Map Click Event Listener:

- Adds a click event listener to the map (`map.on('click', function (e) { ... })`).

- Fetch State Information: Retrieves state information based on the clicked point's coordinates using a WFS request (`fetch(stateUrl)`).

- Updates `selectedState` if a state is found.

- Clears any highlighted districts (`districtLayer.clearLayers()`) and updates the map view to focus on the selected state (`zoomToState(stateName)`).

- Fetches and updates state-level data (`fetchStateLevelData()`).

- Updates the state dropdown menu (`updateStateDropdown(stateName)`).

- Fetch District Information: If the same state is clicked again, fetches district information based on the clicked point's coordinates within the state (`fetchDistrictInformation(latlng)`).

- Updates `selectedDistrict`.

- Zooms to the selected district (`zoomToDistrict(districtName)`).

- Displays a popup with detailed information about the district.

3. Legend Control:

- Checks if the legend has already been added (`!legendAdded`).

- If not, adds the health legend (`addHealthLegend(wmsUrl, layerName)`) using the provided WMS URL and layer name.

- Sets `legendAdded` to `true` once the legend is added.

4. Functions:

- fetchDistrictInformation(latlng): Fetches district information based on the clicked coordinates, updating `selectedDistrict`, zooming to the district, and displaying a popup with district details.

- filterByState(): Filters and updates the WMS layer to display school data for the selected state.

- zoomToState(stateName): Zooms the map view to fit the boundaries of the selected state.

- fetchStateLevelData(): Fetches and processes state-level data, updating and generating charts (`generateBarChart()` and `updateBarChart()`).

- zoomToDistrict(districtName): Zooms the map view to fit the boundaries of the selected district, highlighting it with `highlightStyle`.

- updateStateDropdown(stateName): Updates the state dropdown menu with the selected state.

- updateDistrictDropdown(districtName): Updates the district dropdown menu with the selected district.