# Version\_2:

**Health.html:**  
This HTML code snippet outlines the structure of a Geo-Portal web application. It includes essential elements for navigation, interactive map display, and data visualization using charts. The document starts with standard meta tags and links to external resources like Bootstrap for styling and JavaScript libraries such as Leaflet for mapping and Chart.js for charting.

The navigation bar (`<nav>`) provides links to different sections of the portal, including dropdown menus for selecting various data categories like health centers and districts. A sidebar (`<div class="left\_panel">`) houses form elements for selecting maps, seasons, and months, along with filters for searching states and districts dynamically.

The main content area includes a large map container (`<div id="map">`) styled to occupy a significant portion of the viewport. This area is powered by Leaflet.js, enabling interactive mapping functionalities. Adjacent to it, a chart container (`<div id="chart-container">`) is initially hidden but ready to display charts created with Chart.js based on user-selected data.

Custom JavaScript files (`Script.js`, `State.js`, `District(D).js`, etc.) handle dynamic dropdown population, map interactions, and chart rendering based on user inputs, enhancing the application's interactivity and usability. Overall, this HTML structure supports a comprehensive Geo-Portal experience for exploring and visualizing geographic data interactively.

**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:**

Key Components and Functionality:

1. Health Layer (`addHealthLayer()` Function):

- WMS Layer Initialization: Adds a WMS tile layer (`wmsLayer`) to the map using GeoServer's endpoint (`http://localhost:8080/geoserver/India/wms`).

- Layer Configuration: Specifies layer details such as `layers`, `format`, `transparent`, `version`, and `opacity`.

- Layer Addition: Once initialized, the layer is added to the Leaflet map (`map`) and brought to the front (`bringToFront()`).

- Legend Addition: Calls `addHealthLegend()` function to add a legend control specific to the health data layer.

-Data Fetching: Invokes `fetchNationalLevelData()` to retrieve and aggregate national-level health center data.

2. State and District Handling:

- Dropdown Population (`fetchStates()`):

- Uses GeoServer's WFS service (`http://localhost:8080/geoserver/India/wfs`) to fetch states (`state`) from the `Health2` layer.

- Populates a dropdown menu (`stateDropdownMenu`) dynamically with state names fetched from the data.

- Allows selecting a state, storing it in `selectedState`, and triggering functions (`filterByState()` and `zoomToState(state)`) accordingly.

3. Map Interaction and Information Retrieval:

- Map Click Event (`map.on('click')`):

- Retrieves information (`ID`, `District`, `State`, `No. of Health Centres`) about a clicked point (`latlng`) on the map.

- Uses GeoServer's WFS service to filter data based on the clicked point's coordinates and the selected state (`selectedState`).

- Displays retrieved information in a popup (`L.popup()`).

4. Dropdown and Search Functionality:

- Dropdown Interaction (`stateDropdownMenu.on('click', '.dropdown-item')`):

- Updates the dropdown button text with the selected state name.

- Fetches state-level data (`fetchStateLevelData()`) and generates a bar chart (`generateBarChart()`).

5. Chart Generation and Update:

- Bar Chart (`generateBarChart(data)`):

- Uses Chart.js to create a bar chart (`'bar'` type) within the specified `ctx` (canvas context).

- Dynamically updates with data (`labels` and `datasets`) retrieved from the GeoServer (`Health2` layer) for state or district (`lgd\_distri`) and health center count (`HC\_Count`).

**District.js:**

Key Components and Functionality:

1. District Layer and Variables:

- `districtLayer` and `selectedDistrict`:

- `districtLayer`: Initializes a GeoJSON layer (`L.geoJSON()`) to hold district boundary data, added to the Leaflet map (`map`).

- `selectedDistrict`: Variable initialized to store the currently selected district.

2. Fetching and Populating District Dropdown (`fetchDistricts(selectedState)`):

- Uses GeoServer's WFS service (`http://localhost:8080/geoserver/India/wfs`) to fetch district names (`lgd\_distri`) for the selected state.

- Processes the fetched data to remove duplicates and populates a dropdown menu (`districtDropdownMenu`) dynamically.

- Each dropdown item (`dropdownItem`) is created as an `<a>` element with a click event listener to store the selected district (`selectedDistrict`) and trigger actions (`filterByDistrict()` and `zoomToDistrict(district)`).

3. Zooming to Selected District (`zoomToDistrict(districtName)`):

- Uses a WFS GetFeature request to retrieve the bounds (`getBounds()`) of the selected district (`districtName`).

- Fits the map (`map.fitBounds(bounds)`) to the retrieved bounds with some padding to ensure visibility.

4. Filtering Data by Selected District (`filterByDistrict()`):

- If `selectedDistrict` is set, constructs a query (`query`) to filter the WMS layer (`wmsLayer`) by `lgd\_distri`.

- Updates the parameters (`setParams({ cql\_filter: query })`) of `wmsLayer` to reflect the selected district.

5. Dropdown and Search Functionality:

- Dropdown Interaction (`districtDropdownMenu.on('click', '.dropdown-item')`):

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

- Ensures the dropdown displays truncated text if it exceeds a maximum length (`maxTextLength`).

- Search Input (`districtSearchInput.on('input')`):

- Handles user input to filter dropdown items based on district names (`text.trim().toLowerCase()`).

- Uses regular expressions (`regex`) to match and display items (`item.css('display', match ? 'block' : 'none')`).

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

The State(D).js and District(D).js has the same functionalities as the State.js and District.js only with a different dataset.