# Version\_1:

This version of the Geo\_portal contains 5 sets of codes. Four of them is Javascrit code which supports the HTML code.

**Dashboard.html:**  
The provided HTML code outlines a webpage for a "Geo-portal" platform featuring a navigation bar, a sidebar for selecting filters, and an interactive map. The navigation bar includes a title, a home link, additional links, and a dropdown menu. The sidebar allows users to filter data by selecting a year, season, month, state, district, and sub-district, with dynamically populated dropdown menus. The main content area contains a Leaflet map, which can switch between a basemap and satellite view using a button. A coordinate display box shows map coordinates. The page incorporates Bootstrap for styling and layout, and Leaflet for map functionalities, with additional JavaScript files (`Dashboard.js`, `State.js`, `District.js`, `Sub-district.js`) handling interactive features and data filtering logic.

**Dashboard.js:**  
This JavaScript code initializes and manages a Leaflet map with toggling functionality between street and satellite views, displaying coordinates, and handling click events for spatial data queries.

Firstly, the map is initialized with a view centered on India ([20.5937, 78.9629]) and a zoom level of 4. Two tile layers are defined: a street map from OpenStreetMap and a satellite map styled by the Humanitarian OpenStreetMap Team. The street layer is added to the map by default. The `toggleMap` function allows switching between the street and satellite layers.

The `resetMapView` function resets the map view to the initial center and zoom level. When the mouse moves over the map, the `mousemove` event updates a coordinate display with the current latitude and longitude.

Upon clicking the map, a function is triggered to perform a WFS GetFeature request to a GeoServer instance, querying for district boundary information at the clicked location, filtered by a selected state. The response, containing district and state information, is used to display a popup at the clicked point with details of the sub-district, district, and state. This popup is set to fade out after 9 seconds using CSS transitions.

Any previous district layers are cleared before displaying new information, ensuring the map remains up-to-date with the latest query results.

**State.js:**

This JavaScript code integrates with a Leaflet map to add WMS layers, fetch state data, filter map data based on selected states, and enhance the state dropdown functionality with search and truncation features.

1. Map and Layers Initialization:

- `wmsLayer` is added to the map to display district boundaries from a GeoServer WMS.

- `districtLayer` is initialized as an empty GeoJSON layer.

- `selectedState` is initialized as `null` to store the state selected by the user.

2. Fetching States:

- The `fetchStates` function sends a WFS GetFeature request to the GeoServer to retrieve unique states from the district boundaries dataset.

- The dropdown menu (`#stateDropdownMenu`) is populated with the fetched state names. Clicking a state name sets `selectedState`, filters the map by the selected state, and zooms to the state bounds.

3. Zoom to State:

- The `zoomToState` function sends a WFS GetFeature request to get the geographical bounds of the selected state. The map view is then adjusted to fit these bounds.

4. Filtering by State:

- The `filterByState` function updates the WMS layer to show only the data for the selected state using a CQL filter.

- The function also fetches districts for the selected state.

5. Dropdown Menu Interaction:

- Event listeners are added for the state dropdown menu and search input field.

- Clicking on a dropdown item updates the dropdown button text with a truncated version of the selected state's name.

- The search input field filters the dropdown items based on the user's input.

- If no items match the search term, a "No data" message is displayed.

6. Truncation and Search Functionality:

- The `truncateText` function shortens the text if it exceeds a specified maximum length.

- The search functionality uses a regular expression to match dropdown items against the search term, displaying only the matching items.

7. Event Handling:

- The `DOMContentLoaded` event ensures that the dropdown menu is interactive once the page content is fully loaded.

- Event listeners for the dropdown button and search input enhance the user experience by enabling focus and real-time filtering.

Overall, this code manages a dynamic and interactive map interface, allowing users to filter and view spatial data by state, with enhanced usability features for the dropdown menu.

**District.js:**

**District Layer and Variable Initialization**

- District Layer: A `districtLayer` is added to the map to display GeoJSON data, allowing you to visualize district boundaries.

- Selected District Variable: A variable `selectedDistrict` is initialized to `null` to store the district selected by the user from the dropdown menu.

1. Fetching Districts

- Fetching District Data: The `fetchDistricts` function sends a WFS GetFeature request to the GeoServer to retrieve unique district names within the selected state from the district boundaries dataset.

- Populating the Dropdown Menu: The function then populates the dropdown menu (`#districtDropdownMenu`) with the fetched district names. When a district name is clicked, it sets `selectedDistrict`, filters the map by the selected district, and zooms to the district bounds.

2. Zoom to District

- Fetching District Bounds: The `zoomToDistrict` function sends a WFS GetFeature request to get the geographical bounds of the selected district.

- Adjusting Map View: The map view is adjusted to fit the bounds of the selected district, with some padding added for better visualization.

3. Filtering by District

- Updating WMS Layer: The `filterByDistrict` function updates the WMS layer to show only the data for the selected district using a CQL filter.

- Fetching Subdistricts: The function also fetches subdistricts for the selected district, allowing further drilling down into the geographical data.

4. Dropdown Menu Interaction

- Event Listeners for Dropdown Menu and Search Input: Event listeners are added for the district dropdown menu and search input field to handle user interactions.

- Updating Button Text: Clicking on a dropdown item updates the dropdown button text with a truncated version of the selected district's name.

- Filtering Dropdown Items: The search input field filters the dropdown items based on the user's input in real-time.

- Displaying No Data Message: If no items match the search term, a "No data" message is displayed.

5. Truncation and Search Functionality

- Text Truncation: The `truncateText` function shortens the text if it exceeds a specified maximum length, ensuring that the dropdown button text remains concise.

- Real-Time Filtering: The search functionality uses a regular expression to match dropdown items against the search term, displaying only the matching items dynamically.

6. Event Handling

- DOMContentLoaded Event: The `DOMContentLoaded` event ensures that the dropdown menu is interactive once the page content is fully loaded, making sure all elements are ready for interaction.

- Focus and Filtering: Event listeners for the dropdown button and search input enhance the user experience by enabling focus on the search input and real-time filtering of dropdown items.

**Sub-district.js:**

**Subdistrict Layer and Variable Initialization**

- Subdistrict Layer: A `subdistrictLayer` is initialized as a GeoJSON layer to display subdistrict boundaries on the map.

- Selected Subdistrict Variable: `selectedSubDistrict` is initialized to `null` to store the subdistrict selected by the user from the dropdown menu.

1. Fetching Subdistricts

- Fetching Subdistrict Data: The `fetchSubdistricts` function sends a WFS GetFeature request to GeoServer to retrieve unique subdistrict names within the selected district from the subdistrict boundaries dataset.

- Populating the Dropdown Menu: Upon receiving data, the function populates the dropdown menu (`#subdistrictDropdownMenu`) with the fetched subdistrict names. When a subdistrict name is clicked, it sets `selectedSubDistrict`, filters the map by the selected subdistrict, and zooms to its bounds.

2. Zoom to Subdistrict

- Fetching Subdistrict Bounds: The `zoomToSubDistrict` function sends a WFS GetFeature request to retrieve the geographical bounds of the selected subdistrict.

- Adjusting Map View: If data is found, the map view is adjusted to fit the bounds of the selected subdistrict, providing a focused view. If no features are found, an alert informs the user.

3. Filtering by Subdistrict

- Updating WMS Layer: The `filterBySubDistrict` function updates the WMS layer to show only the data for the selected subdistrict using a CQL filter. This ensures that only relevant data is displayed on the map based on user selection.

4. Dropdown Menu Interaction

- Event Listeners for Dropdown Menu and Search Input: Event listeners are added for the subdistrict dropdown menu (`#subdistrictDropdownMenu`) and search input field (`#subdistrictSearchInput`) to handle user interactions.

- Updating Button Text: Clicking on a dropdown item updates the dropdown button text (`subdistrictDropdownButton`) with the selected subdistrict's name, truncated if necessary to fit the UI.

- Filtering Dropdown Items: The search input field dynamically filters dropdown items based on user input, ensuring only matching subdistricts are displayed. If no matches are found, a "No data" message is shown (`#noDatasubDistrictItem`).

5. Truncation and Search Functionality

- Text Truncation: The `truncateText` function shortens the text if it exceeds a specified maximum length (`maxTextLength`), ensuring a clean UI display.

- Real-Time Filtering: The search functionality uses a regular expression (`regex`) to match subdistrict names against the search term (`searchTerm`), dynamically showing or hiding dropdown items based on the match.

6. Event Handling

- DOMContentLoaded Event: The `DOMContentLoaded` event ensures that the dropdown menu and search functionality are initialized only after the page content is fully loaded, optimizing user interaction and preventing premature UI rendering issues.

- Focus and Filtering: Event listeners for the dropdown button (`subdistrictDropdownButton`) and search input (`subdistrictSearchInput`) enhance user experience by enabling focus on the search input for immediate interaction.