GUI (5 required)

1. **Display zoomable, pannable map of the United States. (required) (sequence diagram not required)**

   Display a map of the United States. The map should clearly indicate all the states selected by your team. Ensure each state is highlighted using a different color for clear demarcation if neighboring states

2. **Select state to display (required)**

   User can pick a state through a dropdown menu and also through clicking on the state in the map of the US.

3. **Map view filter (required) (sequence diagram not required)**

   The GUI should employ a filter that allows some geographic features to be displayed (or not displayed). The user can set the viewing filter through checkboxes in the GUI.

4. **Display congressional district boundaries (preferred)**

   Enable/disable a viewing filter for congressional district boundaries. When the congressional district filter is selected, the map displays the states' current congressional district boundaries for the selected state, overlaid on any other data currently being displayed.

5. **Selectively display precinct borders of selected state. (required) (sequence diagram not required)**

   The map displays the precinct borders of selected states selectively in response to a user initiated event. This feature is desired to avoid any delays in displaying precinct boundary data. The event might be a click on a congressional district, which causes the precincts in that district to be displayed. This use case implies incremental loading of the precinct data in the client. No precinct loading should occur until the user selects the state. Precinct display should also be included in the list of filter checkboxes.

6. **Display neighbors of a precinct (required) (sequence diagram not required if implemented in the client)**

   In response to the user selecting a district, the system will identify the graph connected neighbors of the selected precinct. For example, you might highlight the borders of the neighbor precincts or change the color of the neighbor precincts.

7. **Display voting and demographic data on hover or click. (preferred) (sequence diagram not required)**

   When the user hovers the selector over a precinct or selects the precinct, its demographic data and voting data for the elections selected in the GUI (2016 Congressional, 2018 Congressional, or 2016 Presidential) become visible. Data should display in a fixed area of the screen (not floating near the hover location). At a minimum, demographic data shall include population by US Census category. Large integers (e.g., population) should be displayed with the comma integer notation.

8. **Highlight map changes (preferred) (sequence diagram not required)**

   Allow the user to toggle the display between the original state map and the modified map. You can choose to display the modified map in full or just the changes from the original map.

9. **Display National Park boundaries(preferred) (sequence diagram not required)**

   Enable/disable a viewing filter for National Park geographic boundaries. The user would employ this feature to understand the reason for anomalous data in a precinct. The user could subsequently add a comment to the precinct.

❖ Preprocessing (10 required) (Sequence diagrams not required)

10. **Integrate multiple data sources (required)**

    Integrate and merge US Census data, precinct data, demographic data, state, precinct, and district geographical data and election results data. Geographic boundary data should be in a consistent format (e.g., GeoJSON).

11. **Precinct renaming (required)**

    Rename precinct names to follow a standard naming scheme using a canonical name to meaningfully represent the data. Ensure that the original name and the canonical name are both included in the DB.

12. **Develop a data-error data structure and DB table(s) (required)**

    Develop a data structure that can store all the information required to identify data errors. The DB should be designed to allow storage of this data.

13. **Store preprocessed data in DB (required)**

    The preprocessed data defined should be stored in the DB for subsequent access by the client/server part of the code.

14. **Identify enclosed precincts (required) (sequence diagram not required)**

    Identify all precincts totally enclosed geographically by another precinct. Note that the enclosing precinct might be a single polygon or might be a multi-polygon that defines a hole in the precinct, Look for an enclosed precinct to be contained in the hole. Store the associated data in the errors data structure for subsequent addition to the DB.

15. **Identify overlapping precincts (required)**

    Identify pairs of precincts for which one or more edges of one precinct intersect with edges of the other precinct. Do not identify abutting edges where the edges represent a common border between the precincts. Store the associated data in the errors data structure for subsequent addition to the DB.

16. **Identify possible map coverage errors (required)**

    Identify the geographic area defined as the difference between the geographic area covered by the state and the geographic area covered by the union of all precincts in the state. Store the associated data in the errors data structure for subsequent addition to the DB.

17. **Identify unclosed polygon precinct data (preferred)**

    Identify any precinct boundary data for which the polygon is not closed. Modify the data to close the polygon and classify it as a possible data error for the user to confirm the correction. Store the associated data in the errors data structure for subsequent addition to the DB.

18. **Classify the map coverage errors (required)**

    Classify each of the distinct polygons in the geographic area calculated in the previous use case with a possible error type (e.g., ghost precinct, gap between two precincts, etc.). Store the associated data in the errors data structure for subsequent addition to the DB.

19. **Identify anomalous precinct data (required)**

    At a minimum, precincts should be considered anomalous if the underlying data is not consistent. For example, population or voting data could be zero or voting data might not be a reasonable percentage of population. This is considered a distinct error category, so you should store the associated data in the errors data structure for subsequent addition to the DB.

20. **Clean raw data (preferred)**

    Resolve inconsistencies in the data (e.g., county designation). Use standardized data formats for each type of data.

21. **Identify precinct neighbors (required)**

    Identify two precincts as neighbors if they share a common boundary of at least 200 feet.

Data Corrections (8 required)

22. **Select item to be corrected (required)**

    The user can select an item to be corrected in the GUI. The item should be available in a hierarchal list of corrections or by selecting the item in the map.

23. **Highlight item to be corrected (required)**

    Once an item to be corrected is selected, the item and data associated with the item should be highlighted in the GUI. The GUI should modify its view (e.g., pan and zoom) so that the item is clearly visible in the GUI.

24. **Select merge precincts (required)**

    The user can select two or more precincts to be merged. The GUI will highlight the selected precincts in the map and display the associated precinct data. Note that merged precincts will primarily address the issue of one or more precincts being geographically enclosed by another precinct. However some merge actions can result from a multi-polygon precinct (e.g., islands), and would be handled through labeling a ghost precinct and combining the ghost precinct with each of the polygons in the multi-polygon.

25. **Merge precincts (required)**

    The precincts should be merged and the new precinct boundary changed, a name generated for the new precinct, and the data associated with the precincts combined. At the conclusion of the merge, the map should show the new precinct and the data associated with the new precinct.

26. **Correct precinct boundary (required)**

    Modify the boundary of a precinct. Typically, the user will do this to close a gap between precincts or correct intersecting precinct boundaries. This should be done interactively in the map interface

27. **Snap together two edges from two different precincts (preferred)**

    The user should be able to select two edges, each from a different precinct, and allow the user to specify that the edges be coincident. It may take a sequence of such actions to correct a precinct alignment error.

28. **Define ghost precinct (required)**

    Allow the user to verify that a possible ghost precinct (identified in preprocessing) is a ghost precinct. If so, allow the user to initialize the data for the ghost precinct (e.g., set population to zero)

29. **Group precinct boundary updates into a transaction (preferred)**

    Allow the user to correct a boundary error with a transaction. Each boundary update is a step in the transaction. The user can close to confirm or abort the transaction at any stage of the process.

30. **Verify precinct boundary (preferred)**

    The boundary of the new precinct should be verified to be a single closed polygon with no intersecting edges.

31. **Add/delete precinct neighbors (required)**

    Allow the user to specify that two precinct are neighbors or to delete a precinct neighbor association.

32. **Modify precinct data (preferred)**

    Allow the user to modify any alphanumeric precinct data (e.g., voting results).

33. **Log the relevant data associated with any data correction (required)**

    For any data correction, information about the correction should be logged. The data will vary with the type of correction. For example, if the user changes the shape of a precinct boundary polygon, the old polygon and new polygon should be logged. In addition, you would log the date and time of the logging, along with relevant identifiers (e.g., state and precinct).

34. **View corrections log (preferred)**

    User should have the ability to view the entire corrections log, view the corrections by error category, and the errors associated with a specified precinct.

35. **Log an optional user comment associated with a data correction (preferred)**

    Provide the user with the option of associating a comment with a change. The comment should be associated with the other logging data for this correction.

36. **Allow users to add/remove/update comment associated with a data correction (preferred)**

    The users can view a correction comment in the GUI sections and add, remove, or update the comment.

37. **Modify error list (preferred)**

    Allow users to select an error and see more details about the flagged data (data source, type of error), mark a potential error as correct data. Update database to reflect this change.

General (0 required)

38. Asynchronous client update (optional) (one sequence diagram should show the asynchronous logic)

Implement asynchronous update of the GUI (e.g., using Web Sockets instead of http). At a minimum, the data to be transmitted asynchronously should be precinct boundary data.

39. The GUI will be organized so that controls are visible only when appropriate (preferred) (sequence diagram not required)

If you choose a tabbed display, each tab should be contain logically related components, so that all tab components (including scrolled components) are appropriate to a given stage of processing. You might also satisfy this requirement with hamburger menus.

40. Database normalization (preferred)

Ensure that your DB is in 3rd normal form.

41. Implement client/server traffic analysis tools (preferred)

Use performance monitoring to ensure that client/server data traffic does not cause GUI performance problems.

Additions