Casey:

Editing the backend to take user-input values

1. Upload a CSV
2. The data column input is filled with the column names from the uploaded CSV
3. Select one of the data columns
4. Check what data is loaded into the cartogram by its tooltips on the different areas, and see if it is what is in the uploaded CSV

Backend to analyze user-input data sheet to allow for data to be presented to user in more readable format

1. Upload a CSV
2. Check data column
3. Make sure it has all columns from the CSV file

Implement Google's Material Design Guidelines

1. First make sure understand the premise of Material Design (look at examples, the handbook, Google products)
2. Reimplement clunky elements so they have a simpler design

Switch from transform map to dynamic map

1. Test map and ensure it works like previous setup

Making site compatible with ARIA

1. Run Google Accessibility audit tool (extension from Chrome store)
2. See what they report needs fixing, and look up how to better implement it.
3. Use screen reading emulator software

Jiayao:
File Upload:
Try curl calling the end point with a POST request attaching a file
Verify that the file exist
Build an index page to manually verify the functionality of the php script

Cookie:
Setting cookie behavior in chrome console
Console log and break points
Run the script in multiple environment and collected the cookie information and output in the backend
Other functionality:
Launch heroku and verify it in the production environment

Luke:

- Early-stage user file upload (local DOM storage)
    - open the web page, click the upload button and select the appropriate file
    - launched the app locally and verify that the data selection drop-down displays the data corresponding to the previously selected file.
- String-to-file upload to and file-to-string-download from the server

    - Send a POST request using Ajax with the string information attached
    - Await a response on the other side containing the string that was originally sent, indicating that the file write and read has taken place.
- Save a file on the server that is associated with the current user's session ID
    - Select a CSV using the Upload button
    - Click "Save" button in the Save/Load tab
    - Verify the existence and correctness of the new file on the server
- Association of previously uploaded server-side files with new users:
    - Save a file to the server
    - Verify that the previously-uploaded file exists on the server
    - Load that file using the load form in the Save/Load tab
    - Check if a new file associated with the current user exists on the server
    - Verify that its contents match the contents of the previously uploaded file
- Load the user's session ID into a front-end form for copying
    - Run the program
    - Confirm a match between the ID displayed and the value stored in the browser cookie

Jeff:

Topojson map rendering:

- Testing involved quite a bit of Chrome browser inspection, along with strategic console logs in the JavaScript code.

Extra map generation:

- Testing was not exactly systematic here, as every topojson map follows different conventions. The first test step is simple manual inspection while running the map through a custom python program, to ensure that json files meet our program's various requirements for data, most importantly that the map displays the right regions, and that the regions are identified properly.
- A test stub with a stripped-down version of our map display takes in custom latitude, longitude, and scales to test the ability of topojson maps and .csv's to simply render. I would first test .csv's and topojson maps here before adding them to the main branch.

Map menu:

- In the main branch, testing the map menu required clicking and transitioning between many different combinations of maps, ensuring the buttons worked.This involved making changes locally, and ensuring they worked by running our index.html on Chrome browser through my WebStorm IDE, using its dev tool features every step of the way.

Michael:

Tabs:

-hover over icons, ensure opacity changes

-hover over icons, ensure tooltip appears/goes away after

-click icon

- modal opens for Share and Save/Load
-File browser opens for Upload
- change size of browser to make sure tab sizes change

Modals:
-click ‘Close’ to close
-click outside modal to close
-click ‘esc’ to close

Twitter:
-click Twitter button
-ensure tweet opens up
-ensure correct Session ID is included

Email:
-click email button
-ensure email opens up
-ensure correct Session ID is included

Svg:
-click SVG button
-make sure myCartogram.svg is downloaded
-open to make sure svg info is there (note: xml styling is not displayed)

Png:
-click PNG button
-ensure cartogram.png is downloaded
-open to check cartogram image matches actual cartogram

Dialog:
-click on Save button
-ensure dialog opens, then disappears shortly after

Ahmed:
Manually test by heroku and comparing result with other teammate