Purpose:

The purpose of this React application is to expand on the Project 1 coordinated multiple view user interface by incorporating concepts from Homework 5 and Homework 6. As with Project 1, this application supports the same file menu functionalities in terms of local storage. However, it now also incorporates a MongoDB database that stores default data (“World Population”, “Empty”, and “Grades Distribution”) and allows users to add new datasets into the MongoDB database as well. Additionally, a File menu with cut, copy, and pasting functionality, as well as a linking and brushing functionality were added.

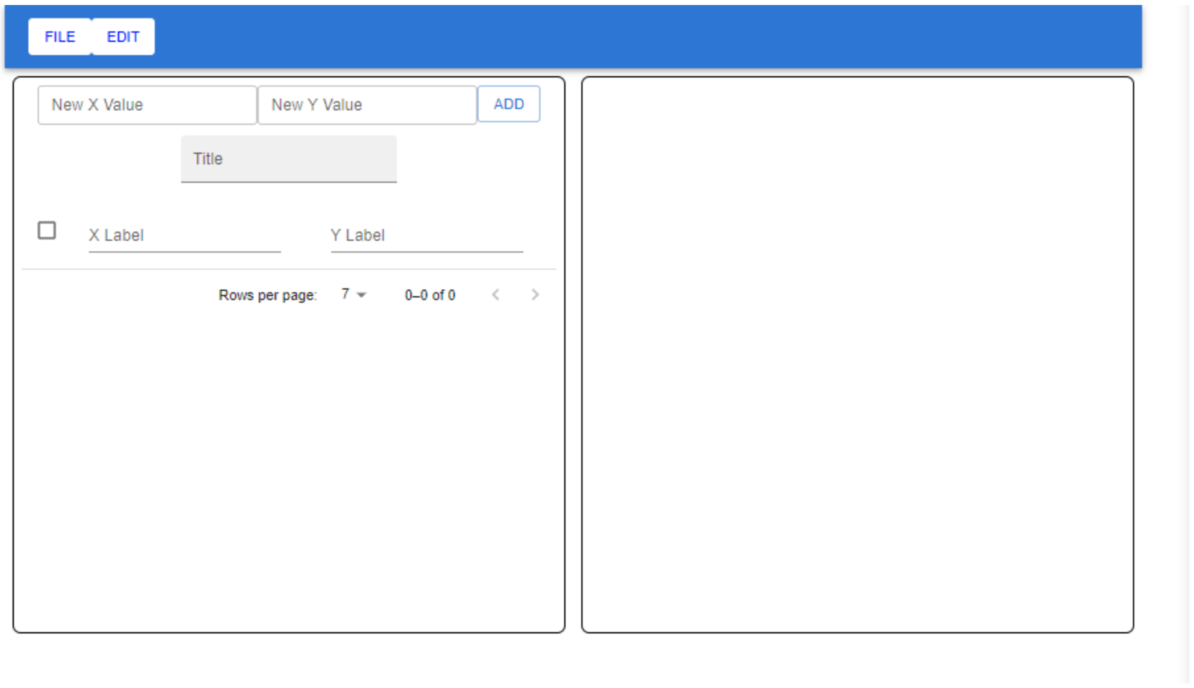


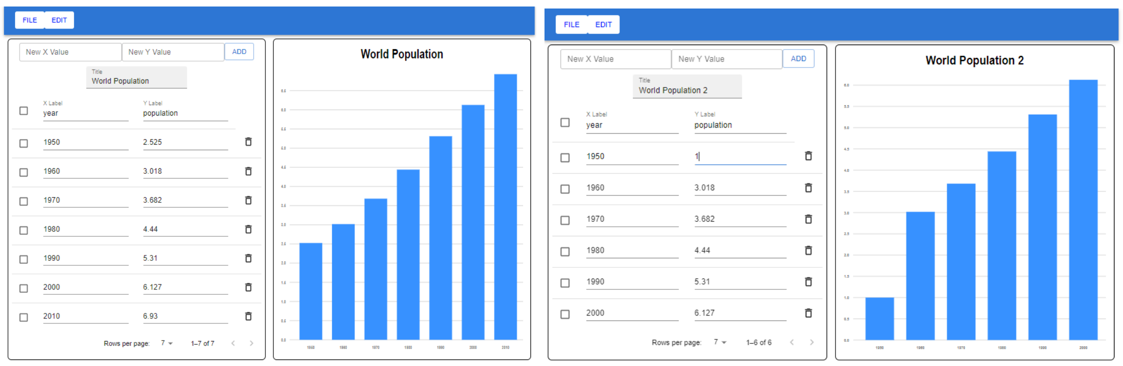
Figure 1: The start-up display, including a editor component on the left, a bar chart component on the right, and a menu bar at the top with the title of the project and a “FIle” menu that consists of the Web Storage API functionality, and an “Edit” menu with Cut, Copy, and Paste functionality.

Description of Usage:

This section will detail the different functionalities of the application with images.

Modifying the dataset

Users can modify the dataset in the editor by selecting the TextField that they wish to edit and changing the values accordingly. Changing the y-axis to a non-numerical value will cause the bar to automatically assume the value 0. Users may add a new row by filling out the “New X Value” and “New Y Value” fields at the top of the editor, and then pressing the “Add” button. Additionally, data points can also be deleted by selecting the corresponding trash can icon next to the row users wish to delete. Performing these actions on the dataset will reflect in the bar chart side of the display in real time, as displayed in Figure 2



Title: World Population

X Label: year	Y Label: population
1950	2.525
1960	3.018
1970	3.682
1980	4.44
1990	5.31
2000	6.127
2010	6.93

Rows per page: 7 1–7 of 7

Title: World Population 2

X Label: year	Y Label: population
1950	1
1960	3.018
1970	3.682
1980	4.44
1990	5.31
2000	6.127

Rows per page: 7 1–6 of 6

Figure 2: In this image, the title has been changed from “World Population” to “World Population 2”, the y-value of the first data point has been changed from 2.525 to 1, and the last data point (2010, 6.93) was deleted and is not longer displayed on the graph. As you can see, the bar chart has been updated accordingly.

File Menu

This application’s File menu functionalities correspond to the MongoDB database and the browser’s local storage. Users can create, save, and load data sets into the browser’s local storage called localStorage, and the modifications of any datasets will be reflected in the MongoDB database as well.

Users can create a new data set by navigating to the File dropdown menu and selecting “New”. This will prompt a Dialog box to open and request a name for the new data set. Once this information has been inputted, a new dataset appears on screen, which consists of default title value, an empty data point, and a blank graph. Users can manipulate their dataset as they wish.

If they wish to save this dataset to retrieve later, they can select File->Save As. This will prompt a Dialog box asking for the file name. Once the user enters a dataset name, the dataset is uploaded into both localStorage and the MongoDB database as a JSON object, with the key being the fileName and the value being another JSON object called fileContent, consisting of a title String and data array. Whether the dataset is a new dataset or a modified version of an existing dataset, performing the “Save As” action creates and uploads a new dataset to localStorage and MongoDB containing the current data displayed on the screen. If the user later wishes to update their dataset, then, rather than saving it again as an entirely new document, they can overwrite the existing data in the data set by selecting File->Save. However, clicking “Save” for a new document will post a new dataset to localStorage and MongoDB with the title being the inputted filename and the data being the default values {x:””, y:0}

Any dataset that is stored in localStorage via the Save As feature can later be retrieved using File->Load. This will trigger a function that refreshes the localStorage contents by syncing the most recent MongoDB database retrieve any dataset that is stored in the browser’s localStorage. When users select File->Load, a Dialog box containing a list of the datasets stored in localStorage are displayed, and selecting a dataset from the list will update the editor and bar chart to reflect the values of this dataset.

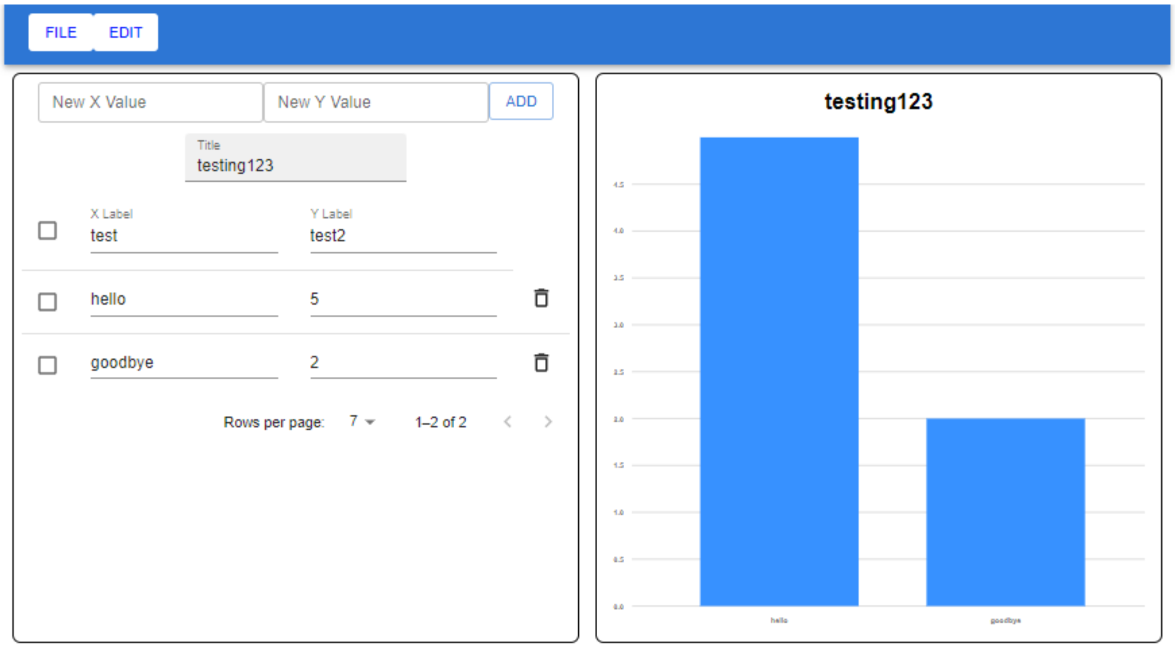


Figure 3.1: New dataset (produced by File->New) that has since been modified but not yet saved to the database.

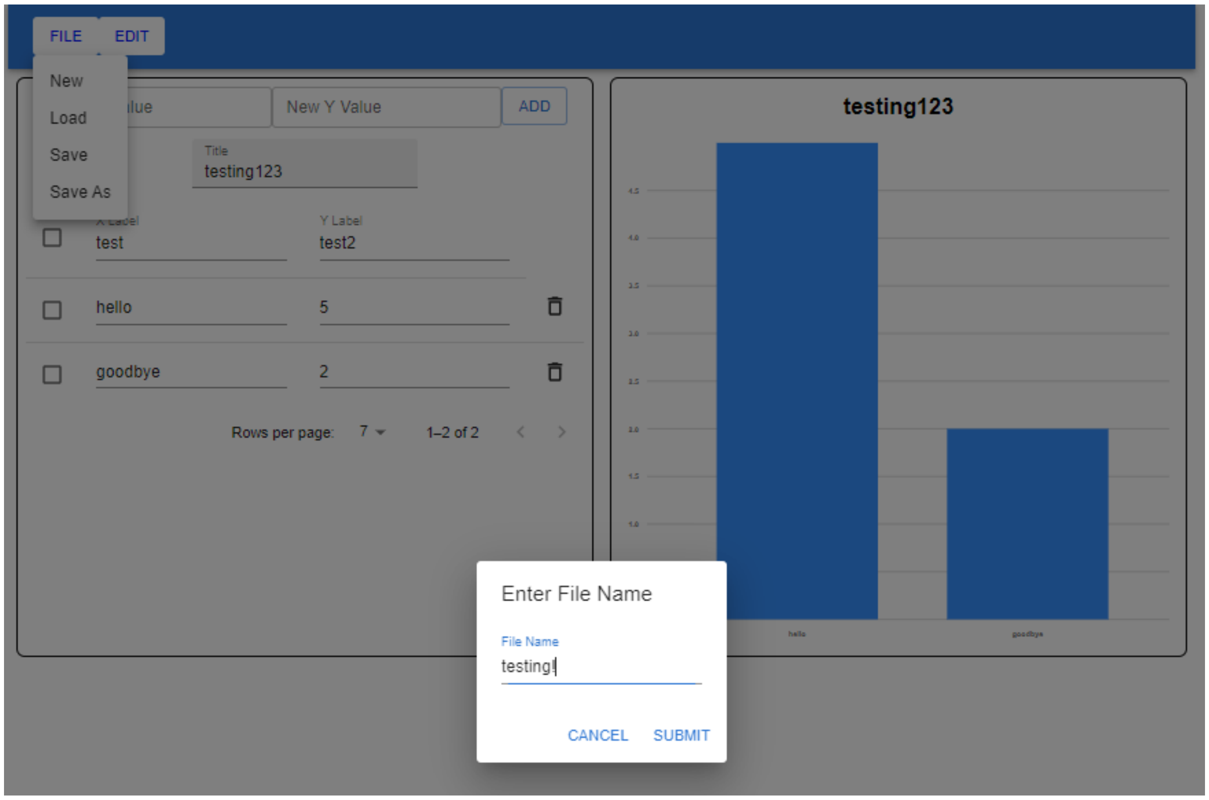
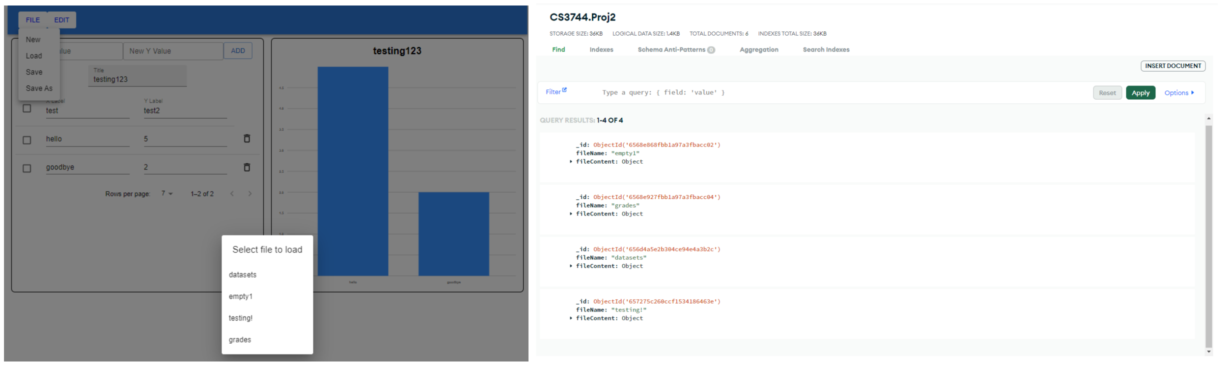


Figure 3.2: The Dialog box that appears when File->Save As is selected. I have manually inputted “testing!” as the desired file name.



Title: testing123

	X Label	Y Label
	test	test2
	hello	5
	goodbye	2

Rows per page: 7 1–2 of 2

CS3744.Proj2

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 1.4KB TOTAL DOCUMENTS: 6 INDEXES TOTAL SIZE: 36KB

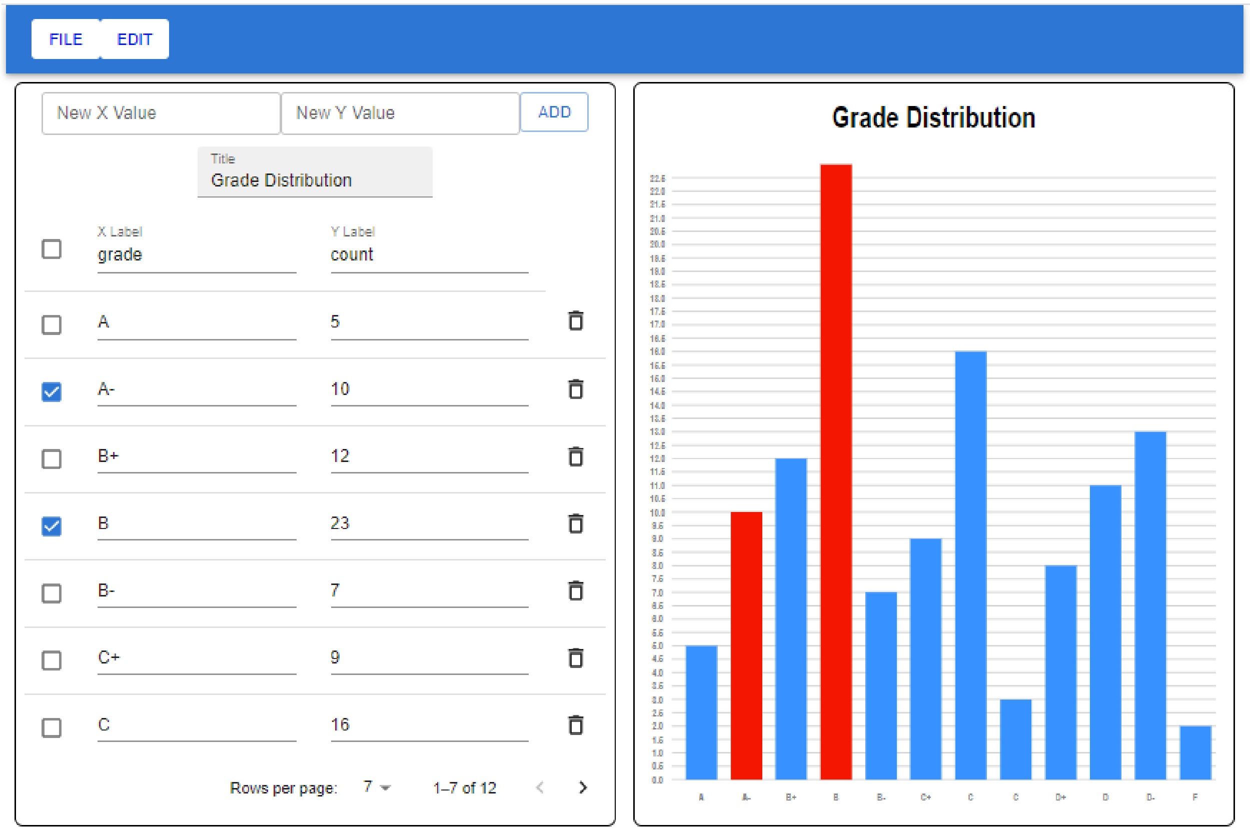
```
Type a query: { field: 'value' }
```

QUERY RESULTS: 1-4 OF 4

```
  _id: ObjectId('6568e868fbb1a97a3fbacc02')
  fileName: "empty1"
▸ fileContent: Object

  _id: ObjectId('6568e927fbb1a97a3fbacc04')
  fileName: "grades"
▸ fileContent: Object

  _id: ObjectId('656d4a5e2b304ce94e4a3b2c')
  fileName: "datasets"
▸ fileContent: Object

  _id: ObjectId('657275c260ccf1534186463e')
  fileName: "testing!"
▸ fileContent: Object
```

Figure 3.3: The Dialog box that appears when File->Load is selected. “testing!” now appears as one of the stored datasets in both localStorage and in MongoDB

Linking and Brushing



Title: Grade Distribution

	X Label	Y Label
☐	grade	count
☐	A	5
✓	A-	10
☐	B+	12
✓	B	23
☐	B-	7
☐	C+	9
☐	C	16

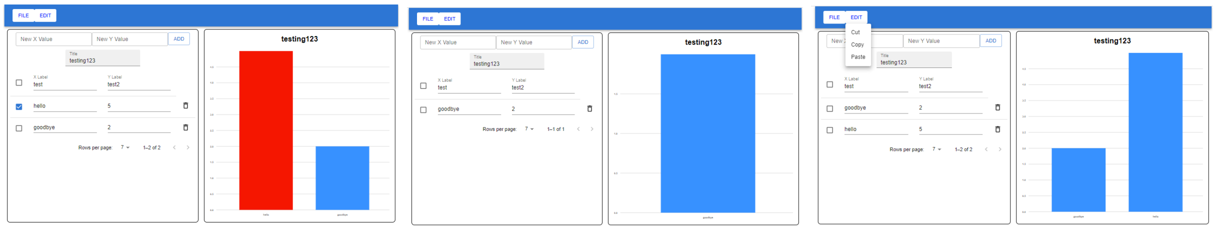
Rows per page: 7 1–7 of 12

Figure 4: selected rows are highlighted red in the bar graph

Linking and brushing was implemented in this project as well. When a row in the editor is selected using the check box, its corresponding bar in the bar chart view is highlighted red. Additionally, when a bar in the bar graph is clicked on, then the corresponding row’s check box is toggled.

Edit Menu

A cut, copy, and paste row feature was implemented in this project. When selected rows are cut, they are stored locally until another cut or copy call is made, in which the stored data is overwritten by the new cut/copy data. Paste appends any cut/copied data stored locally onto the current dataset. See images below.

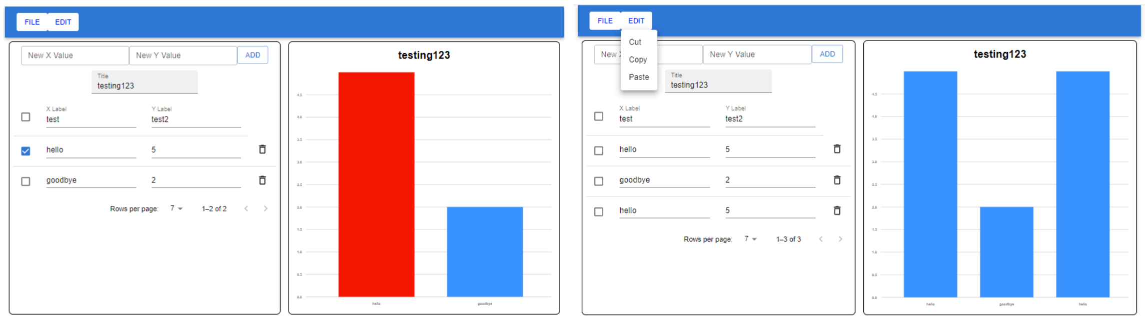


Title testing123
X Label test   Y Label test2
hello 5
goodbye 2
Rows per page: 7   1–2 of 2

Title testing123
X Label test   Y Label test2
goodbye 2
Rows per page: 7   1–1 of 1

Title testing123
X Label test   Y Label test2
goodbye 2
hello 5
Rows per page: 7   1–2 of 2

Figure 5.1: Row “hello” is cut, and the data is removed from the dataset. Then, when pasted, it is appended onto the end of the dataset



Title testing123
X Label test   Y Label test2
hello 5
goodbye 2
Rows per page: 7   1–2 of 2

Title testing123
X Label test   Y Label test2
hello 5
goodbye 2
hello 5
Rows per page: 7   1–3 of 3

Figure 5.2: Row “hello” is copied and, when pasted, is appended onto the end of the dataset

Limitations

Due to the time constraints of this project, there were a few bugs that I was unable to resolve. Below is a list of these bugs

- When linking and brushing multi-page datasets, when selecting a row on one page and then navigating to another page, the indices of the selected rows in the previous page will also be highlighted in the current page. This is due to the handling of selected rows, as it is stored by index rather than item.
- Upon refreshing the page to the start up page, the application crashes when trying to input text into the x and y title fields.