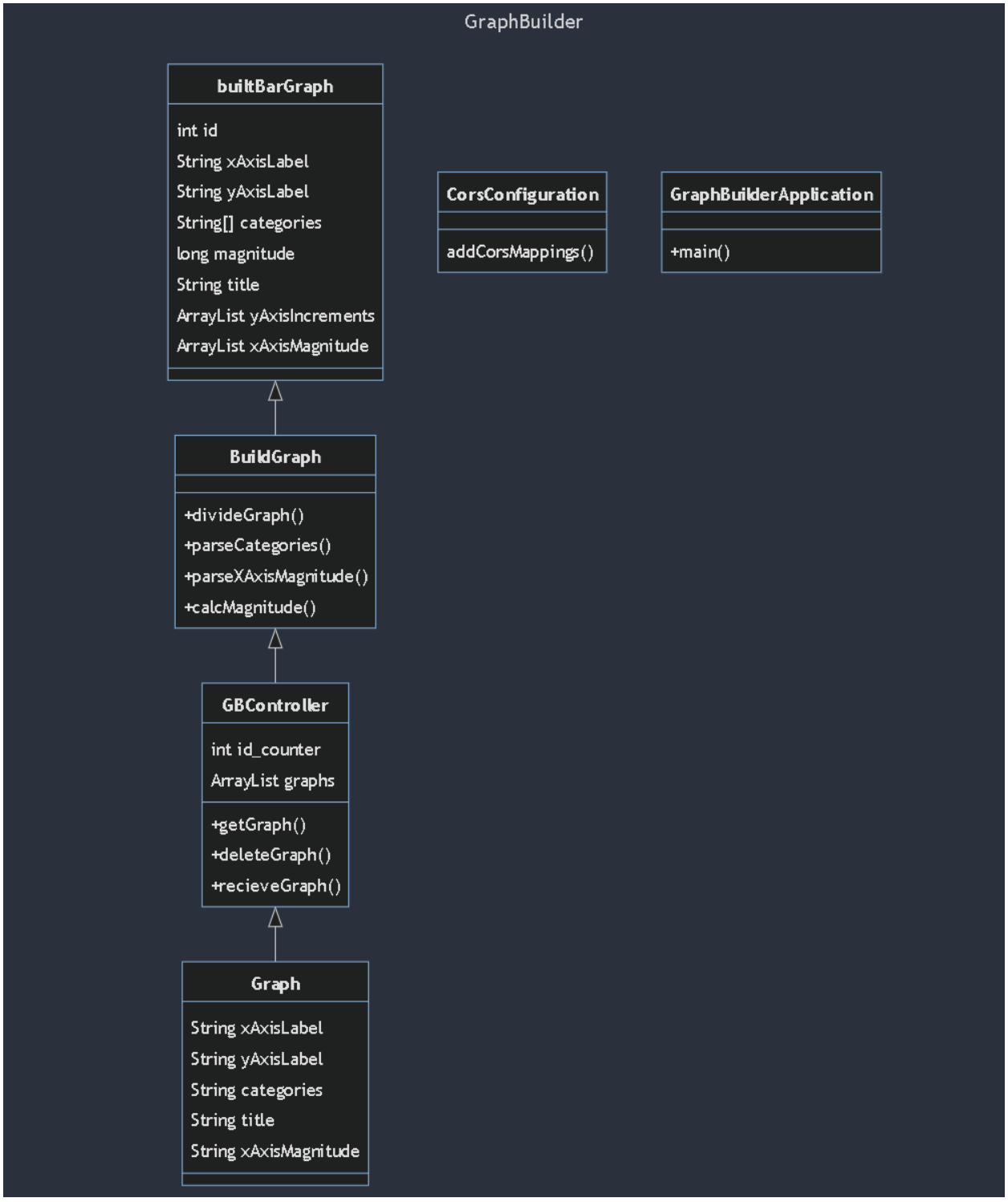
Graph Builder Implementation Manual

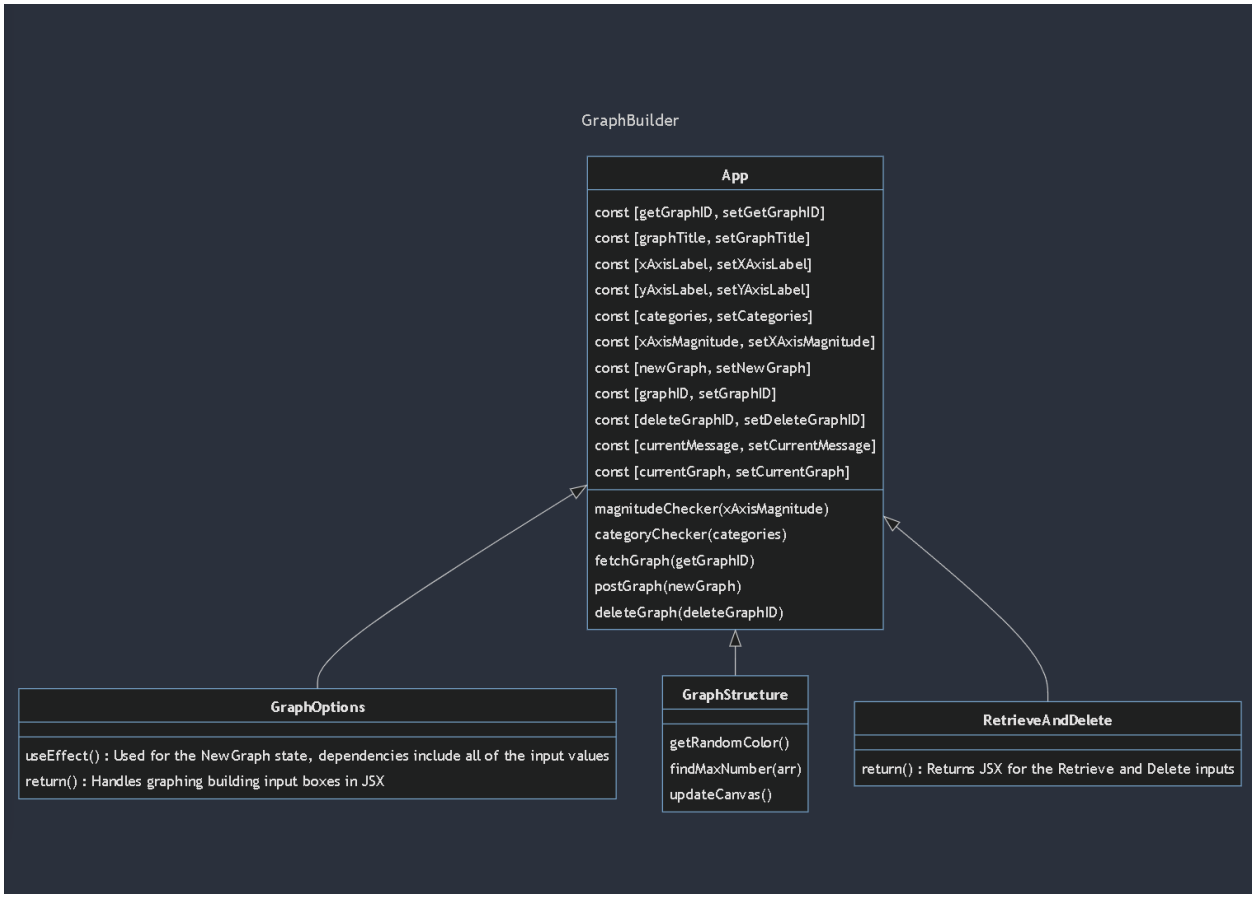
Motivation: Graphs, specifically bar graphs, are useful for visualizing the difference between a number of separate objects and their quantities. The process of building these apparatuses is repetitive and tedious and, therefore, can be automated, which is what this project aims to accomplish.

Strategy: I cut a thin slice through the tech stack to create something that is functional but limited in the number of functionalities. I then aimed to refine these few functionalities as effectively as possible.

Lessons Learned: The user does not care about your tech stack, only the user experience. The tech stack is for the developer's convenience.



Note: The arrows in the diagram above represent a process as opposed to inheritance



Note: The arrows in this diagram represent child components

POST

- The Graph is first constructed with input boxes in the JSX of the GraphOptions component.
    - There is an onChange attribute that sets the state (using a useState hook) of the specific property of the graph that the text box has. While this occurs, there is a useEffect hook that has dependencies of all of these different properties. When the useEffect is triggered, it sets the state of a “newGraph” object.
    - When “Create Graph” is pressed, the postGraph function is called and it sends a POST api request via REST api using Axios to the Spring boot backend. Also in this function, it is ran through a series of checkers to make sure that the input is valid.
- The GB controller then receives this graph and creates a graph object out of it in Java using a record. This is done through the recieveGraph method.
    - Also in the receiveGraph method, the graph object is modified into a builtBarGraph object.
        - This is done using the BuildGraph class, which contains various methods for preparing the graph for display, such as taking the input string from “Categories” and making into an array. This built graph is then entered into the ArrayList of builtBarGraphs which serves as a primitive database.

GET

- The user enters a number input in the input box labeled “Retrieve Graph”
    - Similar to the start of the POST method, this input also has an onChange that is linked to a useState of getGraphID
    - When the submit button is pressed, it calls an anonymous function that calls fetchGraph. This function uses Axios to send a GET request to the backend
- The getGraph function in the GB controller receives the number inputted by the user and searches the array for the ID that matches the number. Once the ID is found, it is returned to the frontend and then destructured. If the ID is not found, then a message is returned that says the graph does not exist.
    - If the ID is found, it takes the destructured object and sets the currentGraph state to it
- The setting of the currentGraph state triggers a re-render of the render function in the GraphStructure component.
    - The render function destructures the current state of currentGraph and uses its attributes to build the bar graph. Some notable components of this process:
        - Creates two perpendicular lines
            - Creates the X-axis via x-axis fragments that extend the graph when more categories are added
        - Places “small” labels that are equally spaced apart along axises
        - Places “big” labels that are centered with the lines and a title that is centered with the bottom line
        - Creates randomly colored bars that represent the magnitude of the data. These bars’ heights is based on their data’s magnitude in comparison with the max value given by the highest “small” label
            - Spaces out small labels and bars based on the character length of the category with the longest character length.

DELETE

- The DELETE method is somewhat identical at the beginning to the GET method
    - The difference begins when the GBcontroller receives the number inputted by the user. It is placed into the deleteGraph method that searches the ArrayList of builtBarGraphs for the graph with the matching ID
        - Once the graph is found, it is then removed from the array
        - If the graph does not get found, it returns a message stating that the graph was not found