**Team No. 4**

**Wyatt Bushman, Thomas Dalbavie, Michael Lindemann, Chris Nunez**

**Cutting Stock Calculator CSC**

**Built by Peter LLC.**

**Milestone 4**

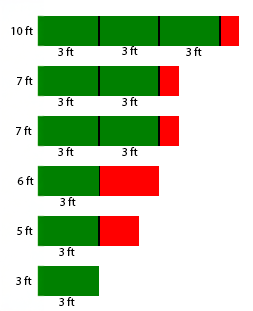
## **1. Introduction:**

There is no free online existing application that elegantly solves the “cutting stock problem” for DIY hobbyists. Our proposed solution is to build one from the ground up using HTML/CSS/NodeJS. The application will implement various algorithms for that are associated with “bin-packing” solutions such as shelf-pack and guillotine [1,2].

## **2. Proposed Solutions to Functional Requirements:**

Functional Requirement 1: 1-Dimensional Problem

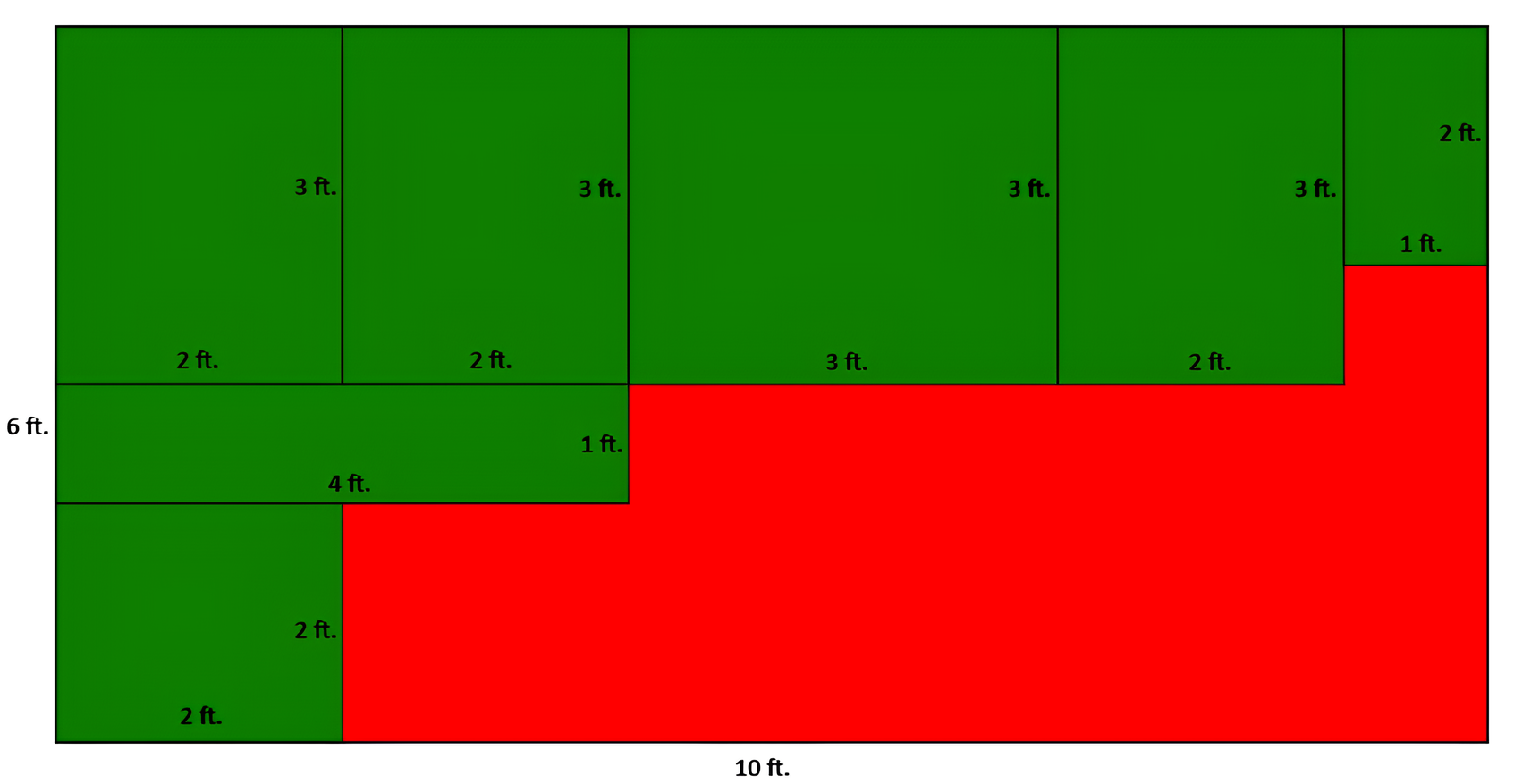
* The user will either: manually input or load a JSON file of desired: lengths of wood, quantity of each, blade thickness, and as an added use-case, optionally: manually input or load a JSON of currently available: lengths of wood, quantity of each, and blade length.
  + Proposed Solution 1: Iteratively cut until exhausted all wood, produce graphical output for user to cut. Example, enter lengths of wood and set a length of 3ft. Will continue to cut until no more 3ft lengths can be cut.
  + Proposed Solution 2: Iteratively cut given a hash table. Example, user wishes to have 15, 3ft pieces, 10, 2ft pieces, and 5 1ft pieces.
  + Preferred Solution: Both will be implemented for more flexibility for the user’s input.



*Figure 1. 1-Dimensional Problem Example Output*

Functional Requirement 2: 2-Dimensional Problem

* The user will either: manually input, or load a JSON file of desired: canvas sizes, quantity of each, blade length, and as an added use-case, can optionally: manually input, or load a JSON of currently available: canvas sizes, quantity of each, and blade length.
  + Proposed Solution 1: Generate a solution using the Shelf algorithm [1].
  + Proposed Solution 2: Generate a solution using the Guillotine algorithm [2].
  + Preferred Solution: Running the Shelf and Guillotine algorithm in parallel. Select the best output by comparing each of the algorithm’s final area of waste. Then returning the result with the lowest value for total area of waste.



*Figure 2. 2-Dimensional Problem Example Output*

Functional Requirement 3: Input Definitions

* The user will have an intuitive but extensive input field to make it simple for basic users, but powerful for professionals. The input should also be easy for bulk entry when working with larger scale cutting problems.
  + Proposed Solution 1: Users will get preset wood cut sizes, canvas sizes, and saw blade thicknesses to make it simple and easy to use for entry level users.
  + Proposed Solution 2: Users can manually input custom sizes for wood cuts, canvases, and saw blade thickness to allow for less typical cuts that cater to specific needs. This can be useful for more professional industrial grade equipment which may not follow the same standards as consumer products.
  + Proposed Solution 3: Allow users to input JSON files to either pull up previously generated blueprints (more on that later), but more importantly, allow for easy bulk input to generate larger scale blueprints.
  + Preferred Solution: A combination of all 3 solutions will be created. Each of these proposed solutions solve a problem for each user class (i.e. DIY/Hobbyist, Professionals, Manufacturers)

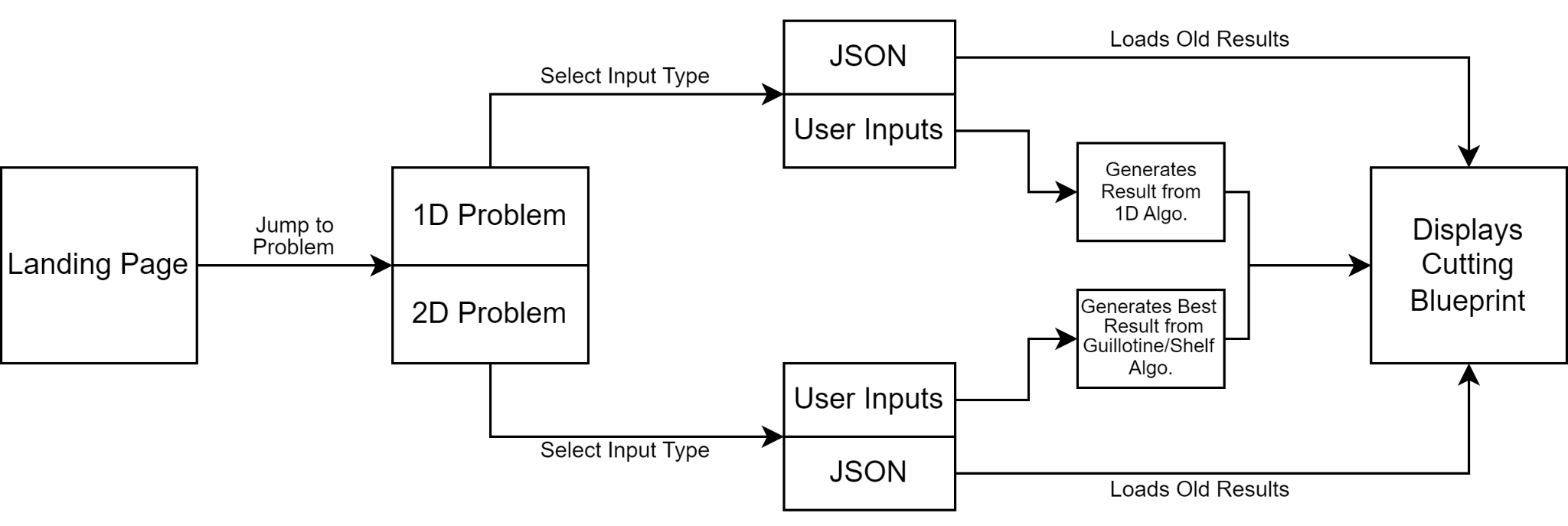
Functional Requirement 4: Output and Visualization

* The user expects to get a visual representation of the lines that need to be followed to cut. The lines should vary in thickness based on the provided saw blade thickness. This will allow users to follow the cut lines without the worry of running out of canvas.
  + Proposed Solution: Provide appropriate labeling for each side of both wood cuts and canvas size to make it clear to the user how to mark off the canvas’ when preparing to cut. This will also indicate what parts of the canvas is considered waste, so the user doesn’t mix up actual cuts they want with remaining waste.

Functional Requirement 5: Waste Minimization

* The user expects to receive the lowest waste option when generating a cutting blueprint to follow along to.
  + Proposed Solution: For the 2D problem, different algorithms can be better in different cases. This is why we will run multiple different algorithms (i.e. guillotine or shelf) in parallel and select the best result given the output.

## **3. System/Data Flow Diagram:**

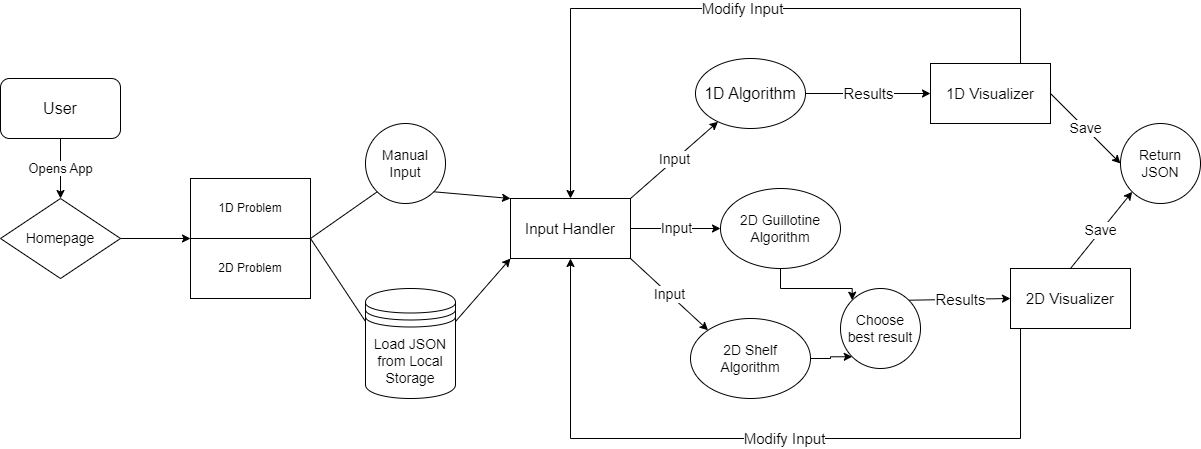


*Figure 3. System Flow Diagram*

The system flow Diagram shown above shows the general workflow of the website. The website follows a simplistic one-page setup where the user will start at the top of containing all the general page information. There will be tabs/nav bar at the top where the user can jump down the page to the problem of their choice (i.e. 1D or 2D). While each problem works similarly, they vary largely on the backend.

For the simpler 1D problem, once the user is on that section of the page, they are either asked for an input or a JSON file. The user will give a list of plank lengths they want to cut, one or more canvas lengths, and saw blade thickness for the algorithm to work from. If the user has a previously generated output saved, they can provide a JSON file and display it that way without need for calculations. Once generated, the user will get a visual blueprint that will show how each cut is organized with the given canvas sizes for a lowest waste combination. This blueprint can then be downloaded as a JSON for the user to save and regenerate through the website later.

For the more complex 2D problem, a lot of the process of generating the blueprint for this is the same to the user. The main difference is that the user will have to input two dimensions per wood block and canvas that the user wants. After generating the blueprint, the system will run the guillotine and Shelf algorithm in parallel and select the option with the least waste to display to the user. Like the 1D problem, this can then be saved as a JSON file and later regenerated again as an input.



*Figure 4. Data Flow Diagram*

The Data Flow Diagram (DFD) in Figure 3 illustrates how our application handles data. The user begins at the homepage and selects between the 1D and 2D versions of our application. From there, they may choose to manually enter their data, such as wood length, canvas size, saw thickness, etc., or upload the information via JSON from local storage. The input data is then processed by the input handler and passed to the appropriate algorithms.

If the user chooses the 1D algorithm, the numerical results are returned and visual results are displayed by the 1D visualizer. For the 2D algorithm, the data is passed to both the Guillotine and Shelf algorithms, which are compared to find the best solution. Like the 1D algorithm, the results are passed to the 2D visualizer, returning both a numeric and visual output to the user. In either case, the user has the option to modify their entries or save their results for future use.

## **4. Proposed GUI Sketches:**

Below is our proposed GUI mock-up. The goal of the application is simplicity. The user can select either a 1D or 2D solution for their wood cutting needs. 1D deals with lengths of wood of varying lengths and simply cutting it to a desired length. Users can manually enter the lengths they have available, their desired length, and the amount they would like cut. There is also an option for users to submit a JSON file which is in the format of:

[

{DesiredLength: 5,Quantity: 1},

{DesiredLength: 2,Quantity: 4},

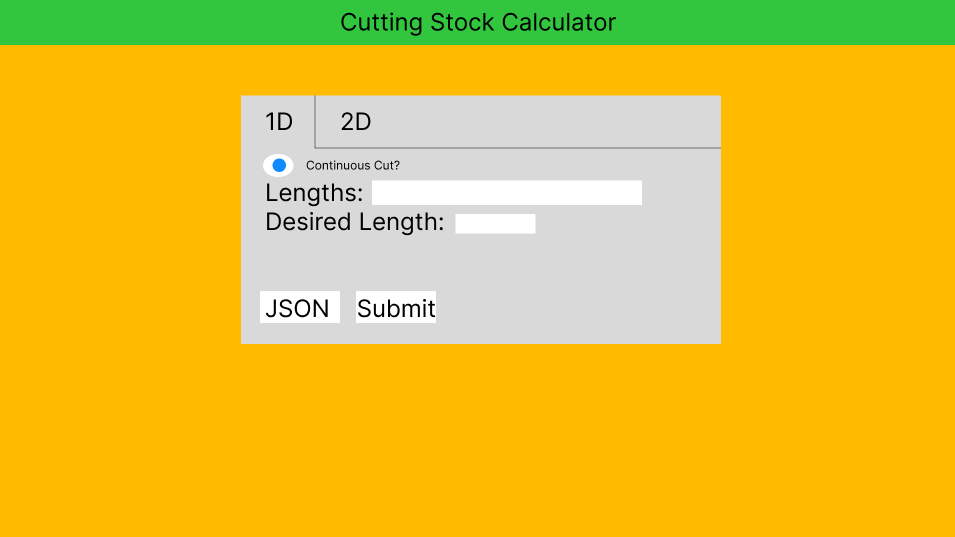
{DesiredLength: 5,Quantity: 1}

]

Users can also select to continuously cut if they aren’t sure about a desired quantity.



*Figure 5. Default 1D problem set up*



*Figure 6. Continuous Cut option selected*

Similarly, upon selecting the 2D option a user can input varying canvas sizes and enter in desired pieces they wish to have cut. A JSON file can again be used to input desired dimensions.

Example:

[

{"width": 2, "height": 3, "quantity": 2},

{"width": 3, "height": 3, "quantity": 1},

{"width": 4, "height": 1, "quantity": 1},

{"width": 2, "height": 3, "quantity": 1},

{"width": 2, "height": 2, "quantity": 1},

{"width": 1, "height": 2, "quantity": 1},

{"width": 2, "height": 3, "quantity": 2},

{"width": 3, "height": 3, "quantity": 1},

{"width": 4, "height": 1, "quantity": 6},

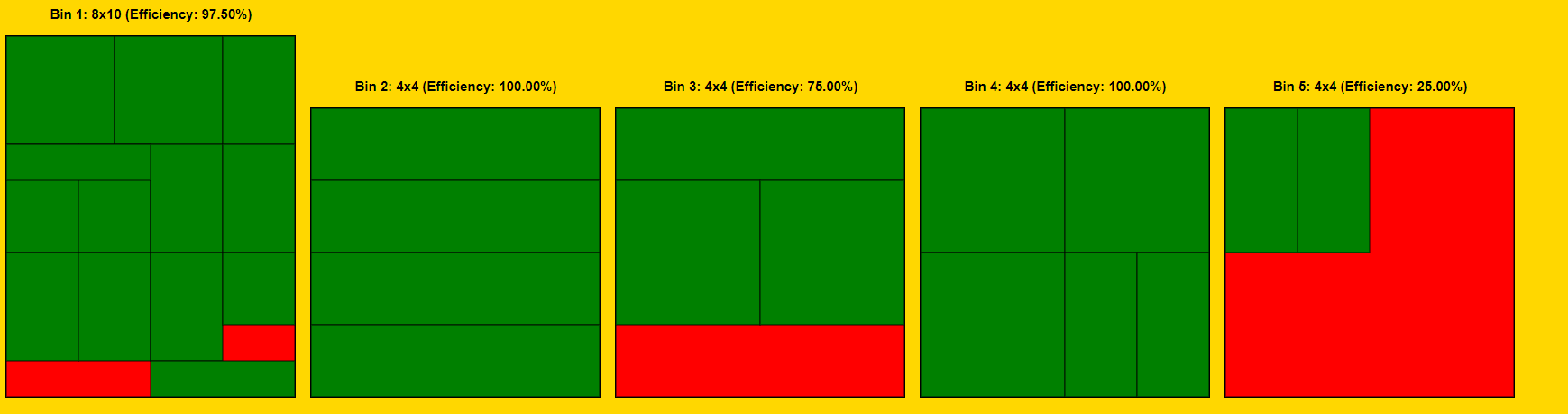
{"width": 2, "height": 3, "quantity": 3},

{"width": 2, "height": 2, "quantity": 5},

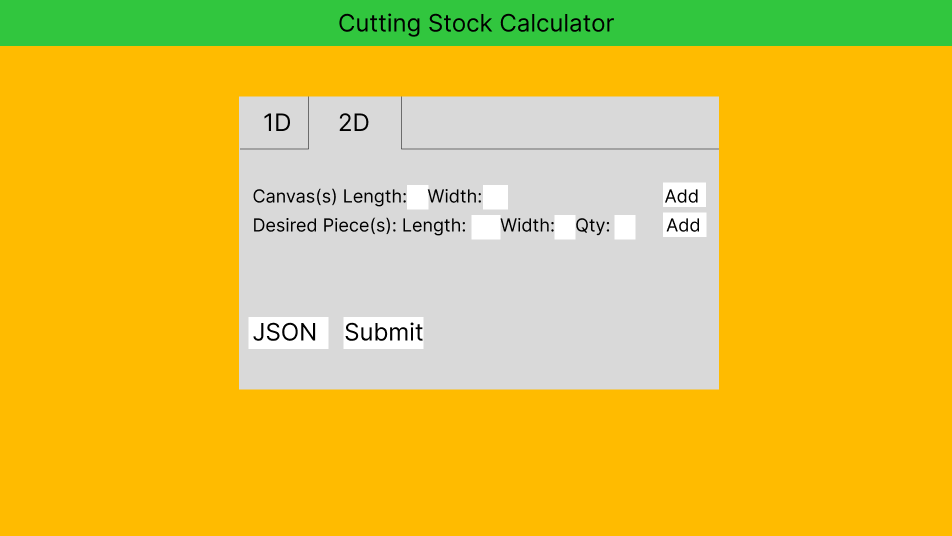
{"width": 1, "height": 2, "quantity": 3}

]

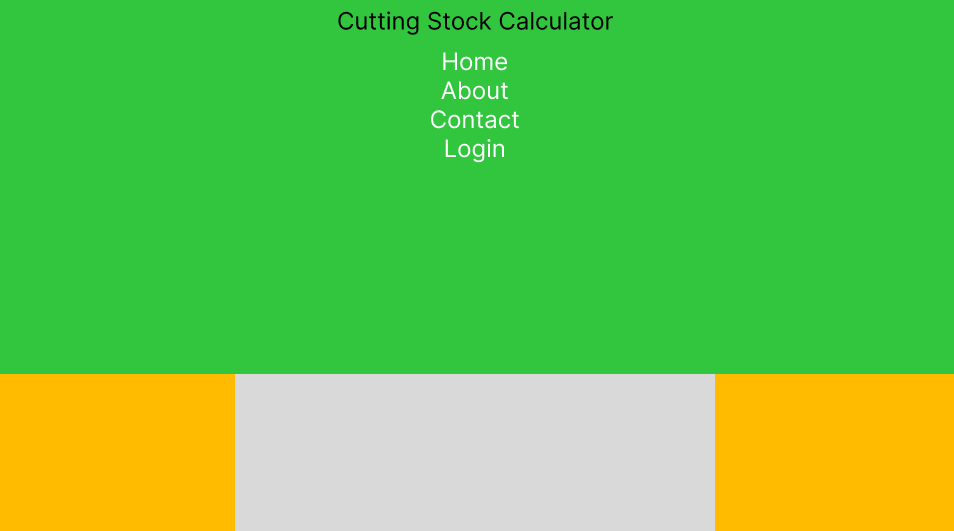
The program will output an answer using the canvas dimensions as a frame of reference. It will use the biggest canvas available and then go down in size to save on wood. Below is an example using the about JSON file specifying a 8x10ft canvas and 4x4ft canvas as the only options available.



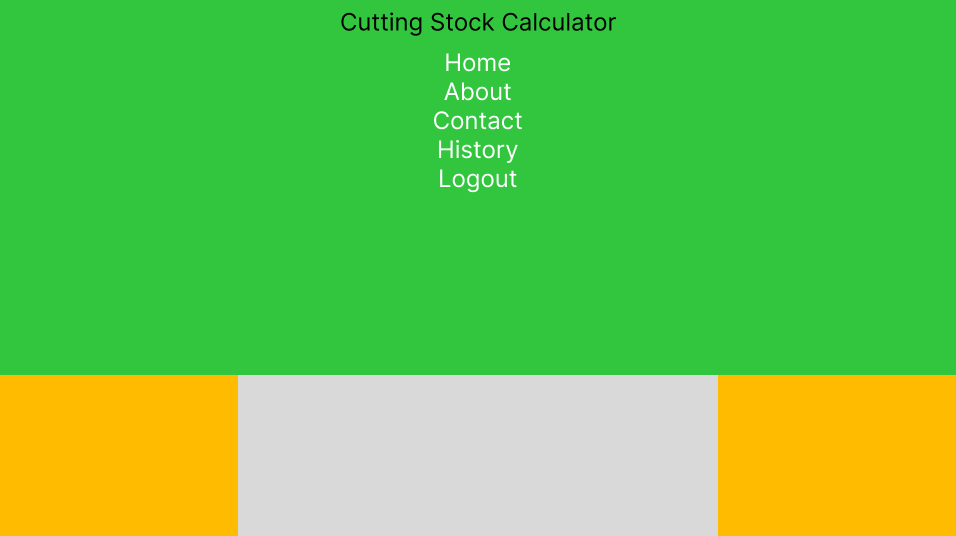
*Figure 7. Output illustrating area used and descending canvas sizes.*



*Figure 8. 2D option selected revealing different inputs*



*Figure 9. Drop down menu selected revealing options to jump to other parts of website*



*Figure 10. Logged in users will be able to see history of work and logout option replaces login*

## **5. References:**

[1] Mapbox, "Shelf-Pack," GitHub repository, 2023. [Online]. Available:<https://github.com/mapbox/shelf-pack>. [Accessed: 30-Sept-2024].

[2] S. Bekaye, "GreedyPacker," GitHub repository, 2023. [Online]. Available:<https://github.com/solomon-b/greedypacker>. [Accessed: 19-Oct-2024].