Data Structure Visualizer

"Visualize the basics"

KARAN SHAH 2020400048
PRATHAMESH PAWAR 202040040
MIRAT SHAH 2020400049

Problem statement

To help visualizing some basic Data Structures (linear DS,Binary tree,BST and heap)

The whole project is uploaded on Github.
The link can be found here:
https://github.com/miratshah27/visualizer

We have also hosted the project so
To run the project simply go to:
https://miratshah27.github.io/visualizer/

The entire presentation link of canva:
https://www.canva.com/design/DAEzRP__6Pk/view?utm_content=DAEzRP__6Pk&utm_campaign=designshare&utm_medium=link&utm_source=publishsharelink

01

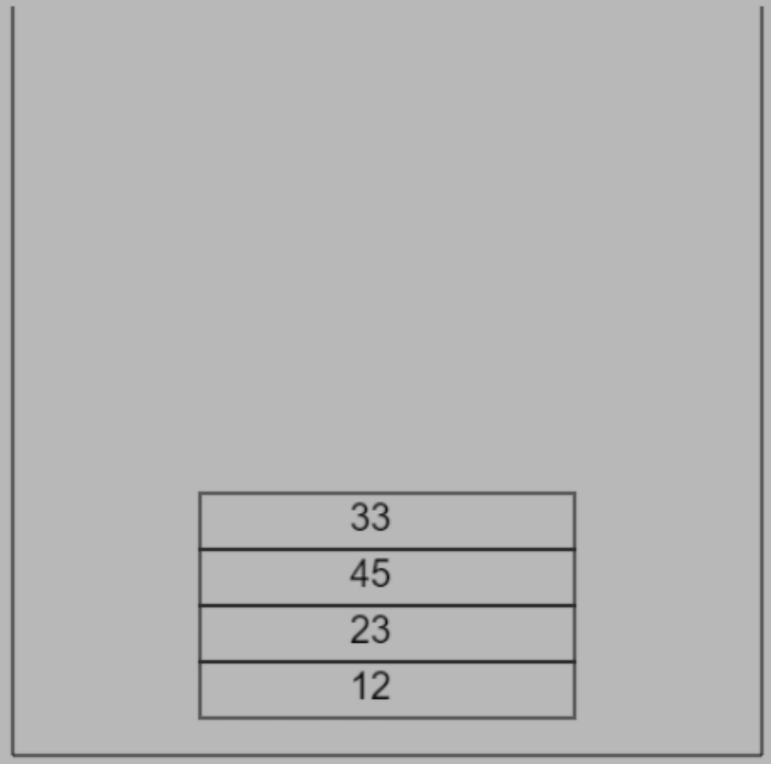
LINEAR DATA STRUCTURES
STACK, QUEUE

STACK

Linear Data Structures Visualization

Stack

Queue



QUEUE

Linear Data Structures Visualization

Stack

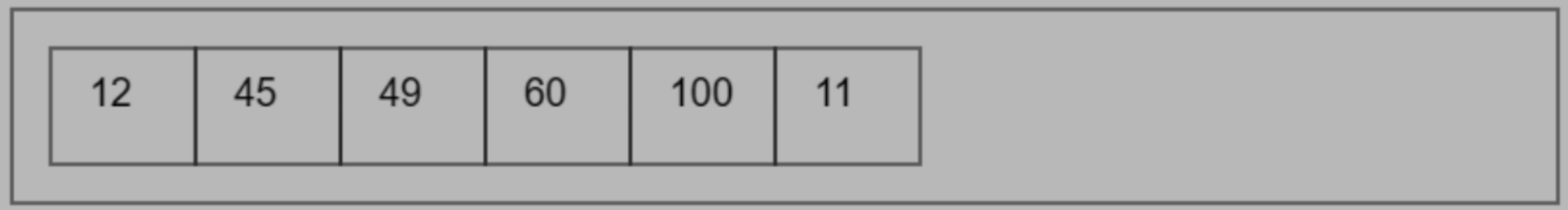
Queue

Queue

Enqueue

Dequeue

Clear the canvas : Clear



01

BINARY TREE

BINARY TREE

What is a Binary Tree?

- A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

CODE USED

```
function insertBT(){
    var data = btInput.value;
    if(data == ""){
        alert("Enter valid input");
        return;
    }
    let node = new TreeNode(parseInt(data));
    if(root == null){
        node.cordX = 500;
        node.cordY = 100;
        this.level = 0;
        root = node;
        paintNode(node.cordX, node.cordY, node.data);
    }
    else if(queue[0].left == null){
        queue[0].left = node;
        node.level = queue[0].level+1;
        node.cordX = queue[0].cordX-(1000/(Math.pow(2, node.level)+1)/2);
        node.cordY = queue[0].cordY+60;
        paintNode(node.cordX, node.cordY, node.data);
        paintLine(node.cordX, node.cordY, queue[0].cordX, queue[0].cordY, "left");
    }
    else{
        queue[0].right = node;
        node.level = queue[0].level+1;
        node.cordX = queue[0].cordX+(1000/(Math.pow(2, node.level)+1)/2);
        node.cordY = queue[0].cordY + 60;
        paintNode(node.cordX, node.cordY, node.data);
        paintLine(node.cordX, node.cordY, queue[0].cordX, queue[0].cordY, "right");
        queue.shift();
    }
    queue.push(node);
    btInput.value = null;
    console.log("Inserted "+data);
}
```

Binary Tree

Insert

Search

Preorder

Inorder

Postorder

Binary Search Tree

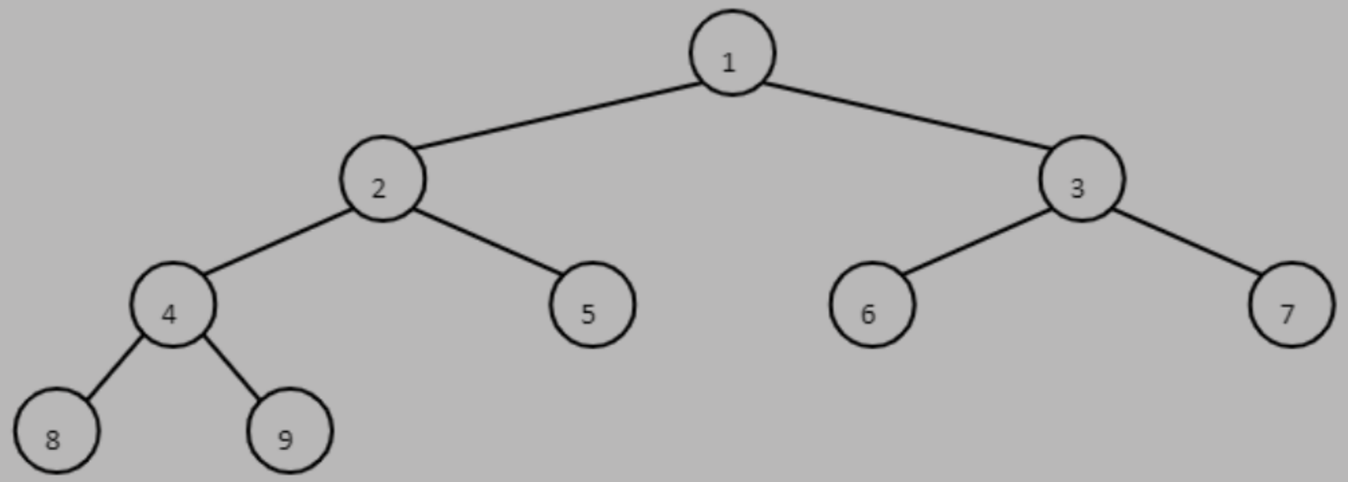
Insert

Search

Preorder

Inorder

Postorder



02

BINARY SEARCH TREE

BINARY SEARCH TREE

What is a Binary search Tree?

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node’s key.
- The right subtree of a node contains only nodes with keys greater than the node’s key.
- The left and right subtree each must also be a binary search tree.

Binary Search Tree

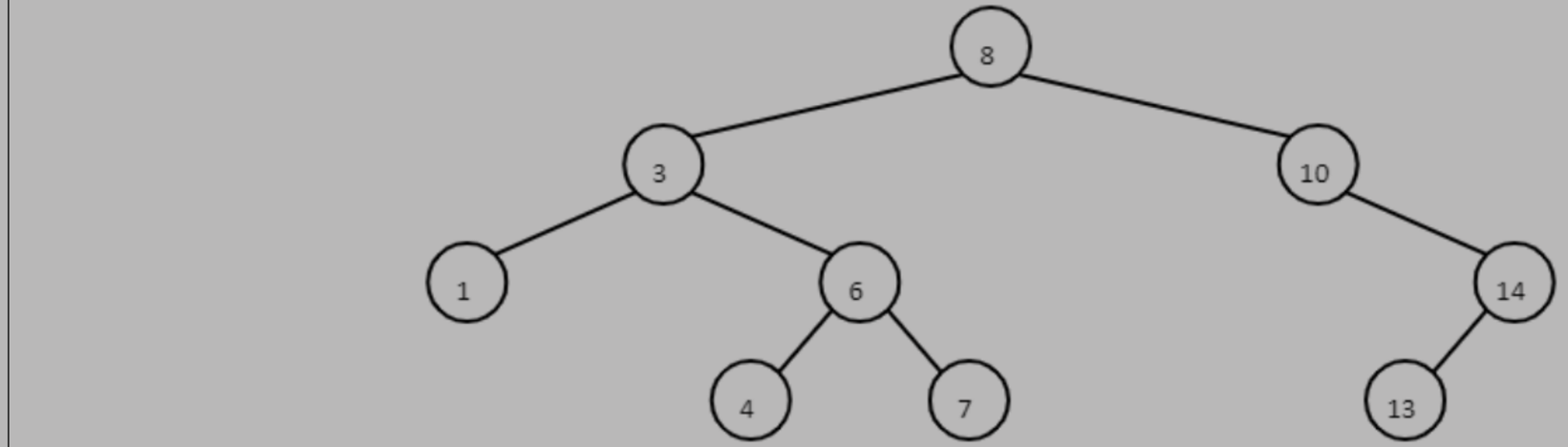
Insert

Search

Preorder

Inorder

Postorder



CODE USED

```
35  function insertBST(data){
36      console.log("Inserted in BST "+data);
37      let node = new TreeNode(data);
38      if(bstRoot.data == null){ ···
45      }
46      let prev = new TreeNode(null);
47      let temp = bstRoot;
48      while(temp!=null){
49          if(data < temp.data){ ···
52          }
53          else if(data > temp.data){ ···
56          }
57      }
58      if(data < prev.data){
59          console.log(prev);
60          node.level = prev.level+1;
61          node.cordX = prev.cordX-(1000/(Math.pow(2, node.level)+1)/2);
62          node.cordY = prev.cordY+60;
63          prev.left = node;
64          paintNode(node.cordX, node.cordY, node.data);
65          paintLine(node.cordX, node.cordY, prev.cordX, prev.cordY, "left");
66          return;
67      }
68      else{ ···
77      }
78  }
79
```

```
async function searchBST(data){
    // await sleep(1000);
    let temp = bstRoot;
    if(temp != null && temp.data == data){
        await sleep(1000);
        paintSearchCircle(bstRoot,'#00fcca');
        return;
    }
    while(temp.data != data){
        if(data < temp.data){ ···
        }
        else if(data > temp.data){ ···
        }
    }
    if(temp.data == data){
        await sleep(1000);
        paintSearchCircle(temp, '#00fcca');
        return;
    }
    else
    {
        alert("Element Not found");
    }
}
```

03

HEAP

What is a Heap?

A Heap is a special Tree-based data structure in which the tree is a complete binary tree. Generally, Heaps can be of two types:

1. Max-Heap: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it’s children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. Min-Heap: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it’s children. The same property must be recursively true for all sub-trees in that Binary Tree.

Heap - Array & Tree based Prototype

Insert the new Element :

Insert

Extract the Maximum :

Extract

Extract Kth Maximum :

Extract

Increase the Key :

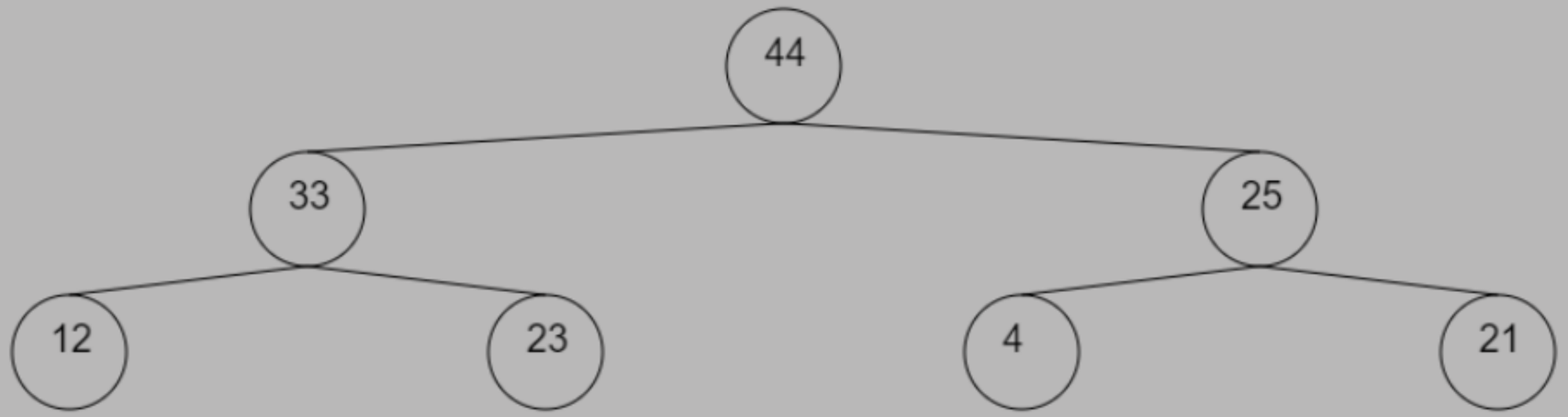
Extract

Heap Sort

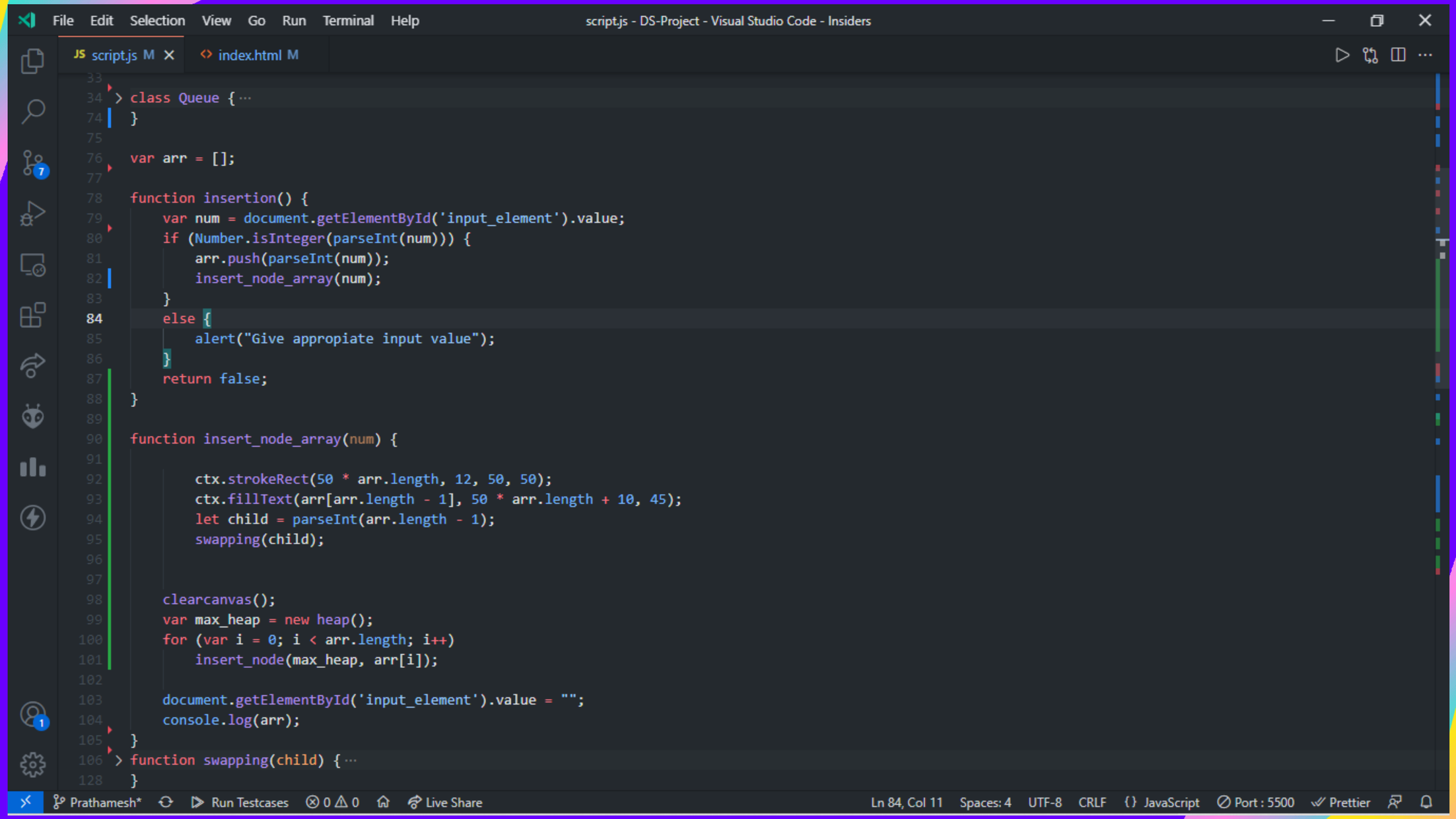
Clear the canvas :

Clear

44	33	25	12	23	4	21



CODE USED


File Edit Selection View Go Run Terminal Help
script.js - DS-Project - Visual Studio Code - Insiders
script.js M
index.html M
class Queue {···
}
var arr = [];
function insertion() {
var num = document.getElementById('input_element').value;
if (Number.isInteger(parseInt(num))) {
arr.push(parseInt(num));
insert_node_array(num);
}
else {
alert("Give appropiate input value");
}
return false;
}
function insert_node_array(num) {
ctx.strokeRect(50 * arr.length, 12, 50, 50);
ctx.fillText(arr[arr.length - 1], 50 * arr.length + 10, 45);
let child = parseInt(arr.length - 1);
swapping(child);
clearcanvas();
var max_heap = new heap();
for (var i = 0; i < arr.length; i++)
insert_node(max_heap, arr[i]);
document.getElementById('input_element').value = "";
console.log(arr);
}
function swapping(child) {···
}
Prathamesh*
Run Testcases
0 0
Live Share
Ln 84, Col 11
Spaces: 4
UTF-8
CRLF
JavaScript
Port : 5500
Prettier


THANK
YOU