**Component Class**

import {

checkParams

} from "./Config";

class **DynamicGrid** {

constructor (config) {

checkParams(config);

if (config.container instanceof HTMLElement) {

this.container = config.container;

this.containerClass = config.container.className;

}

else {

this.containerClass = config.container;

this.container = document.querySelector(config.container);

}

this.items = this.container.children;

this.static = config.static || false;

this.size = config.items;

this.spacing = config.spacing;

this.maxColumns = config.maxColumns || false;

this.useMin = config.useMin || false;

this.useTransform = config.useTransform;

this.animate = config.animate || false;

this.started = false;

this.center = config.center;

this.init();

}

/\***Events** \*/

**init** () {

if (!this.ready() || this.started) return;

this.container.style.position = "relative";

for (let i = 0; i < this.items.length; i++) {

let style = this.items[i].style;

style.position = "absolute";

if (this.animate) {

style.transition = `${this.useTransform ? "transform" : "top, left"} 0.2s ease`;

}

}

this.started = true;

}

colWidth () {

return this.items[0].getBoundingClientRect().width + this.spacing;

}

setup () {

let width = this.container.getBoundingClientRect().width;

let colWidth = this.colWidth();

let numCols = Math.floor(width/colWidth) || 1;

let cols = [];

if (this.maxColumns && numCols > this.maxColumns) {

numCols = this.maxColumns;

}

for (let i = 0; i < numCols; i++) {

cols[i] = {height: 0, index: i};

}

let wSpace = width - numCols \* colWidth + this.spacing;

return {cols, wSpace};

}

nextCol (cols, i) {

if (this.useMin) {

return getMin(cols);

}

return cols[i % cols.length];

}

positionItems () {

let { cols, wSpace } = this.setup();

let maxHeight = 0;

let colWidth = this.colWidth();

wSpace = this.center ? Math.floor(wSpace / 2) : 0;

for (let i = 0; i < this.items.length; i++) {

let col = this.nextCol(cols, i);

let item = this.items[i];

let topspacing = col.height ? this.spacing : 0;

let left = col.index \* colWidth + wSpace + "px";

let top = col.height + topspacing + "px";

if(this.useTransform){

item.style.transform = `translate(${left}, ${top})`;

}

else{

item.style.top = top;

item.style.left = left;

}

col.height += item.getBoundingClientRect().height + topspacing;

if(col.height > maxHeight){

maxHeight = col.height;

}

}

this.container.style.height = maxHeight + this.spacing + "px";

}

ready () {

if (this.static) return true;

return this.items.length >= this.size;

}

getReady () {

let interval = setInterval(() => {

this.container = document.querySelector(this.containerClass);

this.items = this.container.children;

if (this.ready()) {

clearInterval(interval);

this.init();

this.listen();

}

}, 100);

}

listen () {

if (this.ready()) {

let timeout;

window.addEventListener("resize", () => {

if (!timeout){

timeout = setTimeout(() => {

this.positionItems();

timeout = null;

}, 200);

}

});

this.positionItems();

}

else this.getReady();

}

}

export default DynamicGrid;

**Config Class**

const checkParams = config => {

const DEFAULT\_SPACING = 25;

const booleanProps = ["useTransform", "center"];

if (!config) {

throw new Error("No config object has been provided.");

}

for(let prop of booleanProps){

if(typeof config[prop] !== "boolean"){

config[prop] = true;

}

}

if(typeof config.gutter !== "number"){

config.gutter = DEFAULT\_SPACING;

}

if (!config.container) error("container");

if (!config.items && !config.static) error("items or static");

};

const error = prop => {

throw new Error(`Missing property '${prop}' in DynamicGrid config`);

};

export {checkParams};