```
import moment from "moment";
import IntlMessage from "components/util-components/IntlMessage";
import { isEmpty } from "lodash";
import { Grid } from "antd";
import {
  FilePptTwoTone,
  FileWordOutlined,
  FileZipOutlined,
  FileImageOutlined,
  FileExcelTwoTone,
  FileOutlined,
} from "@ant-design/icons";
class Utils {
  static GetFileIcon(extension) {
    switch (extension) {
      case "xlsx":
      case "xls":
        return <FileExcelTwoTone style={{ fontSize: "24px" }} />;
      case "doc":
      case "docx":
        return <FileWordOutlined style={{ fontSize: "24px" }} />;
      case "zip":
      case "rar":
      case "tar":
        return <FileZipOutlined style={{ fontSize: "24px" }} />;
      case "pdf":
        return <FilePptTwoTone style={{ fontSize: "24px" }} />;
      case "png":
      case "jpg":
      case "gif":
      case "bmp":
        return <FileImageOutlined style={{ fontSize: "24px" }} />;
      default:
        return <FileOutlined style={{ fontSize: "24px" }} />;
    }
  }
  /**
   * Get first character from first & last sentences of a username
   * @param {String} name - Username
   * @return {String} 2 characters string
   */

  static removeVietnameseTones(str) {
    str = str.replace(/à|á|ạ|ả|ã|â|ầ|ấ|ậ|ẩ|ẫ|ă|ằ|ắ|ặ|ẳ|ẵ/g, "a");
    str = str.replace(/è|é|ẹ|ẻ|ẽ|ê|ề|ế|ệ|ể|ễ/g, "e");
    str = str.replace(/ì|í|ị|ỉ|ĩ/g, "i");
    str = str.replace(/ò|ó|ọ|ỏ|õ|ô|ồ|ố|ộ|ổ|ỗ|ơ|ờ|ớ|ợ|ở|ỡ/g, "o");
    str = str.replace(/ù|ú|ụ|ủ|ũ|ư|ừ|ứ|ự|ử|ữ/g, "u");
    str = str.replace(/ỳ|ý|ỵ|ỷ|ỹ/g, "y");
    str = str.replace(/đ/g, "d");
    str = str.replace(/À|Á|Ạ|Ả|Ã|Â|Ầ|Ấ|Ậ|Ẩ|Ẫ|Ă|Ằ|Ắ|Ặ|Ẳ|Ẵ/g, "A");
    str = str.replace(/È|É|Ẹ|Ẻ|Ẽ|Ê|Ề|Ế|Ệ|Ể|Ễ/g, "E");
    str = str.replace(/Ì|Í|Ị|Ỉ|Ĩ/g, "I");
    str = str.replace(/Ò|Ó|Ọ|Ỏ|Õ|Ô|Ồ|Ố|Ộ|Ổ|Ỗ|Ơ|Ờ|Ớ|Ợ|Ở|Ỡ/g, "O");
    str = str.replace(/Ù|Ú|Ụ|Ủ|Ũ|Ư|Ừ|Ứ|Ự|Ử|Ữ/g, "U");
    str = str.replace(/Ỳ|Ý|Ỵ|Ỷ|Ỹ/g, "Y");
    str = str.replace(/Đ/g, "D");
    // Some system encode vietnamese combining accent as individual utf-8 characters
    // Một vài bộ encode coi các dấu mũ, dấu chữ như một kí tự riêng biệt nên thêm hai dòng này
    str = str.replace(/\u0300|\u0301|\u0303|\u0309|\u0323/g, ""); // ̀ ́ ̃ ̉ ̣  huyền, sắc, ngã, hỏi, nặng
    str = str.replace(/\u02C6|\u0306|\u031B/g, ""); // ˆ ̆ ̛  Â, Ê, Ă, Ơ, Ư
    // Remove extra spaces
    // Bỏ các khoảng trắng liền nhau
    str = str.replace(/ + /g, " ");
    str = str.trim();
    // Remove punctuations
    // Bỏ dấu câu, kí tự đặc biệt
    str = str.replace(
      // eslint-disable-next-line no-useless-escape
      /!|@|%|\^|\*|\(|\)|\+|\=|\<|\>|\?|\/|,|\.|\:|\;|\'|\"|\&|\#|\[|\]|~|\$|_|`|-|{|}|\||\\/g,
      " "
```

```
    );
    return str;
  }

  static getNameInitial(name) {
    let initials = name.match(/\b\w/g) || [];
    return ((initials.shift() || "") + (initials.pop() || "")).toUpperCase();
  }

  /**
   * Get current path related object from Navigation Tree
   * @param {Array} navTree - Navigation Tree from directory 'configs/NavigationConfig'
   * @param {String} path - Location path you looking for e.g '/app/dashboards/analytic'
   * @return {Object} object that contained the path string
   */
  static getRouteInfo(navTree, path) {
    if (navTree.path === path) {
      return navTree;
    }
    let route;
    for (let p in navTree) {
      if (navTree.hasOwnProperty(p) && typeof navTree[p] === "object") {
        route = this.getRouteInfo(navTree[p], path);
        if (route) {
          return route;
        }
      }
    }
    return route;
  }

  /**
   * Get accessible color contrast
   * @param {String} hex - Hex color code e.g '#3e82f7'
   * @return {String} 'dark' or 'light'
   */
  static getColorContrast(hex) {
    if (!hex) {
      return "dark";
    }
    const threshold = 130;
    const hRed = hexToR(hex);
    const hGreen = hexToG(hex);
    const hBlue = hexToB(hex);
    function hexToR(h) {
      return parseInt(cutHex(h).substring(0, 2), 16);
    }
    function hexToG(h) {
      return parseInt(cutHex(h).substring(2, 4), 16);
    }
    function hexToB(h) {
      return parseInt(cutHex(h).substring(4, 6), 16);
    }
    function cutHex(h) {
      return h.charAt(0) === "#" ? h.substring(1, 7) : h;
    }
    const cBrightness = (hRed * 299 + hGreen * 587 + hBlue * 114) / 1000;
    if (cBrightness > threshold) {
      return "dark";
    } else {
      return "light";
    }
  }

  /**
   * Darken or lighten a hex color
   * @param {String} color - Hex color code e.g '#3e82f7'
   * @param {Number} percent - Percentage -100 to 100, positive for lighten, negative for darken
   * @return {String} Darken or lighten color
   */
  static shadeColor(color, percent) {
    let R = parseInt(color.substring(1, 3), 16);
    let G = parseInt(color.substring(3, 5), 16);
    let B = parseInt(color.substring(5, 7), 16);
```

```
  R = parseInt((R * (100 + percent)) / 100);
  G = parseInt((G * (100 + percent)) / 100);
  B = parseInt((B * (100 + percent)) / 100);
  R = R < 255 ? R : 255;
  G = G < 255 ? G : 255;
  B = B < 255 ? B : 255;
  const RR =
    R.toString(16).length === 1 ? `0${R.toString(16)}` : R.toString(16);
  const GG =
    G.toString(16).length === 1 ? `0${G.toString(16)}` : G.toString(16);
  const BB =
    B.toString(16).length === 1 ? `0${B.toString(16)}` : B.toString(16);
  return `#${RR}${GG}${BB}`;
}

/**
 * Convert RGBA to HEX
 * @param {String} rgba - RGBA color code e.g 'rgba(197, 200, 198, .2)')'
 * @return {String} HEX color
 */
static rgbaToHex(rgba) {
  const trim = (str) => str.replace(/^\s+|\s+$/gm, "");
  const inParts = rgba.substring(rgba.indexOf("(")).split(","),
    r = parseInt(trim(inParts[0].substring(1)), 10),
    g = parseInt(trim(inParts[1]), 10),
    b = parseInt(trim(inParts[2]), 10),
    a = parseFloat(
      trim(inParts[3].substring(0, inParts[3].length - 1))
    ).toFixed(2);
  const outParts = [
    r.toString(16),
    g.toString(16),
    b.toString(16),
    Math.round(a * 255)
      .toString(16)
      .substring(0, 2),
  ];

  outParts.forEach(function (part, i) {
    if (part.length === 1) {
      outParts[i] = "0" + part;
    }
  });
  return `#${outParts.join("")}`;
}

/**
 * Returns either a positive or negative
 * @param {Number} number - number value
 * @param {any} positive - value that return when positive
 * @param {any} negative - value that return when negative
 * @return {any} positive or negative value based on param
 */
static getSignNum(number, positive, negative) {
  if (number > 0) {
    return positive;
  }
  if (number < 0) {
    return negative;
  }
  return null;
}

/**
 * Returns either ascending or descending value
 * @param {Object} a - antd Table sorter param a
 * @param {Object} b - antd Table sorter param b
 * @param {String} key - object key for compare
 * @return {any} a value minus b value
 */
static antdTableSorter(a, b, key) {
  if (typeof a[key] === "number" && typeof b[key] === "number") {
    return a[key] - b[key];
  }
```

```
  if (typeof a[key] === "string" && typeof b[key] === "string") {
    a = a[key].toLowerCase();
    b = b[key].toLowerCase();
    return a > b ? -1 : b > a ? 1 : 0;
  }
  return;
}

/**
 * Filter array of object
 * @param {Array} list - array of objects that need to filter
 * @param {String} key - object key target
 * @param {any} value  - value that excluded from filter
 * @return {Array} a value minus b value
 */
static filterArray(list, key, value) {
  let data = list;
  if (list) {
    data = list.filter((item) => item[key] === value);
  }
  return data;
}

/**
 * Remove object from array by value
 * @param {Array} list - array of objects
 * @param {String} key - object key target
 * @param {any} value  - target value
 * @return {Array} Array that removed target object
 */
static deleteArrayRow(list, key) {
  let data = list;
  if (list) {
    data = list.filter((_, index) => index !== key);
  }
  return data;
}
static setLocale = (localeKey, isLocaleOn = true) =>
  isLocaleOn ? <IntlMessage id={localeKey} /> : localeKey.toString();

static setLanguage = (textVi, textEn, locale) => {
  return locale === "vi" ? textVi : textEn;
};
/**
 * Wild card search on all property of the object
 * @param {Number | String} input - any value to search
 * @param {Array} list - array for search
 * @return {Array} array of object contained keyword
 */
static wildCardSearch(list, input) {
  const searchText = (item) => {
    for (let key in item) {
      if (item[key] == null) {
        continue;
      }
      if (
        item[key]
          .toString()
          .toUpperCase()
          .indexOf(input.toString().toUpperCase()) !== -1
      ) {
        return true;
      }
    }
  };
  list = list.filter((value) => searchText(value));
  return list;
}

/**
 * Get Breakpoint
 * @param {Object} screens - Grid.useBreakpoint() from antd
 * @return {Array} array of breakpoint size
```

```
 */
static getBreakPoint(screens) {
  let breakpoints = [];
  for (const key in screens) {
    if (screens.hasOwnProperty(key)) {
      const element = screens[key];
      if (element) {
        breakpoints.push(key);
      }
    }
  }
  return breakpoints;
}
static formatDate = (dateStr, isDate = false, ignoreDay = false) => {
  const date = moment(dateStr);
  const dateFormatted = isDate
    ? date.format("DD/MM/YYYY")
    : date.format("DD/MM/YYYY HH:mm");

  if (ignoreDay) {
    return isDate ? date.format("MM/YYYY") : date.format("MM/YYYY HH:mm");
  }

  return dateFormatted;
};
static formatTime = (dateStr) => {
  const date = moment(dateStr);
  const dateFormatted = date.format("HH:mm");
  const sliceIndex = 5;
  return dateFormatted.slice(0, sliceIndex) + dateFormatted.slice(sliceIndex);
};

static validatePhoneNumberVN() {
  return {
    validator(_, value) {
      if (isEmpty(value)) return Promise.resolve();
      if (/((^(\+84|84|0084|0){1})(2|3|5|7|8|9))+([0-9]{8})$/.test(value)) {
        return Promise.resolve();
      }
      return Promise.reject(
        new Error("Số điện thoại bắt đầu bằng +84, 84, 0xx")
      );
    },
  };
}
static getNameLetter = (name) => {
  if (!name) return "";
  let str = "";
  name.split(" ")?.map((i) => (str += i.charAt(0)));
  return str.toUpperCase();
};

static checkPermitValue = (crrValue = 0, permitList = [], key) => {
  let isPermit = false;
  if (!permitList) return isPermit;
  const permitOld = permitList.find((i) => i.permit === key);
  if (isEmpty(permitList) || isEmpty(permitOld)) isPermit = false;
  else {
    isPermit = (permitOld.actions & crrValue) !== 0;
  }
  return isPermit;
};

static CheckNumerByPermit = (dataList = [], key) => {
  const data = dataList.find((i) => i.key === key);
  return data?.baget || 0;
};
static layoutMobile = (layout = "lg") => {
  const { useBreakpoint } = Grid;
  const screens = this.getBreakPoint(useBreakpoint());
  return screens.length === 0 ? false : !screens.includes(layout);
};
// type: number/money
static formatterNumber(val, type = "number") {
```

```
    if (!val) return;
    let formater = new Intl.NumberFormat("vi-VN", {
      maximumFractionDigits: 5,
    });
    if (type === "money") {
      formater = new Intl.NumberFormat("vi-VN", {
        style: "currency",
        currency: "VND",
      });
    }
    return formater.format(val);
  }
  static parserNumber = (val) => {
    if (!val) return 0;
    return Number.parseFloat(
      // eslint-disable-next-line no-useless-escape
      val.replace(/\$\s?|(\.*)/g, "").replace(/(\,{1})/g, ".")
    ).toFixed(5);
  };
  static queryToObject(queryString) {
    return Object.fromEntries(new URLSearchParams(queryString));
  }
  //reset hour filter to zero
  static resetTimeToStartOfDay = (date) => {
    if (!date) return null;
    const newDate = new Date(date);
    newDate.setHours(0, 0, 0, 0);
    return newDate;
  };

  static setTimeToEndOfDay = (date) => {
    if (!date) return null;
    const newDate = new Date(date);
    newDate.setHours(23, 59, 59, 999);
    return newDate;
  };
}

export default Utils;
```