```
/**
 * Swiper 6.8.3
 * Most modern mobile touch slider and framework with hardware accelerated transitions
 * https://swiperjs.com
 *
 * Copyright 2014-2021 Vladimir Kharlampidi
 *
 * Released under the MIT License
 *
 * Released on: August 20, 2021
 */

(function (global, factory) {
  typeof exports === 'object' && typeof module !== 'undefined' ? module.exports = factory() :
  typeof define === 'function' && define.amd ? define(factory) :
  (global = typeof globalThis !== 'undefined' ? globalThis : global || self, global.Swiper =
factory());
}(this, (function () { 'use strict';

  function _defineProperties(target, props) {
    for (var i = 0; i < props.length; i++) {
      var descriptor = props[i];
      descriptor.enumerable = descriptor.enumerable || false;
      descriptor.configurable = true;
      if ("value" in descriptor) descriptor.writable = true;
      Object.defineProperty(target, descriptor.key, descriptor);
    }
  }

  function _createClass(Constructor, protoProps, staticProps) {
    if (protoProps) _defineProperties(Constructor.prototype, protoProps);
    if (staticProps) _defineProperties(Constructor, staticProps);
    return Constructor;
  }

  function _extends() {
    _extends = Object.assign || function (target) {
      for (var i = 1; i < arguments.length; i++) {
        var source = arguments[i];

        for (var key in source) {
          if (Object.prototype.hasOwnProperty.call(source, key)) {
            target[key] = source[key];
          }
        }
      }

      return target;
    };

    return _extends.apply(this, arguments);
  }

  /**
   * SSR Window 3.0.0
   * Better handling for window object in SSR environment
   * https://github.com/nolimits4web/ssr-window
   *
   * Copyright 2020, Vladimir Kharlampidi
   *
   * Licensed under MIT
   *
   * Released on: November 9, 2020
   */
  /* eslint-disable no-param-reassign */
  function isObject$1(obj) {
    return obj !== null && typeof obj === 'object' && 'constructor' in obj && obj.constructor ===
Object;
  }

  function extend$1(target, src) {
    if (target === void 0) {
      target = {};
    }

    if (src === void 0) {
      src = {};
    }

    Object.keys(src).forEach(function (key) {
      if (typeof target[key] === 'undefined') target[key] = src[key];else if (isObject$1(src[key]) &&
isObject$1(target[key]) && Object.keys(src[key]).length > 0) {
        extend$1(target[key], src[key]);
      }
    });
  }

  var ssrDocument = {
    body: {},
    addEventListener: function addEventListener() {},
    removeEventListener: function removeEventListener() {},
    activeElement: {
      blur: function blur() {},
      nodeName: ''
    },
    querySelector: function querySelector() {
      return null;
    },
    querySelectorAll: function querySelectorAll() {
      return [];
    },
    getElementById: function getElementById() {
      return null;
    },
    createEvent: function createEvent() {
      return {
        initEvent: function initEvent() {}
      };
    },
    createElement: function createElement() {
      return {
        children: [],
        childNodes: [],
        style: {},
        setAttribute: function setAttribute() {},
        getElementsByTagName: function getElementsByTagName() {
          return [];
        }
      };
    },
    createElementNS: function createElementNS() {
      return {};
    },
    importNode: function importNode() {
      return null;
    },
    location: {
      hash: '',
      host: '',
      hostname: '',
      href: '',
      origin: '',
      pathname: '',
      protocol: '',
      search: ''
    }
  };

  function getDocument() {
    var doc = typeof document !== 'undefined' ? document : {};
    extend$1(doc, ssrDocument);
    return doc;
  }

  var ssrWindow = {
    document: ssrDocument,
    navigator: {
      userAgent: ''
    },
    location: {
      hash: '',
      host: '',
      hostname: '',
      href: '',
      origin: '',
      pathname: '',
      protocol: '',
      search: ''
    },
    history: {
      replaceState: function replaceState() {},
      pushState: function pushState() {},
      go: function go() {},
      back: function back() {}
    },
    CustomEvent: function CustomEvent() {
      return this;
    },
    addEventListener: function addEventListener() {},
    removeEventListener: function removeEventListener() {},
    getComputedStyle: function getComputedStyle() {
      return {
        getPropertyValue: function getPropertyValue() {
          return '';
        }
      };
    },
    Image: function Image() {},
    Date: function Date() {},
    screen: {},
    setTimeout: function setTimeout() {},
    clearTimeout: function clearTimeout() {},
    matchMedia: function matchMedia() {
      return {};
    },
    requestAnimationFrame: function requestAnimationFrame(callback) {
      if (typeof setTimeout === 'undefined') {
        callback();
        return null;
      }

      return setTimeout(callback, 0);
    },
    cancelAnimationFrame: function cancelAnimationFrame(id) {
      if (typeof setTimeout === 'undefined') {
        return;
      }

      clearTimeout(id);
    }
  };

  function getWindow() {
    var win = typeof window !== 'undefined' ? window : {};
    extend$1(win, ssrWindow);
    return win;
  }

  /**
   * Dom7 3.0.0
   * Minimalistic JavaScript library for DOM manipulation, with a jQuery-compatible API
   * https://framework7.io/docs/dom7.html
   *
   * Copyright 2020, Vladimir Kharlampidi
   *
   * Licensed under MIT
   *
   * Released on: November 9, 2020
   */

  function _inheritsLoose(subClass, superClass) {
    subClass.prototype = Object.create(superClass.prototype);
    subClass.prototype.constructor = subClass;
    subClass.__proto__ = superClass;
  }

  function _getPrototypeOf(o) {
    _getPrototypeOf = Object.setPrototypeOf ? Object.getPrototypeOf : function _getPrototypeOf(o) {
      return o.__proto__ || Object.getPrototypeOf(o);
    };
    return _getPrototypeOf(o);
  }

  function _setPrototypeOf(o, p) {
    _setPrototypeOf = Object.setPrototypeOf || function _setPrototypeOf(o, p) {
      o.__proto__ = p;
      return o;
    };

    return _setPrototypeOf(o, p);
  }

  function _isNativeReflectConstruct() {
    if (typeof Reflect === "undefined" || !Reflect.construct) return false;
    if (Reflect.construct.sham) return false;
    if (typeof Proxy === "function") return true;

    try {
      Date.prototype.toString.call(Reflect.construct(Date, [], function () {}));
      return true;
    } catch (e) {
      return false;
    }
  }

  function _construct(Parent, args, Class) {
    if (_isNativeReflectConstruct()) {
      _construct = Reflect.construct;
    } else {
      _construct = function _construct(Parent, args, Class) {
        var a = [null];
        a.push.apply(a, args);
        var Constructor = Function.bind.apply(Parent, a);
        var instance = new Constructor();
        if (Class) _setPrototypeOf(instance, Class.prototype);
        return instance;
      };
    }

    return _construct.apply(null, arguments);
  }

  function _isNativeFunction(fn) {
    return Function.toString.call(fn).indexOf("[native code]") !== -1;
  }

  function _wrapNativeSuper(Class) {
    var _cache = typeof Map === "function" ? new Map() : undefined;

    _wrapNativeSuper = function _wrapNativeSuper(Class) {
      if (Class === null || !_isNativeFunction(Class)) return Class;

      if (typeof Class !== "function") {
        throw new TypeError("Super expression must either be null or a function");
      }

      if (typeof _cache !== "undefined") {
        if (_cache.has(Class)) return _cache.get(Class);

        _cache.set(Class, Wrapper);
      }

      function Wrapper() {
        return _construct(Class, arguments, _getPrototypeOf(this).constructor);
      }

      Wrapper.prototype = Object.create(Class.prototype, {
        constructor: {
          value: Wrapper,
          enumerable: false,
          writable: true,
          configurable: true
        }
      });
      return _setPrototypeOf(Wrapper, Class);
    };

    return _wrapNativeSuper(Class);
  }

  function _assertThisInitialized(self) {
    if (self === void 0) {
      throw new ReferenceError("this hasn't been initialised - super() hasn't been called");
    }

    return self;
  }
  /* eslint-disable no-proto */


  function makeReactive(obj) {
    var proto = obj.__proto__;
    Object.defineProperty(obj, '__proto__', {
```

```
      get: function get() {
        return proto;
      },
      set: function set(value) {
        proto.__proto__ = value;
      }
    });
  }

  var Dom7 = /*#__PURE__*/function (_Array) {
    _inheritsLoose(Dom7, _Array);

    function Dom7(items) {
      var _this;

      _this = _Array.call.apply(_Array, [this].concat(items)) || this;
      makeReactive(_assertThisInitialized(_this));
      return _this;
    }

    return Dom7;
  }( /*#__PURE__*/_wrapNativeSuper(Array));

  function arrayFlat(arr) {
    if (arr === void 0) {
      arr = [];
    }

    var res = [];
    arr.forEach(function (el) {
      if (Array.isArray(el)) {
        res.push.apply(res, arrayFlat(el));
      } else {
        res.push(el);
      }
    });
    return res;
  }

  function arrayFilter(arr, callback) {
    return Array.prototype.filter.call(arr, callback);
  }

  function arrayUnique(arr) {
    var uniqueArray = [];

    for (var i = 0; i < arr.length; i += 1) {
      if (uniqueArray.indexOf(arr[i]) === -1) uniqueArray.push(arr[i]);
    }

    return uniqueArray;
  }

  function qsa(selector, context) {
    if (typeof selector !== 'string') {
      return [selector];
    }

    var a = [];
    var res = context.querySelectorAll(selector);

    for (var i = 0; i < res.length; i += 1) {
      a.push(res[i]);
    }
```

```
      return a;
    }

    function $(selector, context) {
      var window = getWindow();
      var document = getDocument();
      var arr = [];

      if (!context && selector instanceof Dom7) {
        return selector;
      }

      if (!selector) {
        return new Dom7(arr);
      }

      if (typeof selector === 'string') {
        var html = selector.trim();

        if (html.indexOf('<') >= 0 && html.indexOf('>') >= 0) {
          var toCreate = 'div';
          if (html.indexOf('<li') === 0) toCreate = 'ul';
          if (html.indexOf('<tr') === 0) toCreate = 'tbody';
          if (html.indexOf('<td') === 0 || html.indexOf('<th') === 0) toCreate = 'tr';
          if (html.indexOf('<tbody') === 0) toCreate = 'table';
          if (html.indexOf('<option') === 0) toCreate = 'select';
          var tempParent = document.createElement(toCreate);
          tempParent.innerHTML = html;

          for (var i = 0; i < tempParent.childNodes.length; i += 1) {
            arr.push(tempParent.childNodes[i]);
          }
        } else {
          arr = qsa(selector.trim(), context || document);
        } // arr = qsa(selector, document);

      } else if (selector.nodeType || selector === window || selector === document) {
        arr.push(selector);
      } else if (Array.isArray(selector)) {
        if (selector instanceof Dom7) return selector;
        arr = selector;
      }

      return new Dom7(arrayUnique(arr));
    }

    $.fn = Dom7.prototype;

    function addClass() {
      for (var _len = arguments.length, classes = new Array(_len), _key = 0; _key < _len; _key++) {
        classes[_key] = arguments[_key];
      }

      var classNames = arrayFlat(classes.map(function (c) {
        return c.split(' ');
      }));
      this.forEach(function (el) {
        var _el$classList;

        (_el$classList = el.classList).add.apply(_el$classList, classNames);
      });
      return this;
    }

    function removeClass() {
```

```
    for (var _len2 = arguments.length, classes = new Array(_len2), _key2 = 0; _key2 < _len2; _key2++)
{
      classes[_key2] = arguments[_key2];
    }

    var classNames = arrayFlat(classes.map(function (c) {
      return c.split(' ');
    }));
    this.forEach(function (el) {
      var _el$classList2;

      (_el$classList2 = el.classList).remove.apply(_el$classList2, classNames);
    });
    return this;
  }

  function toggleClass() {
    for (var _len3 = arguments.length, classes = new Array(_len3), _key3 = 0; _key3 < _len3; _key3++)
{
      classes[_key3] = arguments[_key3];
    }

    var classNames = arrayFlat(classes.map(function (c) {
      return c.split(' ');
    }));
    this.forEach(function (el) {
      classNames.forEach(function (className) {
        el.classList.toggle(className);
      });
    });
  }

  function hasClass() {
    for (var _len4 = arguments.length, classes = new Array(_len4), _key4 = 0; _key4 < _len4; _key4++)
{
      classes[_key4] = arguments[_key4];
    }

    var classNames = arrayFlat(classes.map(function (c) {
      return c.split(' ');
    }));
    return arrayFilter(this, function (el) {
      return classNames.filter(function (className) {
        return el.classList.contains(className);
      }).length > 0;
    }).length > 0;
  }

  function attr(attrs, value) {
    if (arguments.length === 1 && typeof attrs === 'string') {
      // Get attr
      if (this[0]) return this[0].getAttribute(attrs);
      return undefined;
    } // Set attrs


    for (var i = 0; i < this.length; i += 1) {
      if (arguments.length === 2) {
        // String
        this[i].setAttribute(attrs, value);
      } else {
        // Object
        for (var attrName in attrs) {
          this[i][attrName] = attrs[attrName];
          this[i].setAttribute(attrName, attrs[attrName]);
        }
      }
    }

    return this;
  }

  function removeAttr(attr) {
    for (var i = 0; i < this.length; i += 1) {
      this[i].removeAttribute(attr);
    }

    return this;
  }

  function transform(transform) {
    for (var i = 0; i < this.length; i += 1) {
      this[i].style.transform = transform;
    }

    return this;
  }

  function transition$1(duration) {
    for (var i = 0; i < this.length; i += 1) {
      this[i].style.transitionDuration = typeof duration !== 'string' ? duration + "ms" : duration;
    }

    return this;
  }

  function on() {
    for (var _len5 = arguments.length, args = new Array(_len5), _key5 = 0; _key5 < _len5; _key5++) {
      args[_key5] = arguments[_key5];
    }

    var eventType = args[0],
        targetSelector = args[1],
        listener = args[2],
        capture = args[3];

    if (typeof args[1] === 'function') {
      eventType = args[0];
      listener = args[1];
      capture = args[2];
      targetSelector = undefined;
    }

    if (!capture) capture = false;

    function handleLiveEvent(e) {
      var target = e.target;
      if (!target) return;
      var eventData = e.target.dom7EventData || [];

      if (eventData.indexOf(e) < 0) {
        eventData.unshift(e);
      }

      if ($(target).is(targetSelector)) listener.apply(target, eventData);else {
        var _parents = $(target).parents(); // eslint-disable-line


        for (var k = 0; k < _parents.length; k += 1) {
          if ($(_parents[k]).is(targetSelector)) listener.apply(_parents[k], eventData);
        }
      }
    }

    function handleEvent(e) {
      var eventData = e && e.target ? e.target.dom7EventData || [] : [];

      if (eventData.indexOf(e) < 0) {
        eventData.unshift(e);
      }

      listener.apply(this, eventData);
    }

    var events = eventType.split(' ');
    var j;

    for (var i = 0; i < this.length; i += 1) {
      var el = this[i];

      if (!targetSelector) {
        for (j = 0; j < events.length; j += 1) {
          var event = events[j];
          if (!el.dom7Listeners) el.dom7Listeners = {};
          if (!el.dom7Listeners[event]) el.dom7Listeners[event] = [];
          el.dom7Listeners[event].push({
            listener: listener,
            proxyListener: handleEvent
          });
          el.addEventListener(event, handleEvent, capture);
        }
      } else {
        // Live events
        for (j = 0; j < events.length; j += 1) {
          var _event = events[j];
          if (!el.dom7LiveListeners) el.dom7LiveListeners = {};
          if (!el.dom7LiveListeners[_event]) el.dom7LiveListeners[_event] = [];

          el.dom7LiveListeners[_event].push({
            listener: listener,
            proxyListener: handleLiveEvent
          });

          el.addEventListener(_event, handleLiveEvent, capture);
        }
      }
    }

    return this;
  }

  function off() {
    for (var _len6 = arguments.length, args = new Array(_len6), _key6 = 0; _key6 < _len6; _key6++) {
      args[_key6] = arguments[_key6];
    }

    var eventType = args[0],
        targetSelector = args[1],
        listener = args[2],
        capture = args[3];

    if (typeof args[1] === 'function') {
      eventType = args[0];
      listener = args[1];
      capture = args[2];
      targetSelector = undefined;
    }

    if (!capture) capture = false;
    var events = eventType.split(' ');

    for (var i = 0; i < events.length; i += 1) {
      var event = events[i];

      for (var j = 0; j < this.length; j += 1) {
        var el = this[j];
        var handlers = void 0;

        if (!targetSelector && el.dom7Listeners) {
          handlers = el.dom7Listeners[event];
        } else if (targetSelector && el.dom7LiveListeners) {
          handlers = el.dom7LiveListeners[event];
        }

        if (handlers && handlers.length) {
          for (var k = handlers.length - 1; k >= 0; k -= 1) {
            var handler = handlers[k];

            if (listener && handler.listener === listener) {
              el.removeEventListener(event, handler.proxyListener, capture);
              handlers.splice(k, 1);
            } else if (listener && handler.listener && handler.listener.dom7proxy &&
handler.listener.dom7proxy === listener) {
              el.removeEventListener(event, handler.proxyListener, capture);
              handlers.splice(k, 1);
            } else if (!listener) {
              el.removeEventListener(event, handler.proxyListener, capture);
              handlers.splice(k, 1);
            }
          }
        }
      }
    }

    return this;
  }

  function trigger() {
    var window = getWindow();

    for (var _len9 = arguments.length, args = new Array(_len9), _key9 = 0; _key9 < _len9; _key9++) {
      args[_key9] = arguments[_key9];
    }

    var events = args[0].split(' ');
    var eventData = args[1];

    for (var i = 0; i < events.length; i += 1) {
      var event = events[i];

      for (var j = 0; j < this.length; j += 1) {
        var el = this[j];

        if (window.CustomEvent) {
          var evt = new window.CustomEvent(event, {
            detail: eventData,
            bubbles: true,
            cancelable: true
          });
          el.dom7EventData = args.filter(function (data, dataIndex) {
```

```
            return dataIndex > 0;
          });
          el.dispatchEvent(evt);
          el.dom7EventData = [];
          delete el.dom7EventData;
        }
      }
    }

    return this;
  }

  function transitionEnd$1(callback) {
    var dom = this;

    function fireCallBack(e) {
      if (e.target !== this) return;
      callback.call(this, e);
      dom.off('transitionend', fireCallBack);
    }

    if (callback) {
      dom.on('transitionend', fireCallBack);
    }

    return this;
  }

  function outerWidth(includeMargins) {
    if (this.length > 0) {
      if (includeMargins) {
        var _styles = this.styles();

        return this[0].offsetWidth + parseFloat(_styles.getPropertyValue('margin-right')) +
parseFloat(_styles.getPropertyValue('margin-left'));
      }

      return this[0].offsetWidth;
    }

    return null;
  }

  function outerHeight(includeMargins) {
    if (this.length > 0) {
      if (includeMargins) {
        var _styles2 = this.styles();

        return this[0].offsetHeight + parseFloat(_styles2.getPropertyValue('margin-top')) +
parseFloat(_styles2.getPropertyValue('margin-bottom'));
      }

      return this[0].offsetHeight;
    }

    return null;
  }

  function offset() {
    if (this.length > 0) {
      var window = getWindow();
      var document = getDocument();
      var el = this[0];
      var box = el.getBoundingClientRect();
      var body = document.body;
      var clientTop = el.clientTop || body.clientTop || 0;
      var clientLeft = el.clientLeft || body.clientLeft || 0;
      var scrollTop = el === window ? window.scrollY : el.scrollTop;
      var scrollLeft = el === window ? window.scrollX : el.scrollLeft;
      return {
        top: box.top + scrollTop - clientTop,
        left: box.left + scrollLeft - clientLeft
      };
    }

    return null;
  }

  function styles() {
    var window = getWindow();
    if (this[0]) return window.getComputedStyle(this[0], null);
    return {};
  }

  function css(props, value) {
    var window = getWindow();
    var i;

    if (arguments.length === 1) {
      if (typeof props === 'string') {
        // .css('width')
        if (this[0]) return window.getComputedStyle(this[0], null).getPropertyValue(props);
      } else {
        // .css({ width: '100px' })
        for (i = 0; i < this.length; i += 1) {
          for (var _prop in props) {
            this[i].style[_prop] = props[_prop];
          }
        }

        return this;
      }
    }

    if (arguments.length === 2 && typeof props === 'string') {
      // .css('width', '100px')
      for (i = 0; i < this.length; i += 1) {
        this[i].style[props] = value;
      }

      return this;
    }

    return this;
  }

  function each(callback) {
    if (!callback) return this;
    this.forEach(function (el, index) {
      callback.apply(el, [el, index]);
    });
    return this;
  }

  function filter(callback) {
    var result = arrayFilter(this, callback);
    return $(result);
  }

  function html(html) {
    if (typeof html === 'undefined') {
      return this[0] ? this[0].innerHTML : null;
    }

    for (var i = 0; i < this.length; i += 1) {
      this[i].innerHTML = html;
    }

    return this;
  }

  function text(text) {
    if (typeof text === 'undefined') {
      return this[0] ? this[0].textContent.trim() : null;
    }

    for (var i = 0; i < this.length; i += 1) {
      this[i].textContent = text;
    }

    return this;
  }

  function is(selector) {
    var window = getWindow();
    var document = getDocument();
    var el = this[0];
    var compareWith;
    var i;
    if (!el || typeof selector === 'undefined') return false;

    if (typeof selector === 'string') {
      if (el.matches) return el.matches(selector);
      if (el.webkitMatchesSelector) return el.webkitMatchesSelector(selector);
      if (el.msMatchesSelector) return el.msMatchesSelector(selector);
      compareWith = $(selector);

      for (i = 0; i < compareWith.length; i += 1) {
        if (compareWith[i] === el) return true;
      }

      return false;
    }

    if (selector === document) {
      return el === document;
    }

    if (selector === window) {
      return el === window;
    }

    if (selector.nodeType || selector instanceof Dom7) {
      compareWith = selector.nodeType ? [selector] : selector;

      for (i = 0; i < compareWith.length; i += 1) {
        if (compareWith[i] === el) return true;
      }

      return false;
    }

    return false;
  }
```

```
  function index() {
    var child = this[0];
    var i;

    if (child) {
      i = 0; // eslint-disable-next-line

      while ((child = child.previousSibling) !== null) {
        if (child.nodeType === 1) i += 1;
      }

      return i;
    }

    return undefined;
  }

  function eq(index) {
    if (typeof index === 'undefined') return this;
    var length = this.length;

    if (index > length - 1) {
      return $([]);
    }

    if (index < 0) {
      var returnIndex = length + index;
      if (returnIndex < 0) return $([]);
      return $([this[returnIndex]]);
    }

    return $([this[index]]);
  }

  function append() {
    var newChild;
    var document = getDocument();

    for (var k = 0; k < arguments.length; k += 1) {
      newChild = k < 0 || arguments.length <= k ? undefined : arguments[k];

      for (var i = 0; i < this.length; i += 1) {
        if (typeof newChild === 'string') {
          var tempDiv = document.createElement('div');
          tempDiv.innerHTML = newChild;

          while (tempDiv.firstChild) {
            this[i].appendChild(tempDiv.firstChild);
          }
        } else if (newChild instanceof Dom7) {
          for (var j = 0; j < newChild.length; j += 1) {
            this[i].appendChild(newChild[j]);
          }
        } else {
          this[i].appendChild(newChild);
        }
      }
    }

    return this;
  }

  function prepend(newChild) {
    var document = getDocument();
    var i;
```

```
      var j;

      for (i = 0; i < this.length; i += 1) {
        if (typeof newChild === 'string') {
          var tempDiv = document.createElement('div');
          tempDiv.innerHTML = newChild;

          for (j = tempDiv.childNodes.length - 1; j >= 0; j -= 1) {
            this[i].insertBefore(tempDiv.childNodes[j], this[i].childNodes[0]);
          }
        } else if (newChild instanceof Dom7) {
          for (j = 0; j < newChild.length; j += 1) {
            this[i].insertBefore(newChild[j], this[i].childNodes[0]);
          }
        } else {
          this[i].insertBefore(newChild, this[i].childNodes[0]);
        }
      }

      return this;
    }

    function next(selector) {
      if (this.length > 0) {
        if (selector) {
          if (this[0].nextElementSibling && $(this[0].nextElementSibling).is(selector)) {
            return $([this[0].nextElementSibling]);
          }

          return $([]);
        }

        if (this[0].nextElementSibling) return $([this[0].nextElementSibling]);
        return $([]);
      }

      return $([]);
    }

    function nextAll(selector) {
      var nextEls = [];
      var el = this[0];
      if (!el) return $([]);

      while (el.nextElementSibling) {
        var _next = el.nextElementSibling; // eslint-disable-line

        if (selector) {
          if ($(_next).is(selector)) nextEls.push(_next);
        } else nextEls.push(_next);

        el = _next;
      }

      return $(nextEls);
    }

    function prev(selector) {
      if (this.length > 0) {
        var el = this[0];

        if (selector) {
          if (el.previousElementSibling && $(el.previousElementSibling).is(selector)) {
            return $([el.previousElementSibling]);
          }
          return $([]);
        }

        if (el.previousElementSibling) return $([el.previousElementSibling]);
        return $([]);
      }

      return $([]);
    }

    function prevAll(selector) {
      var prevEls = [];
      var el = this[0];
      if (!el) return $([]);

      while (el.previousElementSibling) {
        var _prev = el.previousElementSibling; // eslint-disable-line

        if (selector) {
          if ($(_prev).is(selector)) prevEls.push(_prev);
        } else prevEls.push(_prev);

        el = _prev;
      }

      return $(prevEls);
    }

    function parent(selector) {
      var parents = []; // eslint-disable-line

      for (var i = 0; i < this.length; i += 1) {
        if (this[i].parentNode !== null) {
          if (selector) {
            if ($(this[i].parentNode).is(selector)) parents.push(this[i].parentNode);
          } else {
            parents.push(this[i].parentNode);
          }
        }
      }

      return $(parents);
    }

    function parents(selector) {
      var parents = []; // eslint-disable-line

      for (var i = 0; i < this.length; i += 1) {
        var _parent = this[i].parentNode; // eslint-disable-line

        while (_parent) {
          if (selector) {
            if ($(_parent).is(selector)) parents.push(_parent);
          } else {
            parents.push(_parent);
          }

          _parent = _parent.parentNode;
        }
      }

      return $(parents);
    }
```

```
  function closest(selector) {
    var closest = this; // eslint-disable-line

    if (typeof selector === 'undefined') {
      return $([]);
    }

    if (!closest.is(selector)) {
      closest = closest.parents(selector).eq(0);
    }

    return closest;
  }

  function find(selector) {
    var foundElements = [];

    for (var i = 0; i < this.length; i += 1) {
      var found = this[i].querySelectorAll(selector);

      for (var j = 0; j < found.length; j += 1) {
        foundElements.push(found[j]);
      }
    }

    return $(foundElements);
  }

  function children(selector) {
    var children = []; // eslint-disable-line

    for (var i = 0; i < this.length; i += 1) {
      var childNodes = this[i].children;

      for (var j = 0; j < childNodes.length; j += 1) {
        if (!selector || $(childNodes[j]).is(selector)) {
          children.push(childNodes[j]);
        }
      }
    }

    return $(children);
  }

  function remove() {
    for (var i = 0; i < this.length; i += 1) {
      if (this[i].parentNode) this[i].parentNode.removeChild(this[i]);
    }

    return this;
  }

  var Methods = {
    addClass: addClass,
    removeClass: removeClass,
    hasClass: hasClass,
    toggleClass: toggleClass,
    attr: attr,
    removeAttr: removeAttr,
    transform: transform,
    transition: transition$1,
    on: on,
    off: off,
    trigger: trigger,
    transitionEnd: transitionEnd$1,
    outerWidth: outerWidth,
    outerHeight: outerHeight,
    styles: styles,
    offset: offset,
    css: css,
    each: each,
    html: html,
    text: text,
    is: is,
    index: index,
    eq: eq,
    append: append,
    prepend: prepend,
    next: next,
    nextAll: nextAll,
    prev: prev,
    prevAll: prevAll,
    parent: parent,
    parents: parents,
    closest: closest,
    find: find,
    children: children,
    filter: filter,
    remove: remove
  };
  Object.keys(Methods).forEach(function (methodName) {
    Object.defineProperty($.fn, methodName, {
      value: Methods[methodName],
      writable: true
    });
  });

  function deleteProps(obj) {
    var object = obj;
    Object.keys(object).forEach(function (key) {
      try {
        object[key] = null;
      } catch (e) {// no getter for object
      }

      try {
        delete object[key];
      } catch (e) {// something got wrong
      }
    });
  }

  function nextTick(callback, delay) {
    if (delay === void 0) {
      delay = 0;
    }

    return setTimeout(callback, delay);
  }

  function now() {
    return Date.now();
  }

  function getComputedStyle$1(el) {
    var window = getWindow();
    var style;

    if (window.getComputedStyle) {
      style = window.getComputedStyle(el, null);
    }

    if (!style && el.currentStyle) {
      style = el.currentStyle;
    }

    if (!style) {
      style = el.style;
    }

    return style;
  }

  function getTranslate(el, axis) {
    if (axis === void 0) {
      axis = 'x';
    }

    var window = getWindow();
    var matrix;
    var curTransform;
    var transformMatrix;
    var curStyle = getComputedStyle$1(el);

    if (window.WebKitCSSMatrix) {
      curTransform = curStyle.transform || curStyle.webkitTransform;

      if (curTransform.split(',').length > 6) {
        curTransform = curTransform.split(', ').map(function (a) {
          return a.replace(',', '.');
        }).join(', ');
      } // Some old versions of Webkit choke when 'none' is passed; pass
      // empty string instead in this case


      transformMatrix = new window.WebKitCSSMatrix(curTransform === 'none' ? '' : curTransform);
    } else {
      transformMatrix = curStyle.MozTransform || curStyle.OTransform || curStyle.MsTransform ||
curStyle.msTransform || curStyle.transform ||
curStyle.getPropertyValue('transform').replace('translate(', 'matrix(1, 0, 0, 1,');
      matrix = transformMatrix.toString().split(',');
    }

    if (axis === 'x') {
      // Latest Chrome and webkits Fix
      if (window.WebKitCSSMatrix) curTransform = transformMatrix.m41; // Crazy IE10 Matrix
      else if (matrix.length === 16) curTransform = parseFloat(matrix[12]); // Normal Browsers
        else curTransform = parseFloat(matrix[4]);
    }

    if (axis === 'y') {
      // Latest Chrome and webkits Fix
      if (window.WebKitCSSMatrix) curTransform = transformMatrix.m42; // Crazy IE10 Matrix
      else if (matrix.length === 16) curTransform = parseFloat(matrix[13]); // Normal Browsers
        else curTransform = parseFloat(matrix[5]);
    }

    return curTransform || 0;
  }

  function isObject(o) {
    return typeof o === 'object' && o !== null && o.constructor &&
Object.prototype.toString.call(o).slice(8, -1) === 'Object';
  }
  function isNode(node) {
    // eslint-disable-next-line
    if (typeof window !== 'undefined' && typeof window.HTMLElement !== 'undefined') {
      return node instanceof HTMLElement;
    }

    return node && (node.nodeType === 1 || node.nodeType === 11);
  }

  function extend() {
    var to = Object(arguments.length <= 0 ? undefined : arguments[0]);
    var noExtend = ['__proto__', 'constructor', 'prototype'];

    for (var i = 1; i < arguments.length; i += 1) {
      var nextSource = i < 0 || arguments.length <= i ? undefined : arguments[i];

      if (nextSource !== undefined && nextSource !== null && !isNode(nextSource)) {
        var keysArray = Object.keys(Object(nextSource)).filter(function (key) {
          return noExtend.indexOf(key) < 0;
        });

        for (var nextIndex = 0, len = keysArray.length; nextIndex < len; nextIndex += 1) {
          var nextKey = keysArray[nextIndex];
          var desc = Object.getOwnPropertyDescriptor(nextSource, nextKey);

          if (desc !== undefined && desc.enumerable) {
            if (isObject(to[nextKey]) && isObject(nextSource[nextKey])) {
              if (nextSource[nextKey].__swiper__) {
                to[nextKey] = nextSource[nextKey];
              } else {
                extend(to[nextKey], nextSource[nextKey]);
              }
            } else if (!isObject(to[nextKey]) && isObject(nextSource[nextKey])) {
              to[nextKey] = {};

              if (nextSource[nextKey].__swiper__) {
                to[nextKey] = nextSource[nextKey];
              } else {
                extend(to[nextKey], nextSource[nextKey]);
              }
            } else {
              to[nextKey] = nextSource[nextKey];
            }
          }
        }
      }
    }

    return to;
  }

  function bindModuleMethods(instance, obj) {
    Object.keys(obj).forEach(function (key) {
      if (isObject(obj[key])) {
        Object.keys(obj[key]).forEach(function (subKey) {
          if (typeof obj[key][subKey] === 'function') {
            obj[key][subKey] = obj[key][subKey].bind(instance);
          }
        });
      }

      instance[key] = obj[key];
    });
  }
  function classesToSelector(classes) {
    if (classes === void 0) {
      classes = '';
    }

    return "." + classes.trim().replace(/([\.:!\/])/g, '\\$1') // eslint-disable-line
    .replace(/ /g, '.');
  }

  function createElementIfNotDefined($container, params, createElements, checkProps) {
    var document = getDocument();

    if (createElements) {
      Object.keys(checkProps).forEach(function (key) {
        if (!params[key] && params.auto === true) {
          var element = document.createElement('div');
          element.className = checkProps[key];
          $container.append(element);
          params[key] = element;
        }
      });
    }

    return params;
  }

  var support;

  function calcSupport() {
    var window = getWindow();
    var document = getDocument();
    return {
      touch: !!('ontouchstart' in window || window.DocumentTouch && document instanceof
window.DocumentTouch),
      pointerEvents: !!window.PointerEvent && 'maxTouchPoints' in window.navigator &&
window.navigator.maxTouchPoints >= 0,
      observer: function checkObserver() {
        return 'MutationObserver' in window || 'WebkitMutationObserver' in window;
      }(),
      passiveListener: function checkPassiveListener() {
        var supportsPassive = false;

        try {
          var opts = Object.defineProperty({}, 'passive', {
            // eslint-disable-next-line
            get: function get() {
              supportsPassive = true;
            }
          });
          window.addEventListener('testPassiveListener', null, opts);
        } catch (e) {// No support
        }

        return supportsPassive;
      }(),
      gestures: function checkGestures() {
        return 'ongesturestart' in window;
      }()
    };
  }

  function getSupport() {
    if (!support) {
      support = calcSupport();
    }
```

```
      return support;
    }

    var device;

    function calcDevice(_temp) {
      var _ref = _temp === void 0 ? {} : _temp,
          userAgent = _ref.userAgent;

      var support = getSupport();
      var window = getWindow();
      var platform = window.navigator.platform;
      var ua = userAgent || window.navigator.userAgent;
      var device = {
        ios: false,
        android: false
      };
      var screenWidth = window.screen.width;
      var screenHeight = window.screen.height;
      var android = ua.match(/(Android);?[\s\/]+([\d.]+)?/); // eslint-disable-line

      var ipad = ua.match(/(iPad).*OS\s([\d_]+)/);
      var ipod = ua.match(/(iPod)(.*OS\s([\d_]+))?/);
      var iphone = !ipad && ua.match(/(iPhone\sOS|iOS)\s([\d_]+)/);
      var windows = platform === 'Win32';
      var macos = platform === 'MacIntel'; // iPadOs 13 fix

      var iPadScreens = ['1024x1366', '1366x1024', '834x1194', '1194x834', '834x1112', '1112x834',
  '768x1024', '1024x768', '820x1180', '1180x820', '810x1080', '1080x810'];

      if (!ipad && macos && support.touch && iPadScreens.indexOf(screenWidth + "x" + screenHeight) >=
  0) {
        ipad = ua.match(/(Version)\/([\d.]+)/);
        if (!ipad) ipad = [0, 1, '13_0_0'];
        macos = false;
      } // Android


      if (android && !windows) {
        device.os = 'android';
        device.android = true;
      }

      if (ipad || iphone || ipod) {
        device.os = 'ios';
        device.ios = true;
      } // Export object


      return device;
    }

    function getDevice(overrides) {
      if (overrides === void 0) {
        overrides = {};
      }

      if (!device) {
        device = calcDevice(overrides);
      }

      return device;
    }
    var browser;

    function calcBrowser() {
      var window = getWindow();

      function isSafari() {
        var ua = window.navigator.userAgent.toLowerCase();
        return ua.indexOf('safari') >= 0 && ua.indexOf('chrome') < 0 && ua.indexOf('android') < 0;
      }

      return {
        isEdge: !!window.navigator.userAgent.match(/Edge/g),
        isSafari: isSafari(),
        isWebView: /(iPhone|iPod|iPad).*AppleWebKit(?!.*Safari)/i.test(window.navigator.userAgent)
      };
    }

    function getBrowser() {
      if (!browser) {
        browser = calcBrowser();
      }

      return browser;
    }

    var supportsResizeObserver = function supportsResizeObserver() {
      var window = getWindow();
      return typeof window.ResizeObserver !== 'undefined';
    };

    var Resize = {
      name: 'resize',
      create: function create() {
        var swiper = this;
        extend(swiper, {
          resize: {
            observer: null,
            createObserver: function createObserver() {
              if (!swiper || swiper.destroyed || !swiper.initialized) return;
              swiper.resize.observer = new ResizeObserver(function (entries) {
                var width = swiper.width,
                    height = swiper.height;
                var newWidth = width;
                var newHeight = height;
                entries.forEach(function (_ref) {
                  var contentBoxSize = _ref.contentBoxSize,
                      contentRect = _ref.contentRect,
                      target = _ref.target;
                  if (target && target !== swiper.el) return;
                  newWidth = contentRect ? contentRect.width : (contentBoxSize[0] ||
  contentBoxSize).inlineSize;
                  newHeight = contentRect ? contentRect.height : (contentBoxSize[0] ||
  contentBoxSize).blockSize;
                });

                if (newWidth !== width || newHeight !== height) {
                  swiper.resize.resizeHandler();
                }
              });
              swiper.resize.observer.observe(swiper.el);
            },
            removeObserver: function removeObserver() {
              if (swiper.resize.observer && swiper.resize.observer.unobserve && swiper.el) {
                swiper.resize.observer.unobserve(swiper.el);
                swiper.resize.observer = null;
              }
            },
            resizeHandler: function resizeHandler() {
              if (!swiper || swiper.destroyed || !swiper.initialized) return;
              swiper.emit('beforeResize');
              swiper.emit('resize');
            },
            orientationChangeHandler: function orientationChangeHandler() {
              if (!swiper || swiper.destroyed || !swiper.initialized) return;
              swiper.emit('orientationchange');
            }
          }
        });
      },
      on: {
        init: function init(swiper) {
          var window = getWindow();

          if (swiper.params.resizeObserver && supportsResizeObserver()) {
            swiper.resize.createObserver();
            return;
          } // Emit resize


          window.addEventListener('resize', swiper.resize.resizeHandler); // Emit orientationchange

          window.addEventListener('orientationchange', swiper.resize.orientationChangeHandler);
        },
        destroy: function destroy(swiper) {
          var window = getWindow();
          swiper.resize.removeObserver();
          window.removeEventListener('resize', swiper.resize.resizeHandler);
          window.removeEventListener('orientationchange', swiper.resize.orientationChangeHandler);
        }
      }
    };

    var Observer = {
      attach: function attach(target, options) {
        if (options === void 0) {
          options = {};
        }

        var window = getWindow();
        var swiper = this;
        var ObserverFunc = window.MutationObserver || window.WebkitMutationObserver;
        var observer = new ObserverFunc(function (mutations) {
          // The observerUpdate event should only be triggered
          // once despite the number of mutations.  Additional
          // triggers are redundant and are very costly
          if (mutations.length === 1) {
            swiper.emit('observerUpdate', mutations[0]);
            return;
          }

          var observerUpdate = function observerUpdate() {
            swiper.emit('observerUpdate', mutations[0]);
          };

          if (window.requestAnimationFrame) {
            window.requestAnimationFrame(observerUpdate);
          } else {
            window.setTimeout(observerUpdate, 0);
          }
        });
```

```
      observer.observe(target, {
        attributes: typeof options.attributes === 'undefined' ? true : options.attributes,
        childList: typeof options.childList === 'undefined' ? true : options.childList,
        characterData: typeof options.characterData === 'undefined' ? true : options.characterData
      });
      swiper.observer.observers.push(observer);
    },
    init: function init() {
      var swiper = this;
      if (!swiper.support.observer || !swiper.params.observer) return;

      if (swiper.params.observeParents) {
        var containerParents = swiper.$el.parents();

        for (var i = 0; i < containerParents.length; i += 1) {
          swiper.observer.attach(containerParents[i]);
        }
      } // Observe container


      swiper.observer.attach(swiper.$el[0], {
        childList: swiper.params.observeSlideChildren
      }); // Observe wrapper

      swiper.observer.attach(swiper.$wrapperEl[0], {
        attributes: false
      });
    },
    destroy: function destroy() {
      var swiper = this;
      swiper.observer.observers.forEach(function (observer) {
        observer.disconnect();
      });
      swiper.observer.observers = [];
    }
  };
  var Observer$1 = {
    name: 'observer',
    params: {
      observer: false,
      observeParents: false,
      observeSlideChildren: false
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        observer: _extends({}, Observer, {
          observers: []
        })
      });
    },
    on: {
      init: function init(swiper) {
        swiper.observer.init();
      },
      destroy: function destroy(swiper) {
        swiper.observer.destroy();
      }
    }
  };

  var modular = {
    useParams: function useParams(instanceParams) {
      var instance = this;
      if (!instance.modules) return;
      Object.keys(instance.modules).forEach(function (moduleName) {
        var module = instance.modules[moduleName]; // Extend params

        if (module.params) {
          extend(instanceParams, module.params);
        }
      });
    },
    useModules: function useModules(modulesParams) {
      if (modulesParams === void 0) {
        modulesParams = {};
      }

      var instance = this;
      if (!instance.modules) return;
      Object.keys(instance.modules).forEach(function (moduleName) {
        var module = instance.modules[moduleName];
        var moduleParams = modulesParams[moduleName] || {}; // Add event listeners

        if (module.on && instance.on) {
          Object.keys(module.on).forEach(function (moduleEventName) {
            instance.on(moduleEventName, module.on[moduleEventName]);
          });
        } // Module create callback


        if (module.create) {
          module.create.bind(instance)(moduleParams);
        }
      });
    }
  };

  /* eslint-disable no-underscore-dangle */
  var eventsEmitter = {
    on: function on(events, handler, priority) {
      var self = this;
      if (typeof handler !== 'function') return self;
      var method = priority ? 'unshift' : 'push';
      events.split(' ').forEach(function (event) {
        if (!self.eventsListeners[event]) self.eventsListeners[event] = [];
        self.eventsListeners[event][method](handler);
      });
      return self;
    },
    once: function once(events, handler, priority) {
      var self = this;
      if (typeof handler !== 'function') return self;

      function onceHandler() {
        self.off(events, onceHandler);

        if (onceHandler.__emitterProxy) {
          delete onceHandler.__emitterProxy;
        }

        for (var _len = arguments.length, args = new Array(_len), _key = 0; _key < _len; _key++) {
          args[_key] = arguments[_key];
        }

        handler.apply(self, args);
      }

      onceHandler.__emitterProxy = handler;
      return self.on(events, onceHandler, priority);
    },
    onAny: function onAny(handler, priority) {
      var self = this;
      if (typeof handler !== 'function') return self;
      var method = priority ? 'unshift' : 'push';

      if (self.eventsAnyListeners.indexOf(handler) < 0) {
        self.eventsAnyListeners[method](handler);
      }

      return self;
    },
    offAny: function offAny(handler) {
      var self = this;
      if (!self.eventsAnyListeners) return self;
      var index = self.eventsAnyListeners.indexOf(handler);

      if (index >= 0) {
        self.eventsAnyListeners.splice(index, 1);
      }

      return self;
    },
    off: function off(events, handler) {
      var self = this;
      if (!self.eventsListeners) return self;
      events.split(' ').forEach(function (event) {
        if (typeof handler === 'undefined') {
          self.eventsListeners[event] = [];
        } else if (self.eventsListeners[event]) {
          self.eventsListeners[event].forEach(function (eventHandler, index) {
            if (eventHandler === handler || eventHandler.__emitterProxy &&
eventHandler.__emitterProxy === handler) {
              self.eventsListeners[event].splice(index, 1);
            }
          });
        }
      });
      return self;
    },
    emit: function emit() {
      var self = this;
      if (!self.eventsListeners) return self;
      var events;
      var data;
      var context;

      for (var _len2 = arguments.length, args = new Array(_len2), _key2 = 0; _key2 < _len2; _key2++)
{
        args[_key2] = arguments[_key2];
      }

      if (typeof args[0] === 'string' || Array.isArray(args[0])) {
        events = args[0];
        data = args.slice(1, args.length);
        context = self;
      } else {
        events = args[0].events;
        data = args[0].data;
        context = args[0].context || self;
      }

      data.unshift(context);
      var eventsArray = Array.isArray(events) ? events : events.split(' ');
      eventsArray.forEach(function (event) {
        if (self.eventsAnyListeners && self.eventsAnyListeners.length) {
          self.eventsAnyListeners.forEach(function (eventHandler) {
            eventHandler.apply(context, [event].concat(data));
          });
        }

        if (self.eventsListeners && self.eventsListeners[event]) {
          self.eventsListeners[event].forEach(function (eventHandler) {
            eventHandler.apply(context, data);
          });
        }
      });
      return self;
    }
  };

  function updateSize() {
    var swiper = this;
    var width;
    var height;
    var $el = swiper.$el;

    if (typeof swiper.params.width !== 'undefined' && swiper.params.width !== null) {
      width = swiper.params.width;
    } else {
      width = $el[0].clientWidth;
    }

    if (typeof swiper.params.height !== 'undefined' && swiper.params.height !== null) {
      height = swiper.params.height;
    } else {
      height = $el[0].clientHeight;
    }

    if (width === 0 && swiper.isHorizontal() || height === 0 && swiper.isVertical()) {
      return;
    } // Subtract paddings


    width = width - parseInt($el.css('padding-left') || 0, 10) - parseInt($el.css('padding-right') ||
0, 10);
    height = height - parseInt($el.css('padding-top') || 0, 10) - parseInt($el.css('padding-bottom')
|| 0, 10);
    if (Number.isNaN(width)) width = 0;
    if (Number.isNaN(height)) height = 0;
    extend(swiper, {
      width: width,
      height: height,
      size: swiper.isHorizontal() ? width : height
    });
  }

  function updateSlides() {
    var swiper = this;

    function getDirectionLabel(property) {
      if (swiper.isHorizontal()) {
        return property;
      } // prettier-ignore


      return {
        'width': 'height',
        'margin-top': 'margin-left',
        'margin-bottom ': 'margin-right',
        'margin-left': 'margin-top',
        'margin-right': 'margin-bottom',
        'padding-left': 'padding-top',
        'padding-right': 'padding-bottom',
        'marginRight': 'marginBottom'
      }[property];
    }

    function getDirectionPropertyValue(node, label) {
      return parseFloat(node.getPropertyValue(getDirectionLabel(label)) || 0);
    }

    var params = swiper.params;
    var $wrapperEl = swiper.$wrapperEl,
        swiperSize = swiper.size,
        rtl = swiper.rtlTranslate,
        wrongRTL = swiper.wrongRTL;
    var isVirtual = swiper.virtual && params.virtual.enabled;
    var previousSlidesLength = isVirtual ? swiper.virtual.slides.length : swiper.slides.length;
    var slides = $wrapperEl.children("." + swiper.params.slideClass);
    var slidesLength = isVirtual ? swiper.virtual.slides.length : slides.length;
    var snapGrid = [];
    var slidesGrid = [];
    var slidesSizesGrid = [];
    var offsetBefore = params.slidesOffsetBefore;

    if (typeof offsetBefore === 'function') {
      offsetBefore = params.slidesOffsetBefore.call(swiper);
    }

    var offsetAfter = params.slidesOffsetAfter;

    if (typeof offsetAfter === 'function') {
      offsetAfter = params.slidesOffsetAfter.call(swiper);
    }

    var previousSnapGridLength = swiper.snapGrid.length;
    var previousSlidesGridLength = swiper.slidesGrid.length;
    var spaceBetween = params.spaceBetween;
    var slidePosition = -offsetBefore;
    var prevSlideSize = 0;
    var index = 0;

    if (typeof swiperSize === 'undefined') {
      return;
    }

    if (typeof spaceBetween === 'string' && spaceBetween.indexOf('%') >= 0) {
      spaceBetween = parseFloat(spaceBetween.replace('%', '')) / 100 * swiperSize;
    }

    swiper.virtualSize = -spaceBetween; // reset margins

    if (rtl) slides.css({
      marginLeft: '',
      marginBottom: '',
      marginTop: ''
    });else slides.css({
      marginRight: '',
      marginBottom: '',
      marginTop: ''
    });
    var slidesNumberEvenToRows;

    if (params.slidesPerColumn > 1) {
      if (Math.floor(slidesLength / params.slidesPerColumn) === slidesLength /
swiper.params.slidesPerColumn) {
        slidesNumberEvenToRows = slidesLength;
      } else {
        slidesNumberEvenToRows = Math.ceil(slidesLength / params.slidesPerColumn) *
params.slidesPerColumn;
      }

      if (params.slidesPerView !== 'auto' && params.slidesPerColumnFill === 'row') {
        slidesNumberEvenToRows = Math.max(slidesNumberEvenToRows, params.slidesPerView *
params.slidesPerColumn);
      }
    } // Calc slides


    var slideSize;
    var slidesPerColumn = params.slidesPerColumn;
    var slidesPerRow = slidesNumberEvenToRows / slidesPerColumn;
    var numFullColumns = Math.floor(slidesLength / params.slidesPerColumn);

    for (var i = 0; i < slidesLength; i += 1) {
      slideSize = 0;
      var slide = slides.eq(i);

      if (params.slidesPerColumn > 1) {
        // Set slides order
        var newSlideOrderIndex = void 0;
        var column = void 0;
        var row = void 0;

        if (params.slidesPerColumnFill === 'row' && params.slidesPerGroup > 1) {
          var groupIndex = Math.floor(i / (params.slidesPerGroup * params.slidesPerColumn));
          var slideIndexInGroup = i - params.slidesPerColumn * params.slidesPerGroup * groupIndex;
          var columnsInGroup = groupIndex === 0 ? params.slidesPerGroup :
Math.min(Math.ceil((slidesLength - groupIndex * slidesPerColumn * params.slidesPerGroup) /
slidesPerColumn), params.slidesPerGroup);
          row = Math.floor(slideIndexInGroup / columnsInGroup);
          column = slideIndexInGroup - row * columnsInGroup + groupIndex * params.slidesPerGroup;
          newSlideOrderIndex = column + row * slidesNumberEvenToRows / slidesPerColumn;
          slide.css({
            '-webkit-box-ordinal-group': newSlideOrderIndex,
            '-moz-box-ordinal-group': newSlideOrderIndex,
            '-ms-flex-order': newSlideOrderIndex,
            '-webkit-order': newSlideOrderIndex,
            order: newSlideOrderIndex
          });
        } else if (params.slidesPerColumnFill === 'column') {
          column = Math.floor(i / slidesPerColumn);
          row = i - column * slidesPerColumn;

          if (column > numFullColumns || column === numFullColumns && row === slidesPerColumn - 1) {
            row += 1;

            if (row >= slidesPerColumn) {
              row = 0;
              column += 1;
            }
          }
        } else {
          row = Math.floor(i / slidesPerRow);
          column = i - row * slidesPerRow;
        }

        slide.css(getDirectionLabel('margin-top'), row !== 0 ? params.spaceBetween &&
params.spaceBetween + "px" : '');
```

```
      }

      if (slide.css('display') === 'none') continue; // eslint-disable-line

      if (params.slidesPerView === 'auto') {
        var slideStyles = getComputedStyle(slide[0]);
        var currentTransform = slide[0].style.transform;
        var currentWebKitTransform = slide[0].style.webkitTransform;

        if (currentTransform) {
          slide[0].style.transform = 'none';
        }

        if (currentWebKitTransform) {
          slide[0].style.webkitTransform = 'none';
        }

        if (params.roundLengths) {
          slideSize = swiper.isHorizontal() ? slide.outerWidth(true) : slide.outerHeight(true);
        } else {
          // eslint-disable-next-line
          var width = getDirectionPropertyValue(slideStyles, 'width');
          var paddingLeft = getDirectionPropertyValue(slideStyles, 'padding-left');
          var paddingRight = getDirectionPropertyValue(slideStyles, 'padding-right');
          var marginLeft = getDirectionPropertyValue(slideStyles, 'margin-left');
          var marginRight = getDirectionPropertyValue(slideStyles, 'margin-right');
          var boxSizing = slideStyles.getPropertyValue('box-sizing');

          if (boxSizing && boxSizing === 'border-box') {
            slideSize = width + marginLeft + marginRight;
          } else {
            var _slide$ = slide[0],
                clientWidth = _slide$.clientWidth,
                offsetWidth = _slide$.offsetWidth;
            slideSize = width + paddingLeft + paddingRight + marginLeft + marginRight + (offsetWidth
- clientWidth);
          }
        }

        if (currentTransform) {
          slide[0].style.transform = currentTransform;
        }

        if (currentWebKitTransform) {
          slide[0].style.webkitTransform = currentWebKitTransform;
        }

        if (params.roundLengths) slideSize = Math.floor(slideSize);
      } else {
        slideSize = (swiperSize - (params.slidesPerView - 1) * spaceBetween) / params.slidesPerView;
        if (params.roundLengths) slideSize = Math.floor(slideSize);

        if (slides[i]) {
          slides[i].style[getDirectionLabel('width')] = slideSize + "px";
        }
      }

      if (slides[i]) {
        slides[i].swiperSlideSize = slideSize;
      }

      slidesSizesGrid.push(slideSize);

      if (params.centeredSlides) {
        slidePosition = slidePosition + slideSize / 2 + prevSlideSize / 2 + spaceBetween;
        if (prevSlideSize === 0 && i !== 0) slidePosition = slidePosition - swiperSize / 2 -
spaceBetween;
        if (i === 0) slidePosition = slidePosition - swiperSize / 2 - spaceBetween;
        if (Math.abs(slidePosition) < 1 / 1000) slidePosition = 0;
        if (params.roundLengths) slidePosition = Math.floor(slidePosition);
        if (index % params.slidesPerGroup === 0) snapGrid.push(slidePosition);
        slidesGrid.push(slidePosition);
      } else {
        if (params.roundLengths) slidePosition = Math.floor(slidePosition);
        if ((index - Math.min(swiper.params.slidesPerGroupSkip, index)) %
swiper.params.slidesPerGroup === 0) snapGrid.push(slidePosition);
        slidesGrid.push(slidePosition);
        slidePosition = slidePosition + slideSize + spaceBetween;
      }

      swiper.virtualSize += slideSize + spaceBetween;
      prevSlideSize = slideSize;
      index += 1;
    }

    swiper.virtualSize = Math.max(swiper.virtualSize, swiperSize) + offsetAfter;
    var newSlidesGrid;

    if (rtl && wrongRTL && (params.effect === 'slide' || params.effect === 'coverflow')) {
      $wrapperEl.css({
        width: swiper.virtualSize + params.spaceBetween + "px"
      });
    }

    if (params.setWrapperSize) {
      var _$wrapperEl$css;

      $wrapperEl.css((_$wrapperEl$css = {}, _$wrapperEl$css[getDirectionLabel('width')] =
swiper.virtualSize + params.spaceBetween + "px", _$wrapperEl$css));
    }

    if (params.slidesPerColumn > 1) {
      var _$wrapperEl$css2;

      swiper.virtualSize = (slideSize + params.spaceBetween) * slidesNumberEvenToRows;
      swiper.virtualSize = Math.ceil(swiper.virtualSize / params.slidesPerColumn) -
params.spaceBetween;
      $wrapperEl.css((_$wrapperEl$css2 = {}, _$wrapperEl$css2[getDirectionLabel('width')] =
swiper.virtualSize + params.spaceBetween + "px", _$wrapperEl$css2));

      if (params.centeredSlides) {
        newSlidesGrid = [];

        for (var _i = 0; _i < snapGrid.length; _i += 1) {
          var slidesGridItem = snapGrid[_i];
          if (params.roundLengths) slidesGridItem = Math.floor(slidesGridItem);
          if (snapGrid[_i] < swiper.virtualSize + snapGrid[0]) newSlidesGrid.push(slidesGridItem);
        }

        snapGrid = newSlidesGrid;
      }
    } // Remove last grid elements depending on width


    if (!params.centeredSlides) {
      newSlidesGrid = [];

      for (var _i2 = 0; _i2 < snapGrid.length; _i2 += 1) {
        var _slidesGridItem = snapGrid[_i2];
        if (params.roundLengths) _slidesGridItem = Math.floor(_slidesGridItem);
```

```
          if (snapGrid[_i2] <= swiper.virtualSize - swiperSize) {
            newSlidesGrid.push(_slidesGridItem);
          }
        }

        snapGrid = newSlidesGrid;

        if (Math.floor(swiper.virtualSize - swiperSize) - Math.floor(snapGrid[snapGrid.length - 1]) >
  1) {
          snapGrid.push(swiper.virtualSize - swiperSize);
        }
      }

      if (snapGrid.length === 0) snapGrid = [0];

      if (params.spaceBetween !== 0) {
        var _slides$filter$css;

        var key = swiper.isHorizontal() && rtl ? 'marginLeft' : getDirectionLabel('marginRight');
        slides.filter(function (_, slideIndex) {
          if (!params.cssMode) return true;

          if (slideIndex === slides.length - 1) {
            return false;
          }

          return true;
        }).css((_slides$filter$css = {}, _slides$filter$css[key] = spaceBetween + "px",
  _slides$filter$css));
      }

      if (params.centeredSlides && params.centeredSlidesBounds) {
        var allSlidesSize = 0;
        slidesSizesGrid.forEach(function (slideSizeValue) {
          allSlidesSize += slideSizeValue + (params.spaceBetween ? params.spaceBetween : 0);
        });
        allSlidesSize -= params.spaceBetween;
        var maxSnap = allSlidesSize - swiperSize;
        snapGrid = snapGrid.map(function (snap) {
          if (snap < 0) return -offsetBefore;
          if (snap > maxSnap) return maxSnap + offsetAfter;
          return snap;
        });
      }

      if (params.centerInsufficientSlides) {
        var _allSlidesSize = 0;
        slidesSizesGrid.forEach(function (slideSizeValue) {
          _allSlidesSize += slideSizeValue + (params.spaceBetween ? params.spaceBetween : 0);
        });
        _allSlidesSize -= params.spaceBetween;

        if (_allSlidesSize < swiperSize) {
          var allSlidesOffset = (swiperSize - _allSlidesSize) / 2;
          snapGrid.forEach(function (snap, snapIndex) {
            snapGrid[snapIndex] = snap - allSlidesOffset;
          });
          slidesGrid.forEach(function (snap, snapIndex) {
            slidesGrid[snapIndex] = snap + allSlidesOffset;
          });
        }
      }

      extend(swiper, {
```

```
      slides: slides,
      snapGrid: snapGrid,
      slidesGrid: slidesGrid,
      slidesSizesGrid: slidesSizesGrid
    });

    if (slidesLength !== previousSlidesLength) {
      swiper.emit('slidesLengthChange');
    }

    if (snapGrid.length !== previousSnapGridLength) {
      if (swiper.params.watchOverflow) swiper.checkOverflow();
      swiper.emit('snapGridLengthChange');
    }

    if (slidesGrid.length !== previousSlidesGridLength) {
      swiper.emit('slidesGridLengthChange');
    }

    if (params.watchSlidesProgress || params.watchSlidesVisibility) {
      swiper.updateSlidesOffset();
    }
  }

  function updateAutoHeight(speed) {
    var swiper = this;
    var activeSlides = [];
    var isVirtual = swiper.virtual && swiper.params.virtual.enabled;
    var newHeight = 0;
    var i;

    if (typeof speed === 'number') {
      swiper.setTransition(speed);
    } else if (speed === true) {
      swiper.setTransition(swiper.params.speed);
    }

    var getSlideByIndex = function getSlideByIndex(index) {
      if (isVirtual) {
        return swiper.slides.filter(function (el) {
          return parseInt(el.getAttribute('data-swiper-slide-index'), 10) === index;
        })[0];
      }

      return swiper.slides.eq(index)[0];
    }; // Find slides currently in view


    if (swiper.params.slidesPerView !== 'auto' && swiper.params.slidesPerView > 1) {
      if (swiper.params.centeredSlides) {
        swiper.visibleSlides.each(function (slide) {
          activeSlides.push(slide);
        });
      } else {
        for (i = 0; i < Math.ceil(swiper.params.slidesPerView); i += 1) {
          var index = swiper.activeIndex + i;
          if (index > swiper.slides.length && !isVirtual) break;
          activeSlides.push(getSlideByIndex(index));
        }
      }
    } else {
      activeSlides.push(getSlideByIndex(swiper.activeIndex));
    } // Find new height from highest slide in view
```

```
      for (i = 0; i < activeSlides.length; i += 1) {
        if (typeof activeSlides[i] !== 'undefined') {
          var height = activeSlides[i].offsetHeight;
          newHeight = height > newHeight ? height : newHeight;
        }
      } // Update Height


      if (newHeight) swiper.$wrapperEl.css('height', newHeight + "px");
    }

    function updateSlidesOffset() {
      var swiper = this;
      var slides = swiper.slides;

      for (var i = 0; i < slides.length; i += 1) {
        slides[i].swiperSlideOffset = swiper.isHorizontal() ? slides[i].offsetLeft :
  slides[i].offsetTop;
      }
    }

    function updateSlidesProgress(translate) {
      if (translate === void 0) {
        translate = this && this.translate || 0;
      }

      var swiper = this;
      var params = swiper.params;
      var slides = swiper.slides,
          rtl = swiper.rtlTranslate;
      if (slides.length === 0) return;
      if (typeof slides[0].swiperSlideOffset === 'undefined') swiper.updateSlidesOffset();
      var offsetCenter = -translate;
      if (rtl) offsetCenter = translate; // Visible Slides

      slides.removeClass(params.slideVisibleClass);
      swiper.visibleSlidesIndexes = [];
      swiper.visibleSlides = [];

      for (var i = 0; i < slides.length; i += 1) {
        var slide = slides[i];
        var slideProgress = (offsetCenter + (params.centeredSlides ? swiper.minTranslate() : 0) -
  slide.swiperSlideOffset) / (slide.swiperSlideSize + params.spaceBetween);

        if (params.watchSlidesVisibility || params.centeredSlides && params.autoHeight) {
          var slideBefore = -(offsetCenter - slide.swiperSlideOffset);
          var slideAfter = slideBefore + swiper.slidesSizesGrid[i];
          var isVisible = slideBefore >= 0 && slideBefore < swiper.size - 1 || slideAfter > 1 &&
  slideAfter <= swiper.size || slideBefore <= 0 && slideAfter >= swiper.size;

          if (isVisible) {
            swiper.visibleSlides.push(slide);
            swiper.visibleSlidesIndexes.push(i);
            slides.eq(i).addClass(params.slideVisibleClass);
          }
        }

        slide.progress = rtl ? -slideProgress : slideProgress;
      }

      swiper.visibleSlides = $(swiper.visibleSlides);
    }

    function updateProgress(translate) {
      var swiper = this;
      if (typeof translate === 'undefined') {
        var multiplier = swiper.rtlTranslate ? -1 : 1; // eslint-disable-next-line

        translate = swiper && swiper.translate && swiper.translate * multiplier || 0;
      }

      var params = swiper.params;
      var translatesDiff = swiper.maxTranslate() - swiper.minTranslate();
      var progress = swiper.progress,
          isBeginning = swiper.isBeginning,
          isEnd = swiper.isEnd;
      var wasBeginning = isBeginning;
      var wasEnd = isEnd;

      if (translatesDiff === 0) {
        progress = 0;
        isBeginning = true;
        isEnd = true;
      } else {
        progress = (translate - swiper.minTranslate()) / translatesDiff;
        isBeginning = progress <= 0;
        isEnd = progress >= 1;
      }

      extend(swiper, {
        progress: progress,
        isBeginning: isBeginning,
        isEnd: isEnd
      });
      if (params.watchSlidesProgress || params.watchSlidesVisibility || params.centeredSlides &&
  params.autoHeight) swiper.updateSlidesProgress(translate);

      if (isBeginning && !wasBeginning) {
        swiper.emit('reachBeginning toEdge');
      }

      if (isEnd && !wasEnd) {
        swiper.emit('reachEnd toEdge');
      }

      if (wasBeginning && !isBeginning || wasEnd && !isEnd) {
        swiper.emit('fromEdge');
      }

      swiper.emit('progress', progress);
    }

    function updateSlidesClasses() {
      var swiper = this;
      var slides = swiper.slides,
          params = swiper.params,
          $wrapperEl = swiper.$wrapperEl,
          activeIndex = swiper.activeIndex,
          realIndex = swiper.realIndex;
      var isVirtual = swiper.virtual && params.virtual.enabled;
      slides.removeClass(params.slideActiveClass + " " + params.slideNextClass + " " +
  params.slidePrevClass + " " + params.slideDuplicateActiveClass + " " + params.slideDuplicateNextClass
  + " " + params.slideDuplicatePrevClass);
      var activeSlide;

      if (isVirtual) {
        activeSlide = swiper.$wrapperEl.find("." + params.slideClass + "[data-swiper-slide-index=\"" +
  activeIndex + "\"]");
      } else {
```

```
      activeSlide = slides.eq(activeIndex);
    } // Active classes


    activeSlide.addClass(params.slideActiveClass);

    if (params.loop) {
      // Duplicate to all looped slides
      if (activeSlide.hasClass(params.slideDuplicateClass)) {
        $wrapperEl.children("." + params.slideClass + ":not(." + params.slideDuplicateClass + ")
[data-swiper-slide-index=\"" + realIndex + "\"]").addClass(params.slideDuplicateActiveClass);
      } else {
        $wrapperEl.children("." + params.slideClass + "." + params.slideDuplicateClass + "[data-
swiper-slide-index=\"" + realIndex + "\"]").addClass(params.slideDuplicateActiveClass);
      }
    } // Next Slide


    var nextSlide = activeSlide.nextAll("." +
params.slideClass).eq(0).addClass(params.slideNextClass);

    if (params.loop && nextSlide.length === 0) {
      nextSlide = slides.eq(0);
      nextSlide.addClass(params.slideNextClass);
    } // Prev Slide


    var prevSlide = activeSlide.prevAll("." +
params.slideClass).eq(0).addClass(params.slidePrevClass);

    if (params.loop && prevSlide.length === 0) {
      prevSlide = slides.eq(-1);
      prevSlide.addClass(params.slidePrevClass);
    }

    if (params.loop) {
      // Duplicate to all looped slides
      if (nextSlide.hasClass(params.slideDuplicateClass)) {
        $wrapperEl.children("." + params.slideClass + ":not(." + params.slideDuplicateClass + ")
[data-swiper-slide-index=\"" + nextSlide.attr('data-swiper-slide-index') +
"\"]").addClass(params.slideDuplicateNextClass);
      } else {
        $wrapperEl.children("." + params.slideClass + "." + params.slideDuplicateClass + "[data-
swiper-slide-index=\"" + nextSlide.attr('data-swiper-slide-index') +
"\"]").addClass(params.slideDuplicateNextClass);
      }

      if (prevSlide.hasClass(params.slideDuplicateClass)) {
        $wrapperEl.children("." + params.slideClass + ":not(." + params.slideDuplicateClass + ")
[data-swiper-slide-index=\"" + prevSlide.attr('data-swiper-slide-index') +
"\"]").addClass(params.slideDuplicatePrevClass);
      } else {
        $wrapperEl.children("." + params.slideClass + "." + params.slideDuplicateClass + "[data-
swiper-slide-index=\"" + prevSlide.attr('data-swiper-slide-index') +
"\"]").addClass(params.slideDuplicatePrevClass);
      }
    }

    swiper.emitSlidesClasses();
  }

  function updateActiveIndex(newActiveIndex) {
    var swiper = this;
    var translate = swiper.rtlTranslate ? swiper.translate : -swiper.translate;
    var slidesGrid = swiper.slidesGrid,
        snapGrid = swiper.snapGrid,
        params = swiper.params,
        previousIndex = swiper.activeIndex,
        previousRealIndex = swiper.realIndex,
        previousSnapIndex = swiper.snapIndex;
    var activeIndex = newActiveIndex;
    var snapIndex;

    if (typeof activeIndex === 'undefined') {
      for (var i = 0; i < slidesGrid.length; i += 1) {
        if (typeof slidesGrid[i + 1] !== 'undefined') {
          if (translate >= slidesGrid[i] && translate < slidesGrid[i + 1] - (slidesGrid[i + 1] -
slidesGrid[i]) / 2) {
            activeIndex = i;
          } else if (translate >= slidesGrid[i] && translate < slidesGrid[i + 1]) {
            activeIndex = i + 1;
          }
        } else if (translate >= slidesGrid[i]) {
          activeIndex = i;
        }
      } // Normalize slideIndex


      if (params.normalizeSlideIndex) {
        if (activeIndex < 0 || typeof activeIndex === 'undefined') activeIndex = 0;
      }
    }

    if (snapGrid.indexOf(translate) >= 0) {
      snapIndex = snapGrid.indexOf(translate);
    } else {
      var skip = Math.min(params.slidesPerGroupSkip, activeIndex);
      snapIndex = skip + Math.floor((activeIndex - skip) / params.slidesPerGroup);
    }

    if (snapIndex >= snapGrid.length) snapIndex = snapGrid.length - 1;

    if (activeIndex === previousIndex) {
      if (snapIndex !== previousSnapIndex) {
        swiper.snapIndex = snapIndex;
        swiper.emit('snapIndexChange');
      }

      return;
    } // Get real index


    var realIndex = parseInt(swiper.slides.eq(activeIndex).attr('data-swiper-slide-index') ||
activeIndex, 10);
    extend(swiper, {
      snapIndex: snapIndex,
      realIndex: realIndex,
      previousIndex: previousIndex,
      activeIndex: activeIndex
    });
    swiper.emit('activeIndexChange');
    swiper.emit('snapIndexChange');

    if (previousRealIndex !== realIndex) {
      swiper.emit('realIndexChange');
    }

    if (swiper.initialized || swiper.params.runCallbacksOnInit) {
      swiper.emit('slideChange');
    }
```

```
  }

  function updateClickedSlide(e) {
    var swiper = this;
    var params = swiper.params;
    var slide = $(e.target).closest("." + params.slideClass)[0];
    var slideFound = false;
    var slideIndex;

    if (slide) {
      for (var i = 0; i < swiper.slides.length; i += 1) {
        if (swiper.slides[i] === slide) {
          slideFound = true;
          slideIndex = i;
          break;
        }
      }
    }

    if (slide && slideFound) {
      swiper.clickedSlide = slide;

      if (swiper.virtual && swiper.params.virtual.enabled) {
        swiper.clickedIndex = parseInt($(slide).attr('data-swiper-slide-index'), 10);
      } else {
        swiper.clickedIndex = slideIndex;
      }
    } else {
      swiper.clickedSlide = undefined;
      swiper.clickedIndex = undefined;
      return;
    }

    if (params.slideToClickedSlide && swiper.clickedIndex !== undefined && swiper.clickedIndex !==
swiper.activeIndex) {
      swiper.slideToClickedSlide();
    }
  }

  var update = {
    updateSize: updateSize,
    updateSlides: updateSlides,
    updateAutoHeight: updateAutoHeight,
    updateSlidesOffset: updateSlidesOffset,
    updateSlidesProgress: updateSlidesProgress,
    updateProgress: updateProgress,
    updateSlidesClasses: updateSlidesClasses,
    updateActiveIndex: updateActiveIndex,
    updateClickedSlide: updateClickedSlide
  };

  function getSwiperTranslate(axis) {
    if (axis === void 0) {
      axis = this.isHorizontal() ? 'x' : 'y';
    }

    var swiper = this;
    var params = swiper.params,
        rtl = swiper.rtlTranslate,
        translate = swiper.translate,
        $wrapperEl = swiper.$wrapperEl;

    if (params.virtualTranslate) {
      return rtl ? -translate : translate;
    }
```

```
      if (params.cssMode) {
        return translate;
      }

      var currentTranslate = getTranslate($wrapperEl[0], axis);
      if (rtl) currentTranslate = -currentTranslate;
      return currentTranslate || 0;
    }

    function setTranslate(translate, byController) {
      var swiper = this;
      var rtl = swiper.rtlTranslate,
          params = swiper.params,
          $wrapperEl = swiper.$wrapperEl,
          wrapperEl = swiper.wrapperEl,
          progress = swiper.progress;
      var x = 0;
      var y = 0;
      var z = 0;

      if (swiper.isHorizontal()) {
        x = rtl ? -translate : translate;
      } else {
        y = translate;
      }

      if (params.roundLengths) {
        x = Math.floor(x);
        y = Math.floor(y);
      }

      if (params.cssMode) {
        wrapperEl[swiper.isHorizontal() ? 'scrollLeft' : 'scrollTop'] = swiper.isHorizontal() ? -x : -
y;
      } else if (!params.virtualTranslate) {
        $wrapperEl.transform("translate3d(" + x + "px, " + y + "px, " + z + "px)");
      }

      swiper.previousTranslate = swiper.translate;
      swiper.translate = swiper.isHorizontal() ? x : y; // Check if we need to update progress

      var newProgress;
      var translatesDiff = swiper.maxTranslate() - swiper.minTranslate();

      if (translatesDiff === 0) {
        newProgress = 0;
      } else {
        newProgress = (translate - swiper.minTranslate()) / translatesDiff;
      }

      if (newProgress !== progress) {
        swiper.updateProgress(translate);
      }

      swiper.emit('setTranslate', swiper.translate, byController);
    }

    function minTranslate() {
      return -this.snapGrid[0];
    }

    function maxTranslate() {
      return -this.snapGrid[this.snapGrid.length - 1];
    }
    function translateTo(translate, speed, runCallbacks, translateBounds, internal) {
      if (translate === void 0) {
        translate = 0;
      }

      if (speed === void 0) {
        speed = this.params.speed;
      }

      if (runCallbacks === void 0) {
        runCallbacks = true;
      }

      if (translateBounds === void 0) {
        translateBounds = true;
      }

      var swiper = this;
      var params = swiper.params,
          wrapperEl = swiper.wrapperEl;

      if (swiper.animating && params.preventInteractionOnTransition) {
        return false;
      }

      var minTranslate = swiper.minTranslate();
      var maxTranslate = swiper.maxTranslate();
      var newTranslate;
      if (translateBounds && translate > minTranslate) newTranslate = minTranslate;else if
  (translateBounds && translate < maxTranslate) newTranslate = maxTranslate;else newTranslate =
  translate; // Update progress

      swiper.updateProgress(newTranslate);

      if (params.cssMode) {
        var isH = swiper.isHorizontal();

        if (speed === 0) {
          wrapperEl[isH ? 'scrollLeft' : 'scrollTop'] = -newTranslate;
        } else {
          // eslint-disable-next-line
          if (wrapperEl.scrollTo) {
            var _wrapperEl$scrollTo;

            wrapperEl.scrollTo((_wrapperEl$scrollTo = {}, _wrapperEl$scrollTo[isH ? 'left' : 'top'] = -
  newTranslate, _wrapperEl$scrollTo.behavior = 'smooth', _wrapperEl$scrollTo));
          } else {
            wrapperEl[isH ? 'scrollLeft' : 'scrollTop'] = -newTranslate;
          }
        }

        return true;
      }

      if (speed === 0) {
        swiper.setTransition(0);
        swiper.setTranslate(newTranslate);

        if (runCallbacks) {
          swiper.emit('beforeTransitionStart', speed, internal);
          swiper.emit('transitionEnd');
        }
      } else {
        swiper.setTransition(speed);
        swiper.setTranslate(newTranslate);

        if (runCallbacks) {
          swiper.emit('beforeTransitionStart', speed, internal);
          swiper.emit('transitionStart');
        }

        if (!swiper.animating) {
          swiper.animating = true;

          if (!swiper.onTranslateToWrapperTransitionEnd) {
            swiper.onTranslateToWrapperTransitionEnd = function transitionEnd(e) {
              if (!swiper || swiper.destroyed) return;
              if (e.target !== this) return;
              swiper.$wrapperEl[0].removeEventListener('transitionend',
  swiper.onTranslateToWrapperTransitionEnd);
              swiper.$wrapperEl[0].removeEventListener('webkitTransitionEnd',
  swiper.onTranslateToWrapperTransitionEnd);
              swiper.onTranslateToWrapperTransitionEnd = null;
              delete swiper.onTranslateToWrapperTransitionEnd;

              if (runCallbacks) {
                swiper.emit('transitionEnd');
              }
            };
          }

          swiper.$wrapperEl[0].addEventListener('transitionend',
  swiper.onTranslateToWrapperTransitionEnd);
          swiper.$wrapperEl[0].addEventListener('webkitTransitionEnd',
  swiper.onTranslateToWrapperTransitionEnd);
        }
      }

      return true;
    }

    var translate = {
      getTranslate: getSwiperTranslate,
      setTranslate: setTranslate,
      minTranslate: minTranslate,
      maxTranslate: maxTranslate,
      translateTo: translateTo
    };

    function setTransition(duration, byController) {
      var swiper = this;

      if (!swiper.params.cssMode) {
        swiper.$wrapperEl.transition(duration);
      }

      swiper.emit('setTransition', duration, byController);
    }

    function transitionStart(runCallbacks, direction) {
      if (runCallbacks === void 0) {
        runCallbacks = true;
      }

      var swiper = this;
      var activeIndex = swiper.activeIndex,
          params = swiper.params,
          previousIndex = swiper.previousIndex;
      if (params.cssMode) return;
```

```
      if (params.autoHeight) {
        swiper.updateAutoHeight();
      }

      var dir = direction;

      if (!dir) {
        if (activeIndex > previousIndex) dir = 'next';else if (activeIndex < previousIndex) dir =
  'prev';else dir = 'reset';
      }

      swiper.emit('transitionStart');

      if (runCallbacks && activeIndex !== previousIndex) {
        if (dir === 'reset') {
          swiper.emit('slideResetTransitionStart');
          return;
        }

        swiper.emit('slideChangeTransitionStart');

        if (dir === 'next') {
          swiper.emit('slideNextTransitionStart');
        } else {
          swiper.emit('slidePrevTransitionStart');
        }
      }
    }

    function transitionEnd(runCallbacks, direction) {
      if (runCallbacks === void 0) {
        runCallbacks = true;
      }

      var swiper = this;
      var activeIndex = swiper.activeIndex,
          previousIndex = swiper.previousIndex,
          params = swiper.params;
      swiper.animating = false;
      if (params.cssMode) return;
      swiper.setTransition(0);
      var dir = direction;

      if (!dir) {
        if (activeIndex > previousIndex) dir = 'next';else if (activeIndex < previousIndex) dir =
  'prev';else dir = 'reset';
      }

      swiper.emit('transitionEnd');

      if (runCallbacks && activeIndex !== previousIndex) {
        if (dir === 'reset') {
          swiper.emit('slideResetTransitionEnd');
          return;
        }

        swiper.emit('slideChangeTransitionEnd');

        if (dir === 'next') {
          swiper.emit('slideNextTransitionEnd');
        } else {
          swiper.emit('slidePrevTransitionEnd');
        }
      }
    }

    var transition = {
      setTransition: setTransition,
      transitionStart: transitionStart,
      transitionEnd: transitionEnd
    };

    function slideTo(index, speed, runCallbacks, internal, initial) {
      if (index === void 0) {
        index = 0;
      }

      if (speed === void 0) {
        speed = this.params.speed;
      }

      if (runCallbacks === void 0) {
        runCallbacks = true;
      }

      if (typeof index !== 'number' && typeof index !== 'string') {
        throw new Error("The 'index' argument cannot have type other than 'number' or 'string'. [" +
  typeof index + "] given.");
      }

      if (typeof index === 'string') {
        /**
         * The `index` argument converted from `string` to `number`.
         * @type {number}
         */
        var indexAsNumber = parseInt(index, 10);
        /**
         * Determines whether the `index` argument is a valid `number`
         * after being converted from the `string` type.
         * @type {boolean}
         */

        var isValidNumber = isFinite(indexAsNumber);

        if (!isValidNumber) {
          throw new Error("The passed-in 'index' (string) couldn't be converted to 'number'. [" + index
  + "] given.");
        } // Knowing that the converted `index` is a valid number,
        // we can update the original argument's value.


        index = indexAsNumber;
      }

      var swiper = this;
      var slideIndex = index;
      if (slideIndex < 0) slideIndex = 0;
      var params = swiper.params,
          snapGrid = swiper.snapGrid,
          slidesGrid = swiper.slidesGrid,
          previousIndex = swiper.previousIndex,
          activeIndex = swiper.activeIndex,
          rtl = swiper.rtlTranslate,
          wrapperEl = swiper.wrapperEl,
          enabled = swiper.enabled;

      if (swiper.animating && params.preventInteractionOnTransition || !enabled && !internal &&
  !initial) {
        return false;
      }

      var skip = Math.min(swiper.params.slidesPerGroupSkip, slideIndex);
      var snapIndex = skip + Math.floor((slideIndex - skip) / swiper.params.slidesPerGroup);
      if (snapIndex >= snapGrid.length) snapIndex = snapGrid.length - 1;

      if ((activeIndex || params.initialSlide || 0) === (previousIndex || 0) && runCallbacks) {
        swiper.emit('beforeSlideChangeStart');
      }

      var translate = -snapGrid[snapIndex]; // Update progress

      swiper.updateProgress(translate); // Normalize slideIndex

      if (params.normalizeSlideIndex) {
        for (var i = 0; i < slidesGrid.length; i += 1) {
          var normalizedTranslate = -Math.floor(translate * 100);
          var normalizedGird = Math.floor(slidesGrid[i] * 100);
          var normalizedGridNext = Math.floor(slidesGrid[i + 1] * 100);

          if (typeof slidesGrid[i + 1] !== 'undefined') {
            if (normalizedTranslate >= normalizedGird && normalizedTranslate < normalizedGridNext -
  (normalizedGridNext - normalizedGird) / 2) {
              slideIndex = i;
            } else if (normalizedTranslate >= normalizedGird && normalizedTranslate <
  normalizedGridNext) {
              slideIndex = i + 1;
            }
          } else if (normalizedTranslate >= normalizedGird) {
            slideIndex = i;
          }
        }
      } // Directions locks


      if (swiper.initialized && slideIndex !== activeIndex) {
        if (!swiper.allowSlideNext && translate < swiper.translate && translate <
  swiper.minTranslate()) {
          return false;
        }

        if (!swiper.allowSlidePrev && translate > swiper.translate && translate >
  swiper.maxTranslate()) {
          if ((activeIndex || 0) !== slideIndex) return false;
        }
      }

      var direction;
      if (slideIndex > activeIndex) direction = 'next';else if (slideIndex < activeIndex) direction =
  'prev';else direction = 'reset'; // Update Index

      if (rtl && -translate === swiper.translate || !rtl && translate === swiper.translate) {
        swiper.updateActiveIndex(slideIndex); // Update Height

        if (params.autoHeight) {
          swiper.updateAutoHeight();
        }

        swiper.updateSlidesClasses();

        if (params.effect !== 'slide') {
          swiper.setTranslate(translate);
        }

        if (direction !== 'reset') {
          swiper.transitionStart(runCallbacks, direction);
          swiper.transitionEnd(runCallbacks, direction);
        }

        return false;
      }

      if (params.cssMode) {
        var isH = swiper.isHorizontal();
        var t = -translate;

        if (rtl) {
          t = wrapperEl.scrollWidth - wrapperEl.offsetWidth - t;
        }

        if (speed === 0) {
          wrapperEl[isH ? 'scrollLeft' : 'scrollTop'] = t;
        } else {
          // eslint-disable-next-line
          if (wrapperEl.scrollTo) {
            var _wrapperEl$scrollTo;

            wrapperEl.scrollTo((_wrapperEl$scrollTo = {}, _wrapperEl$scrollTo[isH ? 'left' : 'top'] =
  t, _wrapperEl$scrollTo.behavior = 'smooth', _wrapperEl$scrollTo));
          } else {
            wrapperEl[isH ? 'scrollLeft' : 'scrollTop'] = t;
          }
        }

        return true;
      }

      if (speed === 0) {
        swiper.setTransition(0);
        swiper.setTranslate(translate);
        swiper.updateActiveIndex(slideIndex);
        swiper.updateSlidesClasses();
        swiper.emit('beforeTransitionStart', speed, internal);
        swiper.transitionStart(runCallbacks, direction);
        swiper.transitionEnd(runCallbacks, direction);
      } else {
        swiper.setTransition(speed);
        swiper.setTranslate(translate);
        swiper.updateActiveIndex(slideIndex);
        swiper.updateSlidesClasses();
        swiper.emit('beforeTransitionStart', speed, internal);
        swiper.transitionStart(runCallbacks, direction);

        if (!swiper.animating) {
          swiper.animating = true;

          if (!swiper.onSlideToWrapperTransitionEnd) {
            swiper.onSlideToWrapperTransitionEnd = function transitionEnd(e) {
              if (!swiper || swiper.destroyed) return;
              if (e.target !== this) return;
              swiper.$wrapperEl[0].removeEventListener('transitionend',
  swiper.onSlideToWrapperTransitionEnd);
              swiper.$wrapperEl[0].removeEventListener('webkitTransitionEnd',
  swiper.onSlideToWrapperTransitionEnd);
              swiper.onSlideToWrapperTransitionEnd = null;
              delete swiper.onSlideToWrapperTransitionEnd;
              swiper.transitionEnd(runCallbacks, direction);
            };
          }
          swiper.$wrapperEl[0].addEventListener('transitionend', swiper.onSlideToWrapperTransitionEnd);
          swiper.$wrapperEl[0].addEventListener('webkitTransitionEnd',
  swiper.onSlideToWrapperTransitionEnd);
        }
      }

      return true;
    }

    function slideToLoop(index, speed, runCallbacks, internal) {
      if (index === void 0) {
        index = 0;
      }

      if (speed === void 0) {
        speed = this.params.speed;
      }

      if (runCallbacks === void 0) {
        runCallbacks = true;
      }

      var swiper = this;
      var newIndex = index;

      if (swiper.params.loop) {
        newIndex += swiper.loopedSlides;
      }

      return swiper.slideTo(newIndex, speed, runCallbacks, internal);
    }

    /* eslint no-unused-vars: "off" */
    function slideNext(speed, runCallbacks, internal) {
      if (speed === void 0) {
        speed = this.params.speed;
      }

      if (runCallbacks === void 0) {
        runCallbacks = true;
      }

      var swiper = this;
      var params = swiper.params,
          animating = swiper.animating,
          enabled = swiper.enabled;
      if (!enabled) return swiper;
      var increment = swiper.activeIndex < params.slidesPerGroupSkip ? 1 : params.slidesPerGroup;

      if (params.loop) {
        if (animating && params.loopPreventsSlide) return false;
        swiper.loopFix(); // eslint-disable-next-line

        swiper._clientLeft = swiper.$wrapperEl[0].clientLeft;
      }

      return swiper.slideTo(swiper.activeIndex + increment, speed, runCallbacks, internal);
    }

    /* eslint no-unused-vars: "off" */
    function slidePrev(speed, runCallbacks, internal) {
      if (speed === void 0) {
        speed = this.params.speed;
      }
```

```
      if (runCallbacks === void 0) {
        runCallbacks = true;
      }

      var swiper = this;
      var params = swiper.params,
          animating = swiper.animating,
          snapGrid = swiper.snapGrid,
          slidesGrid = swiper.slidesGrid,
          rtlTranslate = swiper.rtlTranslate,
          enabled = swiper.enabled;
      if (!enabled) return swiper;

      if (params.loop) {
        if (animating && params.loopPreventsSlide) return false;
        swiper.loopFix(); // eslint-disable-next-line

        swiper._clientLeft = swiper.$wrapperEl[0].clientLeft;
      }

      var translate = rtlTranslate ? swiper.translate : -swiper.translate;

      function normalize(val) {
        if (val < 0) return -Math.floor(Math.abs(val));
        return Math.floor(val);
      }

      var normalizedTranslate = normalize(translate);
      var normalizedSnapGrid = snapGrid.map(function (val) {
        return normalize(val);
      });
      var prevSnap = snapGrid[normalizedSnapGrid.indexOf(normalizedTranslate) - 1];

      if (typeof prevSnap === 'undefined' && params.cssMode) {
        snapGrid.forEach(function (snap) {
          if (!prevSnap && normalizedTranslate >= snap) prevSnap = snap;
        });
      }

      var prevIndex;

      if (typeof prevSnap !== 'undefined') {
        prevIndex = slidesGrid.indexOf(prevSnap);
        if (prevIndex < 0) prevIndex = swiper.activeIndex - 1;
      }

      return swiper.slideTo(prevIndex, speed, runCallbacks, internal);
    }

    /* eslint no-unused-vars: "off" */
    function slideReset(speed, runCallbacks, internal) {
      if (speed === void 0) {
        speed = this.params.speed;
      }

      if (runCallbacks === void 0) {
        runCallbacks = true;
      }

      var swiper = this;
      return swiper.slideTo(swiper.activeIndex, speed, runCallbacks, internal);
    }

    /* eslint no-unused-vars: "off" */
    function slideToClosest(speed, runCallbacks, internal, threshold) {
```

```
    if (speed === void 0) {
      speed = this.params.speed;
    }

    if (runCallbacks === void 0) {
      runCallbacks = true;
    }

    if (threshold === void 0) {
      threshold = 0.5;
    }

    var swiper = this;
    var index = swiper.activeIndex;
    var skip = Math.min(swiper.params.slidesPerGroupSkip, index);
    var snapIndex = skip + Math.floor((index - skip) / swiper.params.slidesPerGroup);
    var translate = swiper.rtlTranslate ? swiper.translate : -swiper.translate;

    if (translate >= swiper.snapGrid[snapIndex]) {
      // The current translate is on or after the current snap index, so the choice
      // is between the current index and the one after it.
      var currentSnap = swiper.snapGrid[snapIndex];
      var nextSnap = swiper.snapGrid[snapIndex + 1];

      if (translate - currentSnap > (nextSnap - currentSnap) * threshold) {
        index += swiper.params.slidesPerGroup;
      }
    } else {
      // The current translate is before the current snap index, so the choice
      // is between the current index and the one before it.
      var prevSnap = swiper.snapGrid[snapIndex - 1];
      var _currentSnap = swiper.snapGrid[snapIndex];

      if (translate - prevSnap <= (_currentSnap - prevSnap) * threshold) {
        index -= swiper.params.slidesPerGroup;
      }
    }

    index = Math.max(index, 0);
    index = Math.min(index, swiper.slidesGrid.length - 1);
    return swiper.slideTo(index, speed, runCallbacks, internal);
  }

  function slideToClickedSlide() {
    var swiper = this;
    var params = swiper.params,
        $wrapperEl = swiper.$wrapperEl;
    var slidesPerView = params.slidesPerView === 'auto' ? swiper.slidesPerViewDynamic() :
params.slidesPerView;
    var slideToIndex = swiper.clickedIndex;
    var realIndex;

    if (params.loop) {
      if (swiper.animating) return;
      realIndex = parseInt($(swiper.clickedSlide).attr('data-swiper-slide-index'), 10);

      if (params.centeredSlides) {
        if (slideToIndex < swiper.loopedSlides - slidesPerView / 2 || slideToIndex >
swiper.slides.length - swiper.loopedSlides + slidesPerView / 2) {
          swiper.loopFix();
          slideToIndex = $wrapperEl.children("." + params.slideClass + "[data-swiper-slide-index=\""
+ realIndex + "\"]:not(." + params.slideDuplicateClass + ")").eq(0).index();
          nextTick(function () {
            swiper.slideTo(slideToIndex);
          });
        } else {
          swiper.slideTo(slideToIndex);
        }
      } else if (slideToIndex > swiper.slides.length - slidesPerView) {
        swiper.loopFix();
        slideToIndex = $wrapperEl.children("." + params.slideClass + "[data-swiper-slide-index=\"" +
realIndex + "\"]:not(." + params.slideDuplicateClass + ")").eq(0).index();
        nextTick(function () {
          swiper.slideTo(slideToIndex);
        });
      } else {
        swiper.slideTo(slideToIndex);
      }
    } else {
      swiper.slideTo(slideToIndex);
    }
  }

  var slide = {
    slideTo: slideTo,
    slideToLoop: slideToLoop,
    slideNext: slideNext,
    slidePrev: slidePrev,
    slideReset: slideReset,
    slideToClosest: slideToClosest,
    slideToClickedSlide: slideToClickedSlide
  };

  function loopCreate() {
    var swiper = this;
    var document = getDocument();
    var params = swiper.params,
        $wrapperEl = swiper.$wrapperEl; // Remove duplicated slides

    $wrapperEl.children("." + params.slideClass + "." + params.slideDuplicateClass).remove();
    var slides = $wrapperEl.children("." + params.slideClass);

    if (params.loopFillGroupWithBlank) {
      var blankSlidesNum = params.slidesPerGroup - slides.length % params.slidesPerGroup;

      if (blankSlidesNum !== params.slidesPerGroup) {
        for (var i = 0; i < blankSlidesNum; i += 1) {
          var blankNode = $(document.createElement('div')).addClass(params.slideClass + " " +
params.slideBlankClass);
          $wrapperEl.append(blankNode);
        }

        slides = $wrapperEl.children("." + params.slideClass);
      }
    }

    if (params.slidesPerView === 'auto' && !params.loopedSlides) params.loopedSlides = slides.length;
    swiper.loopedSlides = Math.ceil(parseFloat(params.loopedSlides || params.slidesPerView, 10));
    swiper.loopedSlides += params.loopAdditionalSlides;

    if (swiper.loopedSlides > slides.length) {
      swiper.loopedSlides = slides.length;
    }

    var prependSlides = [];
    var appendSlides = [];
    slides.each(function (el, index) {
      var slide = $(el);

      if (index < swiper.loopedSlides) {
        appendSlides.push(el);
      }

      if (index < slides.length && index >= slides.length - swiper.loopedSlides) {
        prependSlides.push(el);
      }

      slide.attr('data-swiper-slide-index', index);
    });

    for (var _i = 0; _i < appendSlides.length; _i += 1) {
      $wrapperEl.append($(appendSlides[_i].cloneNode(true)).addClass(params.slideDuplicateClass));
    }

    for (var _i2 = prependSlides.length - 1; _i2 >= 0; _i2 -= 1) {
      $wrapperEl.prepend($(prependSlides[_i2].cloneNode(true)).addClass(params.slideDuplicateClass));
    }
  }

  function loopFix() {
    var swiper = this;
    swiper.emit('beforeLoopFix');
    var activeIndex = swiper.activeIndex,
        slides = swiper.slides,
        loopedSlides = swiper.loopedSlides,
        allowSlidePrev = swiper.allowSlidePrev,
        allowSlideNext = swiper.allowSlideNext,
        snapGrid = swiper.snapGrid,
        rtl = swiper.rtlTranslate;
    var newIndex;
    swiper.allowSlidePrev = true;
    swiper.allowSlideNext = true;
    var snapTranslate = -snapGrid[activeIndex];
    var diff = snapTranslate - swiper.getTranslate(); // Fix For Negative Oversliding

    if (activeIndex < loopedSlides) {
      newIndex = slides.length - loopedSlides * 3 + activeIndex;
      newIndex += loopedSlides;
      var slideChanged = swiper.slideTo(newIndex, 0, false, true);

      if (slideChanged && diff !== 0) {
        swiper.setTranslate((rtl ? -swiper.translate : swiper.translate) - diff);
      }
    } else if (activeIndex >= slides.length - loopedSlides) {
      // Fix For Positive Oversliding
      newIndex = -slides.length + activeIndex + loopedSlides;
      newIndex += loopedSlides;

      var _slideChanged = swiper.slideTo(newIndex, 0, false, true);

      if (_slideChanged && diff !== 0) {
        swiper.setTranslate((rtl ? -swiper.translate : swiper.translate) - diff);
      }
    }

    swiper.allowSlidePrev = allowSlidePrev;
    swiper.allowSlideNext = allowSlideNext;
    swiper.emit('loopFix');
  }

  function loopDestroy() {
    var swiper = this;
    var $wrapperEl = swiper.$wrapperEl,
        params = swiper.params,
        slides = swiper.slides;
    $wrapperEl.children("." + params.slideClass + "." + params.slideDuplicateClass + ",." +
params.slideClass + "." + params.slideBlankClass).remove();
    slides.removeAttr('data-swiper-slide-index');
  }

  var loop = {
    loopCreate: loopCreate,
    loopFix: loopFix,
    loopDestroy: loopDestroy
  };

  function setGrabCursor(moving) {
    var swiper = this;
    if (swiper.support.touch || !swiper.params.simulateTouch || swiper.params.watchOverflow &&
swiper.isLocked || swiper.params.cssMode) return;
    var el = swiper.el;
    el.style.cursor = 'move';
    el.style.cursor = moving ? '-webkit-grabbing' : '-webkit-grab';
    el.style.cursor = moving ? '-moz-grabbin' : '-moz-grab';
    el.style.cursor = moving ? 'grabbing' : 'grab';
  }

  function unsetGrabCursor() {
    var swiper = this;

    if (swiper.support.touch || swiper.params.watchOverflow && swiper.isLocked ||
swiper.params.cssMode) {
      return;
    }

    swiper.el.style.cursor = '';
  }

  var grabCursor = {
    setGrabCursor: setGrabCursor,
    unsetGrabCursor: unsetGrabCursor
  };

  function appendSlide(slides) {
    var swiper = this;
    var $wrapperEl = swiper.$wrapperEl,
        params = swiper.params;

    if (params.loop) {
      swiper.loopDestroy();
    }

    if (typeof slides === 'object' && 'length' in slides) {
      for (var i = 0; i < slides.length; i += 1) {
        if (slides[i]) $wrapperEl.append(slides[i]);
      }
    } else {
      $wrapperEl.append(slides);
    }

    if (params.loop) {
      swiper.loopCreate();
    }

    if (!(params.observer && swiper.support.observer)) {
      swiper.update();
    }
  }

  function prependSlide(slides) {
    var swiper = this;
    var params = swiper.params,
        $wrapperEl = swiper.$wrapperEl,
        activeIndex = swiper.activeIndex;

    if (params.loop) {
      swiper.loopDestroy();
    }

    var newActiveIndex = activeIndex + 1;

    if (typeof slides === 'object' && 'length' in slides) {
      for (var i = 0; i < slides.length; i += 1) {
        if (slides[i]) $wrapperEl.prepend(slides[i]);
      }

      newActiveIndex = activeIndex + slides.length;
    } else {
      $wrapperEl.prepend(slides);
    }

    if (params.loop) {
      swiper.loopCreate();
    }

    if (!(params.observer && swiper.support.observer)) {
      swiper.update();
    }

    swiper.slideTo(newActiveIndex, 0, false);
  }

  function addSlide(index, slides) {
    var swiper = this;
    var $wrapperEl = swiper.$wrapperEl,
        params = swiper.params,
        activeIndex = swiper.activeIndex;
    var activeIndexBuffer = activeIndex;

    if (params.loop) {
      activeIndexBuffer -= swiper.loopedSlides;
      swiper.loopDestroy();
      swiper.slides = $wrapperEl.children("." + params.slideClass);
    }

    var baseLength = swiper.slides.length;

    if (index <= 0) {
      swiper.prependSlide(slides);
      return;
    }

    if (index >= baseLength) {
      swiper.appendSlide(slides);
      return;
    }

    var newActiveIndex = activeIndexBuffer > index ? activeIndexBuffer + 1 : activeIndexBuffer;
    var slidesBuffer = [];

    for (var i = baseLength - 1; i >= index; i -= 1) {
      var currentSlide = swiper.slides.eq(i);
      currentSlide.remove();
      slidesBuffer.unshift(currentSlide);
    }
    if (typeof slides === 'object' && 'length' in slides) {
      for (var _i = 0; _i < slides.length; _i += 1) {
        if (slides[_i]) $wrapperEl.append(slides[_i]);
      }

      newActiveIndex = activeIndexBuffer > index ? activeIndexBuffer + slides.length :
activeIndexBuffer;
    } else {
      $wrapperEl.append(slides);
    }

    for (var _i2 = 0; _i2 < slidesBuffer.length; _i2 += 1) {
      $wrapperEl.append(slidesBuffer[_i2]);
    }

    if (params.loop) {
      swiper.loopCreate();
    }

    if (!(params.observer && swiper.support.observer)) {
      swiper.update();
    }

    if (params.loop) {
      swiper.slideTo(newActiveIndex + swiper.loopedSlides, 0, false);
    } else {
      swiper.slideTo(newActiveIndex, 0, false);
    }
  }

  function removeSlide(slidesIndexes) {
    var swiper = this;
    var params = swiper.params,
        $wrapperEl = swiper.$wrapperEl,
        activeIndex = swiper.activeIndex;
    var activeIndexBuffer = activeIndex;

    if (params.loop) {
      activeIndexBuffer -= swiper.loopedSlides;
      swiper.loopDestroy();
      swiper.slides = $wrapperEl.children("." + params.slideClass);
    }

    var newActiveIndex = activeIndexBuffer;
    var indexToRemove;

    if (typeof slidesIndexes === 'object' && 'length' in slidesIndexes) {
      for (var i = 0; i < slidesIndexes.length; i += 1) {
        indexToRemove = slidesIndexes[i];
        if (swiper.slides[indexToRemove]) swiper.slides.eq(indexToRemove).remove();
        if (indexToRemove < newActiveIndex) newActiveIndex -= 1;
      }

      newActiveIndex = Math.max(newActiveIndex, 0);
    } else {
      indexToRemove = slidesIndexes;
      if (swiper.slides[indexToRemove]) swiper.slides.eq(indexToRemove).remove();
      if (indexToRemove < newActiveIndex) newActiveIndex -= 1;
      newActiveIndex = Math.max(newActiveIndex, 0);
    }

    if (params.loop) {
      swiper.loopCreate();
    }
    if (!(params.observer && swiper.support.observer)) {
      swiper.update();
    }

    if (params.loop) {
      swiper.slideTo(newActiveIndex + swiper.loopedSlides, 0, false);
    } else {
      swiper.slideTo(newActiveIndex, 0, false);
    }
  }

  function removeAllSlides() {
    var swiper = this;
    var slidesIndexes = [];

    for (var i = 0; i < swiper.slides.length; i += 1) {
      slidesIndexes.push(i);
    }

    swiper.removeSlide(slidesIndexes);
  }

  var manipulation = {
    appendSlide: appendSlide,
    prependSlide: prependSlide,
    addSlide: addSlide,
    removeSlide: removeSlide,
    removeAllSlides: removeAllSlides
  };

  function closestElement(selector, base) {
    if (base === void 0) {
      base = this;
    }

    function __closestFrom(el) {
      if (!el || el === getDocument() || el === getWindow()) return null;
      if (el.assignedSlot) el = el.assignedSlot;
      var found = el.closest(selector);
      return found || __closestFrom(el.getRootNode().host);
    }

    return __closestFrom(base);
  }

  function onTouchStart(event) {
    var swiper = this;
    var document = getDocument();
    var window = getWindow();
    var data = swiper.touchEventsData;
    var params = swiper.params,
        touches = swiper.touches,
        enabled = swiper.enabled;
    if (!enabled) return;

    if (swiper.animating && params.preventInteractionOnTransition) {
      return;
    }

    var e = event;
    if (e.originalEvent) e = e.originalEvent;
    var $targetEl = $(e.target);

    if (params.touchEventsTarget === 'wrapper') {
```

```
      if (!$targetEl.closest(swiper.wrapperEl).length) return;
    }

    data.isTouchEvent = e.type === 'touchstart';
    if (!data.isTouchEvent && 'which' in e && e.which === 3) return;
    if (!data.isTouchEvent && 'button' in e && e.button > 0) return;
    if (data.isTouched && data.isMoved) return; // change target el for shadow root component

    var swipingClassHasValue = !!params.noSwipingClass && params.noSwipingClass !== '';

    if (swipingClassHasValue && e.target && e.target.shadowRoot && event.path && event.path[0]) {
      $targetEl = $(event.path[0]);
    }

    var noSwipingSelector = params.noSwipingSelector ? params.noSwipingSelector : "." +
params.noSwipingClass;
    var isTargetShadow = !!(e.target && e.target.shadowRoot); // use closestElement for shadow root
element to get the actual closest for nested shadow root element

    if (params.noSwiping && (isTargetShadow ? closestElement(noSwipingSelector, e.target) :
$targetEl.closest(noSwipingSelector)[0])) {
      swiper.allowClick = true;
      return;
    }

    if (params.swipeHandler) {
      if (!$targetEl.closest(params.swipeHandler)[0]) return;
    }

    touches.currentX = e.type === 'touchstart' ? e.targetTouches[0].pageX : e.pageX;
    touches.currentY = e.type === 'touchstart' ? e.targetTouches[0].pageY : e.pageY;
    var startX = touches.currentX;
    var startY = touches.currentY; // Do NOT start if iOS edge swipe is detected. Otherwise iOS app
cannot swipe-to-go-back anymore

    var edgeSwipeDetection = params.edgeSwipeDetection || params.iOSEdgeSwipeDetection;
    var edgeSwipeThreshold = params.edgeSwipeThreshold || params.iOSEdgeSwipeThreshold;

    if (edgeSwipeDetection && (startX <= edgeSwipeThreshold || startX >= window.innerWidth -
edgeSwipeThreshold)) {
      if (edgeSwipeDetection === 'prevent') {
        event.preventDefault();
      } else {
        return;
      }
    }

    extend(data, {
      isTouched: true,
      isMoved: false,
      allowTouchCallbacks: true,
      isScrolling: undefined,
      startMoving: undefined
    });
    touches.startX = startX;
    touches.startY = startY;
    data.touchStartTime = now();
    swiper.allowClick = true;
    swiper.updateSize();
    swiper.swipeDirection = undefined;
    if (params.threshold > 0) data.allowThresholdMove = false;

    if (e.type !== 'touchstart') {
      var preventDefault = true;
      if ($targetEl.is(data.focusableElements)) preventDefault = false;
      if (document.activeElement && $(document.activeElement).is(data.focusableElements) &&
document.activeElement !== $targetEl[0]) {
        document.activeElement.blur();
      }

      var shouldPreventDefault = preventDefault && swiper.allowTouchMove &&
params.touchStartPreventDefault;

      if ((params.touchStartForcePreventDefault || shouldPreventDefault) &&
!$targetEl[0].isContentEditable) {
        e.preventDefault();
      }
    }

    swiper.emit('touchStart', e);
  }

  function onTouchMove(event) {
    var document = getDocument();
    var swiper = this;
    var data = swiper.touchEventsData;
    var params = swiper.params,
        touches = swiper.touches,
        rtl = swiper.rtlTranslate,
        enabled = swiper.enabled;
    if (!enabled) return;
    var e = event;
    if (e.originalEvent) e = e.originalEvent;

    if (!data.isTouched) {
      if (data.startMoving && data.isScrolling) {
        swiper.emit('touchMoveOpposite', e);
      }

      return;
    }

    if (data.isTouchEvent && e.type !== 'touchmove') return;
    var targetTouch = e.type === 'touchmove' && e.targetTouches && (e.targetTouches[0] ||
e.changedTouches[0]);
    var pageX = e.type === 'touchmove' ? targetTouch.pageX : e.pageX;
    var pageY = e.type === 'touchmove' ? targetTouch.pageY : e.pageY;

    if (e.preventedByNestedSwiper) {
      touches.startX = pageX;
      touches.startY = pageY;
      return;
    }

    if (!swiper.allowTouchMove) {
      // isMoved = true;
      swiper.allowClick = false;

      if (data.isTouched) {
        extend(touches, {
          startX: pageX,
          startY: pageY,
          currentX: pageX,
          currentY: pageY
        });
        data.touchStartTime = now();
      }

      return;
    }

    if (data.isTouchEvent && params.touchReleaseOnEdges && !params.loop) {
      if (swiper.isVertical()) {
        // Vertical
        if (pageY < touches.startY && swiper.translate <= swiper.maxTranslate() || pageY >
touches.startY && swiper.translate >= swiper.minTranslate()) {
          data.isTouched = false;
          data.isMoved = false;
          return;
        }
      } else if (pageX < touches.startX && swiper.translate <= swiper.maxTranslate() || pageX >
touches.startX && swiper.translate >= swiper.minTranslate()) {
        return;
      }
    }

    if (data.isTouchEvent && document.activeElement) {
      if (e.target === document.activeElement && $(e.target).is(data.focusableElements)) {
        data.isMoved = true;
        swiper.allowClick = false;
        return;
      }
    }

    if (data.allowTouchCallbacks) {
      swiper.emit('touchMove', e);
    }

    if (e.targetTouches && e.targetTouches.length > 1) return;
    touches.currentX = pageX;
    touches.currentY = pageY;
    var diffX = touches.currentX - touches.startX;
    var diffY = touches.currentY - touches.startY;
    if (swiper.params.threshold && Math.sqrt(Math.pow(diffX, 2) + Math.pow(diffY, 2)) <
swiper.params.threshold) return;

    if (typeof data.isScrolling === 'undefined') {
      var touchAngle;

      if (swiper.isHorizontal() && touches.currentY === touches.startY || swiper.isVertical() &&
touches.currentX === touches.startX) {
        data.isScrolling = false;
      } else {
        // eslint-disable-next-line
        if (diffX * diffX + diffY * diffY >= 25) {
          touchAngle = Math.atan2(Math.abs(diffY), Math.abs(diffX)) * 180 / Math.PI;
          data.isScrolling = swiper.isHorizontal() ? touchAngle > params.touchAngle : 90 - touchAngle
> params.touchAngle;
        }
      }
    }

    if (data.isScrolling) {
      swiper.emit('touchMoveOpposite', e);
    }

    if (typeof data.startMoving === 'undefined') {
      if (touches.currentX !== touches.startX || touches.currentY !== touches.startY) {
        data.startMoving = true;
      }
    }

    if (data.isScrolling) {
      data.isTouched = false;
```

```
        return;
      }

      if (!data.startMoving) {
        return;
      }

      swiper.allowClick = false;

      if (!params.cssMode && e.cancelable) {
        e.preventDefault();
      }

      if (params.touchMoveStopPropagation && !params.nested) {
        e.stopPropagation();
      }

      if (!data.isMoved) {
        if (params.loop) {
          swiper.loopFix();
        }

        data.startTranslate = swiper.getTranslate();
        swiper.setTransition(0);

        if (swiper.animating) {
          swiper.$wrapperEl.trigger('webkitTransitionEnd transitionend');
        }

        data.allowMomentumBounce = false; // Grab Cursor

        if (params.grabCursor && (swiper.allowSlideNext === true || swiper.allowSlidePrev === true)) {
          swiper.setGrabCursor(true);
        }

        swiper.emit('sliderFirstMove', e);
      }

      swiper.emit('sliderMove', e);
      data.isMoved = true;
      var diff = swiper.isHorizontal() ? diffX : diffY;
      touches.diff = diff;
      diff *= params.touchRatio;
      if (rtl) diff = -diff;
      swiper.swipeDirection = diff > 0 ? 'prev' : 'next';
      data.currentTranslate = diff + data.startTranslate;
      var disableParentSwiper = true;
      var resistanceRatio = params.resistanceRatio;

      if (params.touchReleaseOnEdges) {
        resistanceRatio = 0;
      }

      if (diff > 0 && data.currentTranslate > swiper.minTranslate()) {
        disableParentSwiper = false;
        if (params.resistance) data.currentTranslate = swiper.minTranslate() - 1 + Math.pow(-
  swiper.minTranslate() + data.startTranslate + diff, resistanceRatio);
      } else if (diff < 0 && data.currentTranslate < swiper.maxTranslate()) {
        disableParentSwiper = false;
        if (params.resistance) data.currentTranslate = swiper.maxTranslate() + 1 -
  Math.pow(swiper.maxTranslate() - data.startTranslate - diff, resistanceRatio);
      }

      if (disableParentSwiper) {
        e.preventedByNestedSwiper = true;
```

```
    } // Directions locks


    if (!swiper.allowSlideNext && swiper.swipeDirection === 'next' && data.currentTranslate <
data.startTranslate) {
      data.currentTranslate = data.startTranslate;
    }

    if (!swiper.allowSlidePrev && swiper.swipeDirection === 'prev' && data.currentTranslate >
data.startTranslate) {
      data.currentTranslate = data.startTranslate;
    }

    if (!swiper.allowSlidePrev && !swiper.allowSlideNext) {
      data.currentTranslate = data.startTranslate;
    } // Threshold


    if (params.threshold > 0) {
      if (Math.abs(diff) > params.threshold || data.allowThresholdMove) {
        if (!data.allowThresholdMove) {
          data.allowThresholdMove = true;
          touches.startX = touches.currentX;
          touches.startY = touches.currentY;
          data.currentTranslate = data.startTranslate;
          touches.diff = swiper.isHorizontal() ? touches.currentX - touches.startX : touches.currentY
- touches.startY;
          return;
        }
      } else {
        data.currentTranslate = data.startTranslate;
        return;
      }
    }

    if (!params.followFinger || params.cssMode) return; // Update active index in free mode

    if (params.freeMode || params.watchSlidesProgress || params.watchSlidesVisibility) {
      swiper.updateActiveIndex();
      swiper.updateSlidesClasses();
    }

    if (params.freeMode) {
      // Velocity
      if (data.velocities.length === 0) {
        data.velocities.push({
          position: touches[swiper.isHorizontal() ? 'startX' : 'startY'],
          time: data.touchStartTime
        });
      }

      data.velocities.push({
        position: touches[swiper.isHorizontal() ? 'currentX' : 'currentY'],
        time: now()
      });
    } // Update progress


    swiper.updateProgress(data.currentTranslate); // Update translate

    swiper.setTranslate(data.currentTranslate);
  }

  function onTouchEnd(event) {
    var swiper = this;
    var data = swiper.touchEventsData;
    var params = swiper.params,
        touches = swiper.touches,
        rtl = swiper.rtlTranslate,
        $wrapperEl = swiper.$wrapperEl,
        slidesGrid = swiper.slidesGrid,
        snapGrid = swiper.snapGrid,
        enabled = swiper.enabled;
    if (!enabled) return;
    var e = event;
    if (e.originalEvent) e = e.originalEvent;

    if (data.allowTouchCallbacks) {
      swiper.emit('touchEnd', e);
    }

    data.allowTouchCallbacks = false;

    if (!data.isTouched) {
      if (data.isMoved && params.grabCursor) {
        swiper.setGrabCursor(false);
      }

      data.isMoved = false;
      data.startMoving = false;
      return;
    } // Return Grab Cursor


    if (params.grabCursor && data.isMoved && data.isTouched && (swiper.allowSlideNext === true ||
swiper.allowSlidePrev === true)) {
      swiper.setGrabCursor(false);
    } // Time diff


    var touchEndTime = now();
    var timeDiff = touchEndTime - data.touchStartTime; // Tap, doubleTap, Click

    if (swiper.allowClick) {
      swiper.updateClickedSlide(e);
      swiper.emit('tap click', e);

      if (timeDiff < 300 && touchEndTime - data.lastClickTime < 300) {
        swiper.emit('doubleTap doubleClick', e);
      }
    }

    data.lastClickTime = now();
    nextTick(function () {
      if (!swiper.destroyed) swiper.allowClick = true;
    });

    if (!data.isTouched || !data.isMoved || !swiper.swipeDirection || touches.diff === 0 ||
data.currentTranslate === data.startTranslate) {
      data.isTouched = false;
      data.isMoved = false;
      data.startMoving = false;
      return;
    }

    data.isTouched = false;
    data.isMoved = false;
    data.startMoving = false;
    var currentPos;
```

```
      if (params.followFinger) {
        currentPos = rtl ? swiper.translate : -swiper.translate;
      } else {
        currentPos = -data.currentTranslate;
      }

      if (params.cssMode) {
        return;
      }

      if (params.freeMode) {
        if (currentPos < -swiper.minTranslate()) {
          swiper.slideTo(swiper.activeIndex);
          return;
        }

        if (currentPos > -swiper.maxTranslate()) {
          if (swiper.slides.length < snapGrid.length) {
            swiper.slideTo(snapGrid.length - 1);
          } else {
            swiper.slideTo(swiper.slides.length - 1);
          }

          return;
        }

        if (params.freeModeMomentum) {
          if (data.velocities.length > 1) {
            var lastMoveEvent = data.velocities.pop();
            var velocityEvent = data.velocities.pop();
            var distance = lastMoveEvent.position - velocityEvent.position;
            var time = lastMoveEvent.time - velocityEvent.time;
            swiper.velocity = distance / time;
            swiper.velocity /= 2;

            if (Math.abs(swiper.velocity) < params.freeModeMinimumVelocity) {
              swiper.velocity = 0;
            } // this implies that the user stopped moving a finger then released.
            // There would be no events with distance zero, so the last event is stale.


            if (time > 150 || now() - lastMoveEvent.time > 300) {
              swiper.velocity = 0;
            }
          } else {
            swiper.velocity = 0;
          }

          swiper.velocity *= params.freeModeMomentumVelocityRatio;
          data.velocities.length = 0;
          var momentumDuration = 1000 * params.freeModeMomentumRatio;
          var momentumDistance = swiper.velocity * momentumDuration;
          var newPosition = swiper.translate + momentumDistance;
          if (rtl) newPosition = -newPosition;
          var doBounce = false;
          var afterBouncePosition;
          var bounceAmount = Math.abs(swiper.velocity) * 20 * params.freeModeMomentumBounceRatio;
          var needsLoopFix;

          if (newPosition < swiper.maxTranslate()) {
            if (params.freeModeMomentumBounce) {
              if (newPosition + swiper.maxTranslate() < -bounceAmount) {
                newPosition = swiper.maxTranslate() - bounceAmount;
              }
              afterBouncePosition = swiper.maxTranslate();
              doBounce = true;
              data.allowMomentumBounce = true;
            } else {
              newPosition = swiper.maxTranslate();
            }

            if (params.loop && params.centeredSlides) needsLoopFix = true;
          } else if (newPosition > swiper.minTranslate()) {
            if (params.freeModeMomentumBounce) {
              if (newPosition - swiper.minTranslate() > bounceAmount) {
                newPosition = swiper.minTranslate() + bounceAmount;
              }

              afterBouncePosition = swiper.minTranslate();
              doBounce = true;
              data.allowMomentumBounce = true;
            } else {
              newPosition = swiper.minTranslate();
            }

            if (params.loop && params.centeredSlides) needsLoopFix = true;
          } else if (params.freeModeSticky) {
            var nextSlide;

            for (var j = 0; j < snapGrid.length; j += 1) {
              if (snapGrid[j] > -newPosition) {
                nextSlide = j;
                break;
              }
            }

            if (Math.abs(snapGrid[nextSlide] - newPosition) < Math.abs(snapGrid[nextSlide - 1] -
  newPosition) || swiper.swipeDirection === 'next') {
              newPosition = snapGrid[nextSlide];
            } else {
              newPosition = snapGrid[nextSlide - 1];
            }

            newPosition = -newPosition;
          }

          if (needsLoopFix) {
            swiper.once('transitionEnd', function () {
              swiper.loopFix();
            });
          } // Fix duration


          if (swiper.velocity !== 0) {
            if (rtl) {
              momentumDuration = Math.abs((-newPosition - swiper.translate) / swiper.velocity);
            } else {
              momentumDuration = Math.abs((newPosition - swiper.translate) / swiper.velocity);
            }

            if (params.freeModeSticky) {
              // If freeModeSticky is active and the user ends a swipe with a slow-velocity
              // event, then durations can be 20+ seconds to slide one (or zero!) slides.
              // It's easy to see this when simulating touch with mouse events. To fix this,
              // limit single-slide swipes to the default slide duration. This also has the
              // nice side effect of matching slide speed if the user stopped moving before
              // lifting finger or mouse vs. moving slowly before lifting the finger/mouse.
              // For faster swipes, also apply limits (albeit higher ones).
              var moveDistance = Math.abs((rtl ? -newPosition : newPosition) - swiper.translate);
              var currentSlideSize = swiper.slidesSizesGrid[swiper.activeIndex];

              if (moveDistance < currentSlideSize) {
                momentumDuration = params.speed;
              } else if (moveDistance < 2 * currentSlideSize) {
                momentumDuration = params.speed * 1.5;
              } else {
                momentumDuration = params.speed * 2.5;
              }
            }
          } else if (params.freeModeSticky) {
            swiper.slideToClosest();
            return;
          }

          if (params.freeModeMomentumBounce && doBounce) {
            swiper.updateProgress(afterBouncePosition);
            swiper.setTransition(momentumDuration);
            swiper.setTranslate(newPosition);
            swiper.transitionStart(true, swiper.swipeDirection);
            swiper.animating = true;
            $wrapperEl.transitionEnd(function () {
              if (!swiper || swiper.destroyed || !data.allowMomentumBounce) return;
              swiper.emit('momentumBounce');
              swiper.setTransition(params.speed);
              setTimeout(function () {
                swiper.setTranslate(afterBouncePosition);
                $wrapperEl.transitionEnd(function () {
                  if (!swiper || swiper.destroyed) return;
                  swiper.transitionEnd();
                });
              }, 0);
            });
          } else if (swiper.velocity) {
            swiper.updateProgress(newPosition);
            swiper.setTransition(momentumDuration);
            swiper.setTranslate(newPosition);
            swiper.transitionStart(true, swiper.swipeDirection);

            if (!swiper.animating) {
              swiper.animating = true;
              $wrapperEl.transitionEnd(function () {
                if (!swiper || swiper.destroyed) return;
                swiper.transitionEnd();
              });
            }
          } else {
            swiper.emit('_freeModeNoMomentumRelease');
            swiper.updateProgress(newPosition);
          }

          swiper.updateActiveIndex();
          swiper.updateSlidesClasses();
        } else if (params.freeModeSticky) {
          swiper.slideToClosest();
          return;
        } else if (params.freeMode) {
          swiper.emit('_freeModeNoMomentumRelease');
        }

        if (!params.freeModeMomentum || timeDiff >= params.longSwipesMs) {
          swiper.updateProgress();
          swiper.updateActiveIndex();
          swiper.updateSlidesClasses();
        }
        return;
      } // Find current slide


      var stopIndex = 0;
      var groupSize = swiper.slidesSizesGrid[0];

      for (var i = 0; i < slidesGrid.length; i += i < params.slidesPerGroupSkip ? 1 :
  params.slidesPerGroup) {
        var _increment = i < params.slidesPerGroupSkip - 1 ? 1 : params.slidesPerGroup;

        if (typeof slidesGrid[i + _increment] !== 'undefined') {
          if (currentPos >= slidesGrid[i] && currentPos < slidesGrid[i + _increment]) {
            stopIndex = i;
            groupSize = slidesGrid[i + _increment] - slidesGrid[i];
          }
        } else if (currentPos >= slidesGrid[i]) {
          stopIndex = i;
          groupSize = slidesGrid[slidesGrid.length - 1] - slidesGrid[slidesGrid.length - 2];
        }
      } // Find current slide size


      var ratio = (currentPos - slidesGrid[stopIndex]) / groupSize;
      var increment = stopIndex < params.slidesPerGroupSkip - 1 ? 1 : params.slidesPerGroup;

      if (timeDiff > params.longSwipesMs) {
        // Long touches
        if (!params.longSwipes) {
          swiper.slideTo(swiper.activeIndex);
          return;
        }

        if (swiper.swipeDirection === 'next') {
          if (ratio >= params.longSwipesRatio) swiper.slideTo(stopIndex + increment);else
  swiper.slideTo(stopIndex);
        }

        if (swiper.swipeDirection === 'prev') {
          if (ratio > 1 - params.longSwipesRatio) swiper.slideTo(stopIndex + increment);else
  swiper.slideTo(stopIndex);
        }
      } else {
        // Short swipes
        if (!params.shortSwipes) {
          swiper.slideTo(swiper.activeIndex);
          return;
        }

        var isNavButtonTarget = swiper.navigation && (e.target === swiper.navigation.nextEl || e.target
  === swiper.navigation.prevEl);

        if (!isNavButtonTarget) {
          if (swiper.swipeDirection === 'next') {
            swiper.slideTo(stopIndex + increment);
          }

          if (swiper.swipeDirection === 'prev') {
            swiper.slideTo(stopIndex);
          }
        } else if (e.target === swiper.navigation.nextEl) {
          swiper.slideTo(stopIndex + increment);
        } else {
          swiper.slideTo(stopIndex);
        }
      }
    }

    function onResize() {
      var swiper = this;
      var params = swiper.params,
          el = swiper.el;
      if (el && el.offsetWidth === 0) return; // Breakpoints

      if (params.breakpoints) {
        swiper.setBreakpoint();
      } // Save locks


      var allowSlideNext = swiper.allowSlideNext,
          allowSlidePrev = swiper.allowSlidePrev,
          snapGrid = swiper.snapGrid; // Disable locks on resize

      swiper.allowSlideNext = true;
      swiper.allowSlidePrev = true;
      swiper.updateSize();
      swiper.updateSlides();
      swiper.updateSlidesClasses();

      if ((params.slidesPerView === 'auto' || params.slidesPerView > 1) && swiper.isEnd &&
  !swiper.isBeginning && !swiper.params.centeredSlides) {
        swiper.slideTo(swiper.slides.length - 1, 0, false, true);
      } else {
        swiper.slideTo(swiper.activeIndex, 0, false, true);
      }

      if (swiper.autoplay && swiper.autoplay.running && swiper.autoplay.paused) {
        swiper.autoplay.run();
      } // Return locks after resize


      swiper.allowSlidePrev = allowSlidePrev;
      swiper.allowSlideNext = allowSlideNext;

      if (swiper.params.watchOverflow && snapGrid !== swiper.snapGrid) {
        swiper.checkOverflow();
      }
    }

    function onClick(e) {
      var swiper = this;
      if (!swiper.enabled) return;

      if (!swiper.allowClick) {
        if (swiper.params.preventClicks) e.preventDefault();

        if (swiper.params.preventClicksPropagation && swiper.animating) {
          e.stopPropagation();
          e.stopImmediatePropagation();
        }
      }
    }

    function onScroll() {
      var swiper = this;
      var wrapperEl = swiper.wrapperEl,
          rtlTranslate = swiper.rtlTranslate,
          enabled = swiper.enabled;
      if (!enabled) return;
      swiper.previousTranslate = swiper.translate;

      if (swiper.isHorizontal()) {
        if (rtlTranslate) {
          swiper.translate = wrapperEl.scrollWidth - wrapperEl.offsetWidth - wrapperEl.scrollLeft;
        } else {
          swiper.translate = -wrapperEl.scrollLeft;
        }
      } else {
        swiper.translate = -wrapperEl.scrollTop;
      } // eslint-disable-next-line


      if (swiper.translate === -0) swiper.translate = 0;
      swiper.updateActiveIndex();
      swiper.updateSlidesClasses();
      var newProgress;
      var translatesDiff = swiper.maxTranslate() - swiper.minTranslate();

      if (translatesDiff === 0) {
        newProgress = 0;
      } else {
        newProgress = (swiper.translate - swiper.minTranslate()) / translatesDiff;
      }

      if (newProgress !== swiper.progress) {
        swiper.updateProgress(rtlTranslate ? -swiper.translate : swiper.translate);
      }

      swiper.emit('setTranslate', swiper.translate, false);
    }

    var dummyEventAttached = false;

    function dummyEventListener() {}

    function attachEvents() {
      var swiper = this;
      var document = getDocument();
      var params = swiper.params,
          touchEvents = swiper.touchEvents,
          el = swiper.el,
          wrapperEl = swiper.wrapperEl,
          device = swiper.device,
          support = swiper.support;
      swiper.onTouchStart = onTouchStart.bind(swiper);
      swiper.onTouchMove = onTouchMove.bind(swiper);
      swiper.onTouchEnd = onTouchEnd.bind(swiper);

      if (params.cssMode) {
        swiper.onScroll = onScroll.bind(swiper);
      }

      swiper.onClick = onClick.bind(swiper);
      var capture = !!params.nested; // Touch Events

      if (!support.touch && support.pointerEvents) {
        el.addEventListener(touchEvents.start, swiper.onTouchStart, false);
        document.addEventListener(touchEvents.move, swiper.onTouchMove, capture);
        document.addEventListener(touchEvents.end, swiper.onTouchEnd, false);
      } else {
        if (support.touch) {
          var passiveListener = touchEvents.start === 'touchstart' && support.passiveListener &&
  params.passiveListeners ? {
            passive: true,
            capture: false
          } : false;
          el.addEventListener(touchEvents.start, swiper.onTouchStart, passiveListener);
          el.addEventListener(touchEvents.move, swiper.onTouchMove, support.passiveListener ? {
            passive: false,
            capture: capture
          } : capture);
          el.addEventListener(touchEvents.end, swiper.onTouchEnd, passiveListener);

          if (touchEvents.cancel) {
            el.addEventListener(touchEvents.cancel, swiper.onTouchEnd, passiveListener);
          }

          if (!dummyEventAttached) {
            document.addEventListener('touchstart', dummyEventListener);
            dummyEventAttached = true;
          }
        }

        if (params.simulateTouch && !device.ios && !device.android || params.simulateTouch &&
  !support.touch && device.ios) {
          el.addEventListener('mousedown', swiper.onTouchStart, false);
          document.addEventListener('mousemove', swiper.onTouchMove, capture);
          document.addEventListener('mouseup', swiper.onTouchEnd, false);
        }
      } // Prevent Links Clicks


      if (params.preventClicks || params.preventClicksPropagation) {
        el.addEventListener('click', swiper.onClick, true);
      }

      if (params.cssMode) {
        wrapperEl.addEventListener('scroll', swiper.onScroll);
      } // Resize handler


      if (params.updateOnWindowResize) {
        swiper.on(device.ios || device.android ? 'resize orientationchange observerUpdate' : 'resize
  observerUpdate', onResize, true);
      } else {
        swiper.on('observerUpdate', onResize, true);
      }
    }

    function detachEvents() {
      var swiper = this;
      var document = getDocument();
      var params = swiper.params,
          touchEvents = swiper.touchEvents,
          el = swiper.el,
          wrapperEl = swiper.wrapperEl,
          device = swiper.device,
          support = swiper.support;
      var capture = !!params.nested; // Touch Events

      if (!support.touch && support.pointerEvents) {
        el.removeEventListener(touchEvents.start, swiper.onTouchStart, false);
        document.removeEventListener(touchEvents.move, swiper.onTouchMove, capture);
        document.removeEventListener(touchEvents.end, swiper.onTouchEnd, false);
      } else {
        if (support.touch) {
          var passiveListener = touchEvents.start === 'onTouchStart' && support.passiveListener &&
  params.passiveListeners ? {
            passive: true,
            capture: false
          } : false;
          el.removeEventListener(touchEvents.start, swiper.onTouchStart, passiveListener);
          el.removeEventListener(touchEvents.move, swiper.onTouchMove, capture);
          el.removeEventListener(touchEvents.end, swiper.onTouchEnd, passiveListener);

          if (touchEvents.cancel) {
            el.removeEventListener(touchEvents.cancel, swiper.onTouchEnd, passiveListener);
          }
        }

        if (params.simulateTouch && !device.ios && !device.android || params.simulateTouch &&
  !support.touch && device.ios) {
          el.removeEventListener('mousedown', swiper.onTouchStart, false);
          document.removeEventListener('mousemove', swiper.onTouchMove, capture);
          document.removeEventListener('mouseup', swiper.onTouchEnd, false);
        }
      } // Prevent Links Clicks


      if (params.preventClicks || params.preventClicksPropagation) {
        el.removeEventListener('click', swiper.onClick, true);
      }

      if (params.cssMode) {
        wrapperEl.removeEventListener('scroll', swiper.onScroll);
      } // Resize handler


      swiper.off(device.ios || device.android ? 'resize orientationchange observerUpdate' : 'resize
  observerUpdate', onResize);
    }

    var events = {
      attachEvents: attachEvents,
      detachEvents: detachEvents
    };

    function setBreakpoint() {
      var swiper = this;
      var activeIndex = swiper.activeIndex,
          initialized = swiper.initialized,
          _swiper$loopedSlides = swiper.loopedSlides,
          loopedSlides = _swiper$loopedSlides === void 0 ? 0 : _swiper$loopedSlides,
          params = swiper.params,
          $el = swiper.$el;
      var breakpoints = params.breakpoints;
      if (!breakpoints || breakpoints && Object.keys(breakpoints).length === 0) return; // Get
  breakpoint for window width and update parameters

      var breakpoint = swiper.getBreakpoint(breakpoints, swiper.params.breakpointsBase, swiper.el);
      if (!breakpoint || swiper.currentBreakpoint === breakpoint) return;
      var breakpointOnlyParams = breakpoint in breakpoints ? breakpoints[breakpoint] : undefined;

      if (breakpointOnlyParams) {
        ['slidesPerView', 'spaceBetween', 'slidesPerGroup', 'slidesPerGroupSkip',
  'slidesPerColumn'].forEach(function (param) {
          var paramValue = breakpointOnlyParams[param];
          if (typeof paramValue === 'undefined') return;

          if (param === 'slidesPerView' && (paramValue === 'AUTO' || paramValue === 'auto')) {
            breakpointOnlyParams[param] = 'auto';
          } else if (param === 'slidesPerView') {
            breakpointOnlyParams[param] = parseFloat(paramValue);
          } else {
            breakpointOnlyParams[param] = parseInt(paramValue, 10);
          }
        });
      }

      var breakpointParams = breakpointOnlyParams || swiper.originalParams;
      var wasMultiRow = params.slidesPerColumn > 1;
      var isMultiRow = breakpointParams.slidesPerColumn > 1;
      var wasEnabled = params.enabled;

      if (wasMultiRow && !isMultiRow) {
        $el.removeClass(params.containerModifierClass + "multirow " + params.containerModifierClass +
  "multirow-column");
        swiper.emitContainerClasses();
      } else if (!wasMultiRow && isMultiRow) {
        $el.addClass(params.containerModifierClass + "multirow");

        if (breakpointParams.slidesPerColumnFill && breakpointParams.slidesPerColumnFill === 'column'
  || !breakpointParams.slidesPerColumnFill && params.slidesPerColumnFill === 'column') {
          $el.addClass(params.containerModifierClass + "multirow-column");
        }

        swiper.emitContainerClasses();
      }

      var directionChanged = breakpointParams.direction && breakpointParams.direction !==
  params.direction;
      var needsReLoop = params.loop && (breakpointParams.slidesPerView !== params.slidesPerView ||
  directionChanged);

      if (directionChanged && initialized) {
        swiper.changeDirection();
      }

      extend(swiper.params, breakpointParams);
      var isEnabled = swiper.params.enabled;
      extend(swiper, {
        allowTouchMove: swiper.params.allowTouchMove,
        allowSlideNext: swiper.params.allowSlideNext,
        allowSlidePrev: swiper.params.allowSlidePrev
      });

      if (wasEnabled && !isEnabled) {
        swiper.disable();
      } else if (!wasEnabled && isEnabled) {
        swiper.enable();
      }

      swiper.currentBreakpoint = breakpoint;
      swiper.emit('_beforeBreakpoint', breakpointParams);

      if (needsReLoop && initialized) {
        swiper.loopDestroy();
        swiper.loopCreate();
        swiper.updateSlides();
        swiper.slideTo(activeIndex - loopedSlides + swiper.loopedSlides, 0, false);
      }

      swiper.emit('breakpoint', breakpointParams);
    }

    function getBreakpoint(breakpoints, base, containerEl) {
      if (base === void 0) {
        base = 'window';
      }
```

```
      if (!breakpoints || base === 'container' && !containerEl) return undefined;
      var breakpoint = false;
      var window = getWindow();
      var currentHeight = base === 'window' ? window.innerHeight : containerEl.clientHeight;
      var points = Object.keys(breakpoints).map(function (point) {
        if (typeof point === 'string' && point.indexOf('@') === 0) {
          var minRatio = parseFloat(point.substr(1));
          var value = currentHeight * minRatio;
          return {
            value: value,
            point: point
          };
        }

        return {
          value: point,
          point: point
        };
      });
      points.sort(function (a, b) {
        return parseInt(a.value, 10) - parseInt(b.value, 10);
      });

      for (var i = 0; i < points.length; i += 1) {
        var _points$i = points[i],
            point = _points$i.point,
            value = _points$i.value;

        if (base === 'window') {
          if (window.matchMedia("(min-width: " + value + "px)").matches) {
            breakpoint = point;
          }
        } else if (value <= containerEl.clientWidth) {
          breakpoint = point;
        }
      }

      return breakpoint || 'max';
    }

    var breakpoints = {
      setBreakpoint: setBreakpoint,
      getBreakpoint: getBreakpoint
    };

    function prepareClasses(entries, prefix) {
      var resultClasses = [];
      entries.forEach(function (item) {
        if (typeof item === 'object') {
          Object.keys(item).forEach(function (classNames) {
            if (item[classNames]) {
              resultClasses.push(prefix + classNames);
            }
          });
        } else if (typeof item === 'string') {
          resultClasses.push(prefix + item);
        }
      });
      return resultClasses;
    }

    function addClasses() {
      var swiper = this;
      var classNames = swiper.classNames,
          params = swiper.params,
          rtl = swiper.rtl,
          $el = swiper.$el,
          device = swiper.device,
          support = swiper.support; // prettier-ignore

      var suffixes = prepareClasses(['initialized', params.direction, {
        'pointer-events': support.pointerEvents && !support.touch
      }, {
        'free-mode': params.freeMode
      }, {
        'autoheight': params.autoHeight
      }, {
        'rtl': rtl
      }, {
        'multirow': params.slidesPerColumn > 1
      }, {
        'multirow-column': params.slidesPerColumn > 1 && params.slidesPerColumnFill === 'column'
      }, {
        'android': device.android
      }, {
        'ios': device.ios
      }, {
        'css-mode': params.cssMode
      }], params.containerModifierClass);
      classNames.push.apply(classNames, suffixes);
      $el.addClass([].concat(classNames).join(' '));
      swiper.emitContainerClasses();
    }

    function removeClasses() {
      var swiper = this;
      var $el = swiper.$el,
          classNames = swiper.classNames;
      $el.removeClass(classNames.join(' '));
      swiper.emitContainerClasses();
    }

    var classes = {
      addClasses: addClasses,
      removeClasses: removeClasses
    };

    function loadImage(imageEl, src, srcset, sizes, checkForComplete, callback) {
      var window = getWindow();
      var image;

      function onReady() {
        if (callback) callback();
      }

      var isPicture = $(imageEl).parent('picture')[0];

      if (!isPicture && (!imageEl.complete || !checkForComplete)) {
        if (src) {
          image = new window.Image();
          image.onload = onReady;
          image.onerror = onReady;

          if (sizes) {
            image.sizes = sizes;
          }

          if (srcset) {
            image.srcset = srcset;
          }

          if (src) {
            image.src = src;
          }
        } else {
          onReady();
        }
      } else {
        // image already loaded...
        onReady();
      }
    }

    function preloadImages() {
      var swiper = this;
      swiper.imagesToLoad = swiper.$el.find('img');

      function onReady() {
        if (typeof swiper === 'undefined' || swiper === null || !swiper || swiper.destroyed) return;
        if (swiper.imagesLoaded !== undefined) swiper.imagesLoaded += 1;

        if (swiper.imagesLoaded === swiper.imagesToLoad.length) {
          if (swiper.params.updateOnImagesReady) swiper.update();
          swiper.emit('imagesReady');
        }
      }

      for (var i = 0; i < swiper.imagesToLoad.length; i += 1) {
        var imageEl = swiper.imagesToLoad[i];
        swiper.loadImage(imageEl, imageEl.currentSrc || imageEl.getAttribute('src'), imageEl.srcset ||
  imageEl.getAttribute('srcset'), imageEl.sizes || imageEl.getAttribute('sizes'), true, onReady);
      }
    }

    var images = {
      loadImage: loadImage,
      preloadImages: preloadImages
    };

    function checkOverflow() {
      var swiper = this;
      var params = swiper.params;
      var wasLocked = swiper.isLocked;
      var lastSlidePosition = swiper.slides.length > 0 && params.slidesOffsetBefore +
  params.spaceBetween * (swiper.slides.length - 1) + swiper.slides[0].offsetWidth *
  swiper.slides.length;

      if (params.slidesOffsetBefore && params.slidesOffsetAfter && lastSlidePosition) {
        swiper.isLocked = lastSlidePosition <= swiper.size;
      } else {
        swiper.isLocked = swiper.snapGrid.length === 1;
      }

      swiper.allowSlideNext = !swiper.isLocked;
      swiper.allowSlidePrev = !swiper.isLocked; // events

      if (wasLocked !== swiper.isLocked) swiper.emit(swiper.isLocked ? 'lock' : 'unlock');

      if (wasLocked && wasLocked !== swiper.isLocked) {
        swiper.isEnd = false;
        if (swiper.navigation) swiper.navigation.update();
      }
    }
    var checkOverflow$1 = {
      checkOverflow: checkOverflow
    };

    var defaults = {
      init: true,
      direction: 'horizontal',
      touchEventsTarget: 'container',
      initialSlide: 0,
      speed: 300,
      cssMode: false,
      updateOnWindowResize: true,
      resizeObserver: false,
      nested: false,
      createElements: false,
      enabled: true,
      focusableElements: 'input, select, option, textarea, button, video, label',
      // Overrides
      width: null,
      height: null,
      //
      preventInteractionOnTransition: false,
      // ssr
      userAgent: null,
      url: null,
      // To support iOS's swipe-to-go-back gesture (when being used in-app).
      edgeSwipeDetection: false,
      edgeSwipeThreshold: 20,
      // Free mode
      freeMode: false,
      freeModeMomentum: true,
      freeModeMomentumRatio: 1,
      freeModeMomentumBounce: true,
      freeModeMomentumBounceRatio: 1,
      freeModeMomentumVelocityRatio: 1,
      freeModeSticky: false,
      freeModeMinimumVelocity: 0.02,
      // Autoheight
      autoHeight: false,
      // Set wrapper width
      setWrapperSize: false,
      // Virtual Translate
      virtualTranslate: false,
      // Effects
      effect: 'slide',
      // 'slide' or 'fade' or 'cube' or 'coverflow' or 'flip'
      // Breakpoints
      breakpoints: undefined,
      breakpointsBase: 'window',
      // Slides grid
      spaceBetween: 0,
      slidesPerView: 1,
      slidesPerColumn: 1,
      slidesPerColumnFill: 'column',
      slidesPerGroup: 1,
      slidesPerGroupSkip: 0,
      centeredSlides: false,
      centeredSlidesBounds: false,
      slidesOffsetBefore: 0,
      // in px
      slidesOffsetAfter: 0,
      // in px
      normalizeSlideIndex: true,
      centerInsufficientSlides: false,
      // Disable swiper and hide navigation when container not overflow
      watchOverflow: false,
      // Round length
      roundLengths: false,
      // Touches
      touchRatio: 1,
      touchAngle: 45,
      simulateTouch: true,
      shortSwipes: true,
      longSwipes: true,
      longSwipesRatio: 0.5,
      longSwipesMs: 300,
      followFinger: true,
      allowTouchMove: true,
      threshold: 0,
      touchMoveStopPropagation: false,
      touchStartPreventDefault: true,
      touchStartForcePreventDefault: false,
      touchReleaseOnEdges: false,
      // Unique Navigation Elements
      uniqueNavElements: true,
      // Resistance
      resistance: true,
      resistanceRatio: 0.85,
      // Progress
      watchSlidesProgress: false,
      watchSlidesVisibility: false,
      // Cursor
      grabCursor: false,
      // Clicks
      preventClicks: true,
      preventClicksPropagation: true,
      slideToClickedSlide: false,
      // Images
      preloadImages: true,
      updateOnImagesReady: true,
      // loop
      loop: false,
      loopAdditionalSlides: 0,
      loopedSlides: null,
      loopFillGroupWithBlank: false,
      loopPreventsSlide: true,
      // Swiping/no swiping
      allowSlidePrev: true,
      allowSlideNext: true,
      swipeHandler: null,
      // '.swipe-handler',
      noSwiping: true,
      noSwipingClass: 'swiper-no-swiping',
      noSwipingSelector: null,
      // Passive Listeners
      passiveListeners: true,
      // NS
      containerModifierClass: 'swiper-container-',
      // NEW
      slideClass: 'swiper-slide',
      slideBlankClass: 'swiper-slide-invisible-blank',
      slideActiveClass: 'swiper-slide-active',
      slideDuplicateActiveClass: 'swiper-slide-duplicate-active',
      slideVisibleClass: 'swiper-slide-visible',
      slideDuplicateClass: 'swiper-slide-duplicate',
      slideNextClass: 'swiper-slide-next',
      slideDuplicateNextClass: 'swiper-slide-duplicate-next',
      slidePrevClass: 'swiper-slide-prev',
      slideDuplicatePrevClass: 'swiper-slide-duplicate-prev',
      wrapperClass: 'swiper-wrapper',
      // Callbacks
      runCallbacksOnInit: true,
      // Internals
      _emitClasses: false
    };

    var prototypes = {
      modular: modular,
      eventsEmitter: eventsEmitter,
      update: update,
      translate: translate,
      transition: transition,
      slide: slide,
      loop: loop,
      grabCursor: grabCursor,
      manipulation: manipulation,
      events: events,
      breakpoints: breakpoints,
      checkOverflow: checkOverflow$1,
      classes: classes,
      images: images
    };
    var extendedDefaults = {};

    var Swiper = /*#__PURE__*/function () {
      function Swiper() {
        var el;
        var params;

        for (var _len = arguments.length, args = new Array(_len), _key = 0; _key < _len; _key++) {
          args[_key] = arguments[_key];
        }

        if (args.length === 1 && args[0].constructor &&
  Object.prototype.toString.call(args[0]).slice(8, -1) === 'Object') {
          params = args[0];
        } else {
          el = args[0];
          params = args[1];
        }

        if (!params) params = {};
        params = extend({}, params);
        if (el && !params.el) params.el = el;

        if (params.el && $(params.el).length > 1) {
          var swipers = [];
          $(params.el).each(function (containerEl) {
            var newParams = extend({}, params, {
              el: containerEl
            });
            swipers.push(new Swiper(newParams));
          });
          return swipers;
        } // Swiper Instance


        var swiper = this;
        swiper.__swiper__ = true;
        swiper.support = getSupport();
        swiper.device = getDevice({
          userAgent: params.userAgent
        });
        swiper.browser = getBrowser();
        swiper.eventsListeners = {};
        swiper.eventsAnyListeners = [];

        if (typeof swiper.modules === 'undefined') {
          swiper.modules = {};
        }

        Object.keys(swiper.modules).forEach(function (moduleName) {
          var module = swiper.modules[moduleName];

          if (module.params) {
            var moduleParamName = Object.keys(module.params)[0];
            var moduleParams = module.params[moduleParamName];
            if (typeof moduleParams !== 'object' || moduleParams === null) return;

            if (['navigation', 'pagination', 'scrollbar'].indexOf(moduleParamName) >= 0 &&
  params[moduleParamName] === true) {
              params[moduleParamName] = {
                auto: true
              };
            }

            if (!(moduleParamName in params && 'enabled' in moduleParams)) return;

            if (params[moduleParamName] === true) {
              params[moduleParamName] = {
                enabled: true
              };
            }

            if (typeof params[moduleParamName] === 'object' && !('enabled' in params[moduleParamName]))
  {
              params[moduleParamName].enabled = true;
            }

            if (!params[moduleParamName]) params[moduleParamName] = {
              enabled: false
            };
          }
        }); // Extend defaults with modules params

        var swiperParams = extend({}, defaults);
        swiper.useParams(swiperParams); // Extend defaults with passed params

        swiper.params = extend({}, swiperParams, extendedDefaults, params);
        swiper.originalParams = extend({}, swiper.params);
        swiper.passedParams = extend({}, params); // add event listeners

        if (swiper.params && swiper.params.on) {
          Object.keys(swiper.params.on).forEach(function (eventName) {
            swiper.on(eventName, swiper.params.on[eventName]);
          });
        }

        if (swiper.params && swiper.params.onAny) {
          swiper.onAny(swiper.params.onAny);
        } // Save Dom lib


        swiper.$ = $; // Extend Swiper

        extend(swiper, {
          enabled: swiper.params.enabled,
          el: el,
          // Classes
          classNames: [],
          // Slides
          slides: $(),
          slidesGrid: [],
          snapGrid: [],
          slidesSizesGrid: [],
          // isDirection
          isHorizontal: function isHorizontal() {
            return swiper.params.direction === 'horizontal';
          },
          isVertical: function isVertical() {
            return swiper.params.direction === 'vertical';
          },
          // Indexes
          activeIndex: 0,
          realIndex: 0,
          //
          isBeginning: true,
          isEnd: false,
          // Props
          translate: 0,
          previousTranslate: 0,
          progress: 0,
          velocity: 0,
          animating: false,
          // Locks
          allowSlideNext: swiper.params.allowSlideNext,
          allowSlidePrev: swiper.params.allowSlidePrev,
          // Touch Events
          touchEvents: function touchEvents() {
            var touch = ['touchstart', 'touchmove', 'touchend', 'touchcancel'];
            var desktop = ['mousedown', 'mousemove', 'mouseup'];

            if (swiper.support.pointerEvents) {
              desktop = ['pointerdown', 'pointermove', 'pointerup'];
            }

            swiper.touchEventsTouch = {
              start: touch[0],
              move: touch[1],
              end: touch[2],
              cancel: touch[3]
            };
            swiper.touchEventsDesktop = {
              start: desktop[0],
              move: desktop[1],
              end: desktop[2]
            };
            return swiper.support.touch || !swiper.params.simulateTouch ? swiper.touchEventsTouch :
  swiper.touchEventsDesktop;
          }(),
          touchEventsData: {
            isTouched: undefined,
            isMoved: undefined,
            allowTouchCallbacks: undefined,
            touchStartTime: undefined,
            isScrolling: undefined,
            currentTranslate: undefined,
            startTranslate: undefined,
            allowThresholdMove: undefined,
            // Form elements to match
            focusableElements: swiper.params.focusableElements,
            // Last click time
            lastClickTime: now(),
            clickTimeout: undefined,
            // Velocities
            velocities: [],
            allowMomentumBounce: undefined,
            isTouchEvent: undefined,
            startMoving: undefined
          },
          // Clicks
          allowClick: true,
          // Touches
          allowTouchMove: swiper.params.allowTouchMove,
          touches: {
            startX: 0,
            startY: 0,
            currentX: 0,
            currentY: 0,
            diff: 0
          },
          // Images
          imagesToLoad: [],
          imagesLoaded: 0
        }); // Install Modules

        swiper.useModules();
        swiper.emit('_swiper'); // Init

        if (swiper.params.init) {
          swiper.init();
        } // Return app instance


        return swiper;
      }

      var _proto = Swiper.prototype;

      _proto.enable = function enable() {
        var swiper = this;
        if (swiper.enabled) return;
        swiper.enabled = true;

        if (swiper.params.grabCursor) {
          swiper.setGrabCursor();
        }

        swiper.emit('enable');
      };

      _proto.disable = function disable() {
        var swiper = this;
        if (!swiper.enabled) return;
        swiper.enabled = false;

        if (swiper.params.grabCursor) {
          swiper.unsetGrabCursor();
        }

        swiper.emit('disable');
      };

      _proto.setProgress = function setProgress(progress, speed) {
        var swiper = this;
        progress = Math.min(Math.max(progress, 0), 1);
        var min = swiper.minTranslate();
        var max = swiper.maxTranslate();
        var current = (max - min) * progress + min;
        swiper.translateTo(current, typeof speed === 'undefined' ? 0 : speed);
```

```
      swiper.updateActiveIndex();
      swiper.updateSlidesClasses();
    };

    _proto.emitContainerClasses = function emitContainerClasses() {
      var swiper = this;
      if (!swiper.params._emitClasses || !swiper.el) return;
      var classes = swiper.el.className.split(' ').filter(function (className) {
        return className.indexOf('swiper-container') === 0 ||
className.indexOf(swiper.params.containerModifierClass) === 0;
      });
      swiper.emit('_containerClasses', classes.join(' '));
    };

    _proto.getSlideClasses = function getSlideClasses(slideEl) {
      var swiper = this;
      return slideEl.className.split(' ').filter(function (className) {
        return className.indexOf('swiper-slide') === 0 || className.indexOf(swiper.params.slideClass)
=== 0;
      }).join(' ');
    };

    _proto.emitSlidesClasses = function emitSlidesClasses() {
      var swiper = this;
      if (!swiper.params._emitClasses || !swiper.el) return;
      var updates = [];
      swiper.slides.each(function (slideEl) {
        var classNames = swiper.getSlideClasses(slideEl);
        updates.push({
          slideEl: slideEl,
          classNames: classNames
        });
        swiper.emit('_slideClass', slideEl, classNames);
      });
      swiper.emit('_slideClasses', updates);
    };

    _proto.slidesPerViewDynamic = function slidesPerViewDynamic() {
      var swiper = this;
      var params = swiper.params,
          slides = swiper.slides,
          slidesGrid = swiper.slidesGrid,
          swiperSize = swiper.size,
          activeIndex = swiper.activeIndex;
      var spv = 1;

      if (params.centeredSlides) {
        var slideSize = slides[activeIndex].swiperSlideSize;
        var breakLoop;

        for (var i = activeIndex + 1; i < slides.length; i += 1) {
          if (slides[i] && !breakLoop) {
            slideSize += slides[i].swiperSlideSize;
            spv += 1;
            if (slideSize > swiperSize) breakLoop = true;
          }
        }

        for (var _i = activeIndex - 1; _i >= 0; _i -= 1) {
          if (slides[_i] && !breakLoop) {
            slideSize += slides[_i].swiperSlideSize;
            spv += 1;
            if (slideSize > swiperSize) breakLoop = true;
          }
        }
      } else {
        for (var _i2 = activeIndex + 1; _i2 < slides.length; _i2 += 1) {
          if (slidesGrid[_i2] - slidesGrid[activeIndex] < swiperSize) {
            spv += 1;
          }
        }
      }

      return spv;
    };

    _proto.update = function update() {
      var swiper = this;
      if (!swiper || swiper.destroyed) return;
      var snapGrid = swiper.snapGrid,
          params = swiper.params; // Breakpoints

      if (params.breakpoints) {
        swiper.setBreakpoint();
      }

      swiper.updateSize();
      swiper.updateSlides();
      swiper.updateProgress();
      swiper.updateSlidesClasses();

      function setTranslate() {
        var translateValue = swiper.rtlTranslate ? swiper.translate * -1 : swiper.translate;
        var newTranslate = Math.min(Math.max(translateValue, swiper.maxTranslate()),
swiper.minTranslate());
        swiper.setTranslate(newTranslate);
        swiper.updateActiveIndex();
        swiper.updateSlidesClasses();
      }

      var translated;

      if (swiper.params.freeMode) {
        setTranslate();

        if (swiper.params.autoHeight) {
          swiper.updateAutoHeight();
        }
      } else {
        if ((swiper.params.slidesPerView === 'auto' || swiper.params.slidesPerView > 1) &&
swiper.isEnd && !swiper.params.centeredSlides) {
          translated = swiper.slideTo(swiper.slides.length - 1, 0, false, true);
        } else {
          translated = swiper.slideTo(swiper.activeIndex, 0, false, true);
        }

        if (!translated) {
          setTranslate();
        }
      }

      if (params.watchOverflow && snapGrid !== swiper.snapGrid) {
        swiper.checkOverflow();
      }

      swiper.emit('update');
    };

    _proto.changeDirection = function changeDirection(newDirection, needUpdate) {
      if (needUpdate === void 0) {
        needUpdate = true;
      }

      var swiper = this;
      var currentDirection = swiper.params.direction;

      if (!newDirection) {
        // eslint-disable-next-line
        newDirection = currentDirection === 'horizontal' ? 'vertical' : 'horizontal';
      }

      if (newDirection === currentDirection || newDirection !== 'horizontal' && newDirection !==
'vertical') {
        return swiper;
      }

      swiper.$el.removeClass("" + swiper.params.containerModifierClass +
currentDirection).addClass("" + swiper.params.containerModifierClass + newDirection);
      swiper.emitContainerClasses();
      swiper.params.direction = newDirection;
      swiper.slides.each(function (slideEl) {
        if (newDirection === 'vertical') {
          slideEl.style.width = '';
        } else {
          slideEl.style.height = '';
        }
      });
      swiper.emit('changeDirection');
      if (needUpdate) swiper.update();
      return swiper;
    };

    _proto.mount = function mount(el) {
      var swiper = this;
      if (swiper.mounted) return true; // Find el

      var $el = $(el || swiper.params.el);
      el = $el[0];

      if (!el) {
        return false;
      }

      el.swiper = swiper;

      var getWrapperSelector = function getWrapperSelector() {
        return "." + (swiper.params.wrapperClass || '').trim().split(' ').join('.');
      };

      var getWrapper = function getWrapper() {
        if (el && el.shadowRoot && el.shadowRoot.querySelector) {
          var res = $(el.shadowRoot.querySelector(getWrapperSelector())); // Children needs to return
slot items

          res.children = function (options) {
            return $el.children(options);
          };

          return res;
        }

        return $el.children(getWrapperSelector());
      }; // Find Wrapper
      var $wrapperEl = getWrapper();

      if ($wrapperEl.length === 0 && swiper.params.createElements) {
        var document = getDocument();
        var wrapper = document.createElement('div');
        $wrapperEl = $(wrapper);
        wrapper.className = swiper.params.wrapperClass;
        $el.append(wrapper);
        $el.children("." + swiper.params.slideClass).each(function (slideEl) {
          $wrapperEl.append(slideEl);
        });
      }

      extend(swiper, {
        $el: $el,
        el: el,
        $wrapperEl: $wrapperEl,
        wrapperEl: $wrapperEl[0],
        mounted: true,
        // RTL
        rtl: el.dir.toLowerCase() === 'rtl' || $el.css('direction') === 'rtl',
        rtlTranslate: swiper.params.direction === 'horizontal' && (el.dir.toLowerCase() === 'rtl' ||
$el.css('direction') === 'rtl'),
        wrongRTL: $wrapperEl.css('display') === '-webkit-box'
      });
      return true;
    };

    _proto.init = function init(el) {
      var swiper = this;
      if (swiper.initialized) return swiper;
      var mounted = swiper.mount(el);
      if (mounted === false) return swiper;
      swiper.emit('beforeInit'); // Set breakpoint

      if (swiper.params.breakpoints) {
        swiper.setBreakpoint();
      } // Add Classes


      swiper.addClasses(); // Create loop

      if (swiper.params.loop) {
        swiper.loopCreate();
      } // Update size


      swiper.updateSize(); // Update slides

      swiper.updateSlides();

      if (swiper.params.watchOverflow) {
        swiper.checkOverflow();
      } // Set Grab Cursor


      if (swiper.params.grabCursor && swiper.enabled) {
        swiper.setGrabCursor();
      }

      if (swiper.params.preloadImages) {
        swiper.preloadImages();
      } // Slide To Initial Slide
      if (swiper.params.loop) {
        swiper.slideTo(swiper.params.initialSlide + swiper.loopedSlides, 0,
swiper.params.runCallbacksOnInit, false, true);
      } else {
        swiper.slideTo(swiper.params.initialSlide, 0, swiper.params.runCallbacksOnInit, false, true);
      } // Attach events


      swiper.attachEvents(); // Init Flag

      swiper.initialized = true; // Emit

      swiper.emit('init');
      swiper.emit('afterInit');
      return swiper;
    };

    _proto.destroy = function destroy(deleteInstance, cleanStyles) {
      if (deleteInstance === void 0) {
        deleteInstance = true;
      }

      if (cleanStyles === void 0) {
        cleanStyles = true;
      }

      var swiper = this;
      var params = swiper.params,
          $el = swiper.$el,
          $wrapperEl = swiper.$wrapperEl,
          slides = swiper.slides;

      if (typeof swiper.params === 'undefined' || swiper.destroyed) {
        return null;
      }

      swiper.emit('beforeDestroy'); // Init Flag

      swiper.initialized = false; // Detach events

      swiper.detachEvents(); // Destroy loop

      if (params.loop) {
        swiper.loopDestroy();
      } // Cleanup styles


      if (cleanStyles) {
        swiper.removeClasses();
        $el.removeAttr('style');
        $wrapperEl.removeAttr('style');

        if (slides && slides.length) {
          slides.removeClass([params.slideVisibleClass, params.slideActiveClass,
params.slideNextClass, params.slidePrevClass].join(' ')).removeAttr('style').removeAttr('data-swiper-
slide-index');
        }
      }

      swiper.emit('destroy'); // Detach emitter events

      Object.keys(swiper.eventsListeners).forEach(function (eventName) {
        swiper.off(eventName);
      });
```

```
        if (deleteInstance !== false) {
          swiper.$el[0].swiper = null;
          deleteProps(swiper);
        }

        swiper.destroyed = true;
        return null;
      };

      Swiper.extendDefaults = function extendDefaults(newDefaults) {
        extend(extendedDefaults, newDefaults);
      };

      Swiper.installModule = function installModule(module) {
        if (!Swiper.prototype.modules) Swiper.prototype.modules = {};
        var name = module.name || Object.keys(Swiper.prototype.modules).length + "_" + now();
        Swiper.prototype.modules[name] = module;
      };

      Swiper.use = function use(module) {
        if (Array.isArray(module)) {
          module.forEach(function (m) {
            return Swiper.installModule(m);
          });
          return Swiper;
        }

        Swiper.installModule(module);
        return Swiper;
      };

      _createClass(Swiper, null, [{
        key: "extendedDefaults",
        get: function get() {
          return extendedDefaults;
        }
      }, {
        key: "defaults",
        get: function get() {
          return defaults;
        }
      }]);

      return Swiper;
    }();

    Object.keys(prototypes).forEach(function (prototypeGroup) {
      Object.keys(prototypes[prototypeGroup]).forEach(function (protoMethod) {
        Swiper.prototype[protoMethod] = prototypes[prototypeGroup][protoMethod];
      });
    });
    Swiper.use([Resize, Observer$1]);

    var Virtual = {
      update: function update(force) {
        var swiper = this;
        var _swiper$params = swiper.params,
            slidesPerView = _swiper$params.slidesPerView,
            slidesPerGroup = _swiper$params.slidesPerGroup,
            centeredSlides = _swiper$params.centeredSlides;
        var _swiper$params$virtua = swiper.params.virtual,
            addSlidesBefore = _swiper$params$virtua.addSlidesBefore,
            addSlidesAfter = _swiper$params$virtua.addSlidesAfter;
        var _swiper$virtual = swiper.virtual,
            previousFrom = _swiper$virtual.from,
            previousTo = _swiper$virtual.to,
            slides = _swiper$virtual.slides,
            previousSlidesGrid = _swiper$virtual.slidesGrid,
            renderSlide = _swiper$virtual.renderSlide,
            previousOffset = _swiper$virtual.offset;
        swiper.updateActiveIndex();
        var activeIndex = swiper.activeIndex || 0;
        var offsetProp;
        if (swiper.rtlTranslate) offsetProp = 'right';else offsetProp = swiper.isHorizontal() ? 'left'
  : 'top';
        var slidesAfter;
        var slidesBefore;

        if (centeredSlides) {
          slidesAfter = Math.floor(slidesPerView / 2) + slidesPerGroup + addSlidesAfter;
          slidesBefore = Math.floor(slidesPerView / 2) + slidesPerGroup + addSlidesBefore;
        } else {
          slidesAfter = slidesPerView + (slidesPerGroup - 1) + addSlidesAfter;
          slidesBefore = slidesPerGroup + addSlidesBefore;
        }

        var from = Math.max((activeIndex || 0) - slidesBefore, 0);
        var to = Math.min((activeIndex || 0) + slidesAfter, slides.length - 1);
        var offset = (swiper.slidesGrid[from] || 0) - (swiper.slidesGrid[0] || 0);
        extend(swiper.virtual, {
          from: from,
          to: to,
          offset: offset,
          slidesGrid: swiper.slidesGrid
        });

        function onRendered() {
          swiper.updateSlides();
          swiper.updateProgress();
          swiper.updateSlidesClasses();

          if (swiper.lazy && swiper.params.lazy.enabled) {
            swiper.lazy.load();
          }
        }

        if (previousFrom === from && previousTo === to && !force) {
          if (swiper.slidesGrid !== previousSlidesGrid && offset !== previousOffset) {
            swiper.slides.css(offsetProp, offset + "px");
          }

          swiper.updateProgress();
          return;
        }

        if (swiper.params.virtual.renderExternal) {
          swiper.params.virtual.renderExternal.call(swiper, {
            offset: offset,
            from: from,
            to: to,
            slides: function getSlides() {
              var slidesToRender = [];

              for (var i = from; i <= to; i += 1) {
                slidesToRender.push(slides[i]);
              }

              return slidesToRender;
            }()
          });
          if (swiper.params.virtual.renderExternalUpdate) {
            onRendered();
          }

          return;
        }

        var prependIndexes = [];
        var appendIndexes = [];

        if (force) {
          swiper.$wrapperEl.find("." + swiper.params.slideClass).remove();
        } else {
          for (var i = previousFrom; i <= previousTo; i += 1) {
            if (i < from || i > to) {
              swiper.$wrapperEl.find("." + swiper.params.slideClass + "[data-swiper-slide-index=\"" + i
  + "\"]").remove();
            }
          }
        }

        for (var _i = 0; _i < slides.length; _i += 1) {
          if (_i >= from && _i <= to) {
            if (typeof previousTo === 'undefined' || force) {
              appendIndexes.push(_i);
            } else {
              if (_i > previousTo) appendIndexes.push(_i);
              if (_i < previousFrom) prependIndexes.push(_i);
            }
          }
        }

        appendIndexes.forEach(function (index) {
          swiper.$wrapperEl.append(renderSlide(slides[index], index));
        });
        prependIndexes.sort(function (a, b) {
          return b - a;
        }).forEach(function (index) {
          swiper.$wrapperEl.prepend(renderSlide(slides[index], index));
        });
        swiper.$wrapperEl.children('.swiper-slide').css(offsetProp, offset + "px");
        onRendered();
      },
      renderSlide: function renderSlide(slide, index) {
        var swiper = this;
        var params = swiper.params.virtual;

        if (params.cache && swiper.virtual.cache[index]) {
          return swiper.virtual.cache[index];
        }

        var $slideEl = params.renderSlide ? $(params.renderSlide.call(swiper, slide, index)) : $("<div
  class=\"" + swiper.params.slideClass + "\" data-swiper-slide-index=\"" + index + "\">" + slide + "
  </div>");
        if (!$slideEl.attr('data-swiper-slide-index')) $slideEl.attr('data-swiper-slide-index', index);
        if (params.cache) swiper.virtual.cache[index] = $slideEl;
        return $slideEl;
      },
      appendSlide: function appendSlide(slides) {
        var swiper = this;

        if (typeof slides === 'object' && 'length' in slides) {
          for (var i = 0; i < slides.length; i += 1) {
            if (slides[i]) swiper.virtual.slides.push(slides[i]);
          }
        } else {
          swiper.virtual.slides.push(slides);
        }

        swiper.virtual.update(true);
      },
      prependSlide: function prependSlide(slides) {
        var swiper = this;
        var activeIndex = swiper.activeIndex;
        var newActiveIndex = activeIndex + 1;
        var numberOfNewSlides = 1;

        if (Array.isArray(slides)) {
          for (var i = 0; i < slides.length; i += 1) {
            if (slides[i]) swiper.virtual.slides.unshift(slides[i]);
          }

          newActiveIndex = activeIndex + slides.length;
          numberOfNewSlides = slides.length;
        } else {
          swiper.virtual.slides.unshift(slides);
        }

        if (swiper.params.virtual.cache) {
          var cache = swiper.virtual.cache;
          var newCache = {};
          Object.keys(cache).forEach(function (cachedIndex) {
            var $cachedEl = cache[cachedIndex];
            var cachedElIndex = $cachedEl.attr('data-swiper-slide-index');

            if (cachedElIndex) {
              $cachedEl.attr('data-swiper-slide-index', parseInt(cachedElIndex, 10) + 1);
            }

            newCache[parseInt(cachedIndex, 10) + numberOfNewSlides] = $cachedEl;
          });
          swiper.virtual.cache = newCache;
        }

        swiper.virtual.update(true);
        swiper.slideTo(newActiveIndex, 0);
      },
      removeSlide: function removeSlide(slidesIndexes) {
        var swiper = this;
        if (typeof slidesIndexes === 'undefined' || slidesIndexes === null) return;
        var activeIndex = swiper.activeIndex;

        if (Array.isArray(slidesIndexes)) {
          for (var i = slidesIndexes.length - 1; i >= 0; i -= 1) {
            swiper.virtual.slides.splice(slidesIndexes[i], 1);

            if (swiper.params.virtual.cache) {
              delete swiper.virtual.cache[slidesIndexes[i]];
            }

            if (slidesIndexes[i] < activeIndex) activeIndex -= 1;
            activeIndex = Math.max(activeIndex, 0);
          }
        } else {
          swiper.virtual.slides.splice(slidesIndexes, 1);

          if (swiper.params.virtual.cache) {
            delete swiper.virtual.cache[slidesIndexes];
          }
          if (slidesIndexes < activeIndex) activeIndex -= 1;
          activeIndex = Math.max(activeIndex, 0);
        }

        swiper.virtual.update(true);
        swiper.slideTo(activeIndex, 0);
      },
      removeAllSlides: function removeAllSlides() {
        var swiper = this;
        swiper.virtual.slides = [];

        if (swiper.params.virtual.cache) {
          swiper.virtual.cache = {};
        }

        swiper.virtual.update(true);
        swiper.slideTo(0, 0);
      }
    };
    var Virtual$1 = {
      name: 'virtual',
      params: {
        virtual: {
          enabled: false,
          slides: [],
          cache: true,
          renderSlide: null,
          renderExternal: null,
          renderExternalUpdate: true,
          addSlidesBefore: 0,
          addSlidesAfter: 0
        }
      },
      create: function create() {
        var swiper = this;
        bindModuleMethods(swiper, {
          virtual: _extends({}, Virtual, {
            slides: swiper.params.virtual.slides,
            cache: {}
          })
        });
      },
      on: {
        beforeInit: function beforeInit(swiper) {
          if (!swiper.params.virtual.enabled) return;
          swiper.classNames.push(swiper.params.containerModifierClass + "virtual");
          var overwriteParams = {
            watchSlidesProgress: true
          };
          extend(swiper.params, overwriteParams);
          extend(swiper.originalParams, overwriteParams);

          if (!swiper.params.initialSlide) {
            swiper.virtual.update();
          }
        },
        setTranslate: function setTranslate(swiper) {
          if (!swiper.params.virtual.enabled) return;
          swiper.virtual.update();
        }
      }
    };

    var Keyboard = {
      handle: function handle(event) {
        var swiper = this;
        if (!swiper.enabled) return;
        var window = getWindow();
        var document = getDocument();
        var rtl = swiper.rtlTranslate;
        var e = event;
        if (e.originalEvent) e = e.originalEvent; // jquery fix

        var kc = e.keyCode || e.charCode;
        var pageUpDown = swiper.params.keyboard.pageUpDown;
        var isPageUp = pageUpDown && kc === 33;
        var isPageDown = pageUpDown && kc === 34;
        var isArrowLeft = kc === 37;
        var isArrowRight = kc === 39;
        var isArrowUp = kc === 38;
        var isArrowDown = kc === 40; // Directions locks

        if (!swiper.allowSlideNext && (swiper.isHorizontal() && isArrowRight || swiper.isVertical() &&
  isArrowDown || isPageDown)) {
          return false;
        }

        if (!swiper.allowSlidePrev && (swiper.isHorizontal() && isArrowLeft || swiper.isVertical() &&
  isArrowUp || isPageUp)) {
          return false;
        }

        if (e.shiftKey || e.altKey || e.ctrlKey || e.metaKey) {
          return undefined;
        }

        if (document.activeElement && document.activeElement.nodeName &&
  (document.activeElement.nodeName.toLowerCase() === 'input' ||
  document.activeElement.nodeName.toLowerCase() === 'textarea')) {
          return undefined;
        }

        if (swiper.params.keyboard.onlyInViewport && (isPageUp || isPageDown || isArrowLeft ||
  isArrowRight || isArrowUp || isArrowDown)) {
          var inView = false; // Check that swiper should be inside of visible area of window

          if (swiper.$el.parents("." + swiper.params.slideClass).length > 0 && swiper.$el.parents("." +
  swiper.params.slideActiveClass).length === 0) {
            return undefined;
          }

          var $el = swiper.$el;
          var swiperWidth = $el[0].clientWidth;
          var swiperHeight = $el[0].clientHeight;
          var windowWidth = window.innerWidth;
          var windowHeight = window.innerHeight;
          var swiperOffset = swiper.$el.offset();
          if (rtl) swiperOffset.left -= swiper.$el[0].scrollLeft;
          var swiperCoord = [[swiperOffset.left, swiperOffset.top], [swiperOffset.left + swiperWidth,
  swiperOffset.top], [swiperOffset.left, swiperOffset.top + swiperHeight], [swiperOffset.left +
  swiperWidth, swiperOffset.top + swiperHeight]];

          for (var i = 0; i < swiperCoord.length; i += 1) {
            var point = swiperCoord[i];

            if (point[0] >= 0 && point[0] <= windowWidth && point[1] >= 0 && point[1] <= windowHeight)
  {
              if (point[0] === 0 && point[1] === 0) continue; // eslint-disable-line
              inView = true;
            }
          }

          if (!inView) return undefined;
        }

        if (swiper.isHorizontal()) {
          if (isPageUp || isPageDown || isArrowLeft || isArrowRight) {
            if (e.preventDefault) e.preventDefault();else e.returnValue = false;
          }

          if ((isPageDown || isArrowRight) && !rtl || (isPageUp || isArrowLeft) && rtl)
  swiper.slideNext();
          if ((isPageUp || isArrowLeft) && !rtl || (isPageDown || isArrowRight) && rtl)
  swiper.slidePrev();
        } else {
          if (isPageUp || isPageDown || isArrowUp || isArrowDown) {
            if (e.preventDefault) e.preventDefault();else e.returnValue = false;
          }

          if (isPageDown || isArrowDown) swiper.slideNext();
          if (isPageUp || isArrowUp) swiper.slidePrev();
        }

        swiper.emit('keyPress', kc);
        return undefined;
      },
      enable: function enable() {
        var swiper = this;
        var document = getDocument();
        if (swiper.keyboard.enabled) return;
        $(document).on('keydown', swiper.keyboard.handle);
        swiper.keyboard.enabled = true;
      },
      disable: function disable() {
        var swiper = this;
        var document = getDocument();
        if (!swiper.keyboard.enabled) return;
        $(document).off('keydown', swiper.keyboard.handle);
        swiper.keyboard.enabled = false;
      }
    };
    var Keyboard$1 = {
      name: 'keyboard',
      params: {
        keyboard: {
          enabled: false,
          onlyInViewport: true,
          pageUpDown: true
        }
      },
      create: function create() {
        var swiper = this;
        bindModuleMethods(swiper, {
          keyboard: _extends({
            enabled: false
          }, Keyboard)
        });
      },
      on: {
        init: function init(swiper) {
          if (swiper.params.keyboard.enabled) {
            swiper.keyboard.enable();
          }
        },
        destroy: function destroy(swiper) {
          if (swiper.keyboard.enabled) {
            swiper.keyboard.disable();
          }
        }
      }
    };

    /* eslint-disable consistent-return */

    function isEventSupported() {
      var document = getDocument();
      var eventName = 'onwheel';
      var isSupported = (eventName in document);

      if (!isSupported) {
        var element = document.createElement('div');
        element.setAttribute(eventName, 'return;');
        isSupported = typeof element[eventName] === 'function';
      }

      if (!isSupported && document.implementation && document.implementation.hasFeature && // always
  returns true in newer browsers as per the standard.
      // @see http://dom.spec.whatwg.org/#dom-domimplementation-hasfeature
      document.implementation.hasFeature('', '') !== true) {
        // This is the only way to test support for the `wheel` event in IE9+.
        isSupported = document.implementation.hasFeature('Events.wheel', '3.0');
      }

      return isSupported;
    }

    var Mousewheel = {
      lastScrollTime: now(),
      lastEventBeforeSnap: undefined,
      recentWheelEvents: [],
      event: function event() {
        var window = getWindow();
        if (window.navigator.userAgent.indexOf('firefox') > -1) return 'DOMMouseScroll';
        return isEventSupported() ? 'wheel' : 'mousewheel';
      },
      normalize: function normalize(e) {
        // Reasonable defaults
        var PIXEL_STEP = 10;
        var LINE_HEIGHT = 40;
        var PAGE_HEIGHT = 800;
        var sX = 0;
        var sY = 0; // spinX, spinY

        var pX = 0;
        var pY = 0; // pixelX, pixelY
        // Legacy

        if ('detail' in e) {
          sY = e.detail;
        }

        if ('wheelDelta' in e) {
          sY = -e.wheelDelta / 120;
        }

        if ('wheelDeltaY' in e) {
          sY = -e.wheelDeltaY / 120;
        }
```

```
        if ('wheelDeltaX' in e) {
          sX = -e.wheelDeltaX / 120;
        } // side scrolling on FF with DOMMouseScroll


        if ('axis' in e && e.axis === e.HORIZONTAL_AXIS) {
          sX = sY;
          sY = 0;
        }

        pX = sX * PIXEL_STEP;
        pY = sY * PIXEL_STEP;

        if ('deltaY' in e) {
          pY = e.deltaY;
        }

        if ('deltaX' in e) {
          pX = e.deltaX;
        }

        if (e.shiftKey && !pX) {
          // if user scrolls with shift he wants horizontal scroll
          pX = pY;
          pY = 0;
        }

        if ((pX || pY) && e.deltaMode) {
          if (e.deltaMode === 1) {
            // delta in LINE units
            pX *= LINE_HEIGHT;
            pY *= LINE_HEIGHT;
          } else {
            // delta in PAGE units
            pX *= PAGE_HEIGHT;
            pY *= PAGE_HEIGHT;
          }
        } // Fall-back if spin cannot be determined


        if (pX && !sX) {
          sX = pX < 1 ? -1 : 1;
        }

        if (pY && !sY) {
          sY = pY < 1 ? -1 : 1;
        }

        return {
          spinX: sX,
          spinY: sY,
          pixelX: pX,
          pixelY: pY
        };
      },
      handleMouseEnter: function handleMouseEnter() {
        var swiper = this;
        if (!swiper.enabled) return;
        swiper.mouseEntered = true;
      },
      handleMouseLeave: function handleMouseLeave() {
        var swiper = this;
        if (!swiper.enabled) return;
        swiper.mouseEntered = false;
```

```
    },
    handle: function handle(event) {
      var e = event;
      var disableParentSwiper = true;
      var swiper = this;
      if (!swiper.enabled) return;
      var params = swiper.params.mousewheel;

      if (swiper.params.cssMode) {
        e.preventDefault();
      }

      var target = swiper.$el;

      if (swiper.params.mousewheel.eventsTarget !== 'container') {
        target = $(swiper.params.mousewheel.eventsTarget);
      }

      if (!swiper.mouseEntered && !target[0].contains(e.target) && !params.releaseOnEdges) return
true;
      if (e.originalEvent) e = e.originalEvent; // jquery fix

      var delta = 0;
      var rtlFactor = swiper.rtlTranslate ? -1 : 1;
      var data = Mousewheel.normalize(e);

      if (params.forceToAxis) {
        if (swiper.isHorizontal()) {
          if (Math.abs(data.pixelX) > Math.abs(data.pixelY)) delta = -data.pixelX * rtlFactor;else
return true;
        } else if (Math.abs(data.pixelY) > Math.abs(data.pixelX)) delta = -data.pixelY;else return
true;
      } else {
        delta = Math.abs(data.pixelX) > Math.abs(data.pixelY) ? -data.pixelX * rtlFactor : -
data.pixelY;
      }

      if (delta === 0) return true;
      if (params.invert) delta = -delta; // Get the scroll positions

      var positions = swiper.getTranslate() + delta * params.sensitivity;
      if (positions >= swiper.minTranslate()) positions = swiper.minTranslate();
      if (positions <= swiper.maxTranslate()) positions = swiper.maxTranslate(); // When loop is
true:
      //     the disableParentSwiper will be true.
      // When loop is false:
      //     if the scroll positions is not on edge,
      //     then the disableParentSwiper will be true.
      //     if the scroll on edge positions,
      //     then the disableParentSwiper will be false.

      disableParentSwiper = swiper.params.loop ? true : !(positions === swiper.minTranslate() ||
positions === swiper.maxTranslate());
      if (disableParentSwiper && swiper.params.nested) e.stopPropagation();

      if (!swiper.params.freeMode) {
        // Register the new event in a variable which stores the relevant data
        var newEvent = {
          time: now(),
          delta: Math.abs(delta),
          direction: Math.sign(delta),
          raw: event
        }; // Keep the most recent events

        var recentWheelEvents = swiper.mousewheel.recentWheelEvents;
        if (recentWheelEvents.length >= 2) {
          recentWheelEvents.shift(); // only store the last N events
        }

        var prevEvent = recentWheelEvents.length ? recentWheelEvents[recentWheelEvents.length - 1] :
undefined;
        recentWheelEvents.push(newEvent); // If there is at least one previous recorded event:
        //   If direction has changed or
        //   if the scroll is quicker than the previous one:
        //     Animate the slider.
        // Else (this is the first time the wheel is moved):
        //     Animate the slider.

        if (prevEvent) {
          if (newEvent.direction !== prevEvent.direction || newEvent.delta > prevEvent.delta ||
newEvent.time > prevEvent.time + 150) {
            swiper.mousewheel.animateSlider(newEvent);
          }
        } else {
          swiper.mousewheel.animateSlider(newEvent);
        } // If it's time to release the scroll:
        //   Return now so you don't hit the preventDefault.


        if (swiper.mousewheel.releaseScroll(newEvent)) {
          return true;
        }
      } else {
        // Freemode or scrollContainer:
        // If we recently snapped after a momentum scroll, then ignore wheel events
        // to give time for the deceleration to finish. Stop ignoring after 500 msecs
        // or if it's a new scroll (larger delta or inverse sign as last event before
        // an end-of-momentum snap).
        var _newEvent = {
          time: now(),
          delta: Math.abs(delta),
          direction: Math.sign(delta)
        };
        var lastEventBeforeSnap = swiper.mousewheel.lastEventBeforeSnap;
        var ignoreWheelEvents = lastEventBeforeSnap && _newEvent.time < lastEventBeforeSnap.time +
500 && _newEvent.delta <= lastEventBeforeSnap.delta && _newEvent.direction ===
lastEventBeforeSnap.direction;

        if (!ignoreWheelEvents) {
          swiper.mousewheel.lastEventBeforeSnap = undefined;

          if (swiper.params.loop) {
            swiper.loopFix();
          }

          var position = swiper.getTranslate() + delta * params.sensitivity;
          var wasBeginning = swiper.isBeginning;
          var wasEnd = swiper.isEnd;
          if (position >= swiper.minTranslate()) position = swiper.minTranslate();
          if (position <= swiper.maxTranslate()) position = swiper.maxTranslate();
          swiper.setTransition(0);
          swiper.setTranslate(position);
          swiper.updateProgress();
          swiper.updateActiveIndex();
          swiper.updateSlidesClasses();

          if (!wasBeginning && swiper.isBeginning || !wasEnd && swiper.isEnd) {
            swiper.updateSlidesClasses();
          }
          if (swiper.params.freeModeSticky) {
            // When wheel scrolling starts with sticky (aka snap) enabled, then detect
            // the end of a momentum scroll by storing recent (N=15?) wheel events.
            // 1. do all N events have decreasing or same (absolute value) delta?
            // 2. did all N events arrive in the last M (M=500?) msecs?
            // 3. does the earliest event have an (absolute value) delta that's
            //    at least P (P=1?) larger than the most recent event's delta?
            // 4. does the latest event have a delta that's smaller than Q (Q=6?) pixels?
            // If 1-4 are "yes" then we're near the end of a momentum scroll deceleration.
            // Snap immediately and ignore remaining wheel events in this scroll.
            // See comment above for "remaining wheel events in this scroll" determination.
            // If 1-4 aren't satisfied, then wait to snap until 500ms after the last event.
            clearTimeout(swiper.mousewheel.timeout);
            swiper.mousewheel.timeout = undefined;
            var _recentWheelEvents = swiper.mousewheel.recentWheelEvents;

            if (_recentWheelEvents.length >= 15) {
              _recentWheelEvents.shift(); // only store the last N events

            }

            var _prevEvent = _recentWheelEvents.length ? _recentWheelEvents[_recentWheelEvents.length
- 1] : undefined;

            var firstEvent = _recentWheelEvents[0];

            _recentWheelEvents.push(_newEvent);

            if (_prevEvent && (_newEvent.delta > _prevEvent.delta || _newEvent.direction !==
_prevEvent.direction)) {
              // Increasing or reverse-sign delta means the user started scrolling again. Clear the
wheel event log.
              _recentWheelEvents.splice(0);
            } else if (_recentWheelEvents.length >= 15 && _newEvent.time - firstEvent.time < 500 &&
firstEvent.delta - _newEvent.delta >= 1 && _newEvent.delta <= 6) {
              // We're at the end of the deceleration of a momentum scroll, so there's no need
              // to wait for more events. Snap ASAP on the next tick.
              // Also, because there's some remaining momentum we'll bias the snap in the
              // direction of the ongoing scroll because it's better UX for the scroll to snap
              // in the same direction as the scroll instead of reversing to snap.  Therefore,
              // if it's already scrolled more than 20% in the current direction, keep going.
              var snapToThreshold = delta > 0 ? 0.8 : 0.2;
              swiper.mousewheel.lastEventBeforeSnap = _newEvent;

              _recentWheelEvents.splice(0);

              swiper.mousewheel.timeout = nextTick(function () {
                swiper.slideToClosest(swiper.params.speed, true, undefined, snapToThreshold);
              }, 0); // no delay; move on next tick
            }

            if (!swiper.mousewheel.timeout) {
              // if we get here, then we haven't detected the end of a momentum scroll, so
              // we'll consider a scroll "complete" when there haven't been any wheel events
              // for 500ms.
              swiper.mousewheel.timeout = nextTick(function () {
                var snapToThreshold = 0.5;
                swiper.mousewheel.lastEventBeforeSnap = _newEvent;

                _recentWheelEvents.splice(0);

                swiper.slideToClosest(swiper.params.speed, true, undefined, snapToThreshold);
              }, 500);
            }
          } // Emit event


          if (!ignoreWheelEvents) swiper.emit('scroll', e); // Stop autoplay

          if (swiper.params.autoplay && swiper.params.autoplayDisableOnInteraction)
swiper.autoplay.stop(); // Return page scroll on edge positions

          if (position === swiper.minTranslate() || position === swiper.maxTranslate()) return true;
        }
      }

      if (e.preventDefault) e.preventDefault();else e.returnValue = false;
      return false;
    },
    animateSlider: function animateSlider(newEvent) {
      var swiper = this;
      var window = getWindow();

      if (this.params.mousewheel.thresholdDelta && newEvent.delta <
this.params.mousewheel.thresholdDelta) {
        // Prevent if delta of wheel scroll delta is below configured threshold
        return false;
      }

      if (this.params.mousewheel.thresholdTime && now() - swiper.mousewheel.lastScrollTime <
this.params.mousewheel.thresholdTime) {
        // Prevent if time between scrolls is below configured threshold
        return false;
      } // If the movement is NOT big enough and
      // if the last time the user scrolled was too close to the current one (avoid continuously
triggering the slider):
      //   Don't go any further (avoid insignificant scroll movement).


      if (newEvent.delta >= 6 && now() - swiper.mousewheel.lastScrollTime < 60) {
        // Return false as a default
        return true;
      } // If user is scrolling towards the end:
      //   If the slider hasn't hit the latest slide or
      //   if the slider is a loop and
      //   if the slider isn't moving right now:
      //     Go to next slide and
      //     emit a scroll event.
      // Else (the user is scrolling towards the beginning) and
      // if the slider hasn't hit the first slide or
      // if the slider is a loop and
      // if the slider isn't moving right now:
      //   Go to prev slide and
      //   emit a scroll event.


      if (newEvent.direction < 0) {
        if ((!swiper.isEnd || swiper.params.loop) && !swiper.animating) {
          swiper.slideNext();
          swiper.emit('scroll', newEvent.raw);
        }
      } else if ((!swiper.isBeginning || swiper.params.loop) && !swiper.animating) {
        swiper.slidePrev();
        swiper.emit('scroll', newEvent.raw);
      } // If you got here is because an animation has been triggered so store the current time


      swiper.mousewheel.lastScrollTime = new window.Date().getTime(); // Return false as a default
```

```
        return false;
      },
      releaseScroll: function releaseScroll(newEvent) {
        var swiper = this;
        var params = swiper.params.mousewheel;

        if (newEvent.direction < 0) {
          if (swiper.isEnd && !swiper.params.loop && params.releaseOnEdges) {
            // Return true to animate scroll on edges
            return true;
          }
        } else if (swiper.isBeginning && !swiper.params.loop && params.releaseOnEdges) {
          // Return true to animate scroll on edges
          return true;
        }

        return false;
      },
      enable: function enable() {
        var swiper = this;
        var event = Mousewheel.event();

        if (swiper.params.cssMode) {
          swiper.wrapperEl.removeEventListener(event, swiper.mousewheel.handle);
          return true;
        }

        if (!event) return false;
        if (swiper.mousewheel.enabled) return false;
        var target = swiper.$el;

        if (swiper.params.mousewheel.eventsTarget !== 'container') {
          target = $(swiper.params.mousewheel.eventsTarget);
        }

        target.on('mouseenter', swiper.mousewheel.handleMouseEnter);
        target.on('mouseleave', swiper.mousewheel.handleMouseLeave);
        target.on(event, swiper.mousewheel.handle);
        swiper.mousewheel.enabled = true;
        return true;
      },
      disable: function disable() {
        var swiper = this;
        var event = Mousewheel.event();

        if (swiper.params.cssMode) {
          swiper.wrapperEl.addEventListener(event, swiper.mousewheel.handle);
          return true;
        }

        if (!event) return false;
        if (!swiper.mousewheel.enabled) return false;
        var target = swiper.$el;

        if (swiper.params.mousewheel.eventsTarget !== 'container') {
          target = $(swiper.params.mousewheel.eventsTarget);
        }

        target.off(event, swiper.mousewheel.handle);
        swiper.mousewheel.enabled = false;
        return true;
      }
    };
    var Mousewheel$1 = {
      name: 'mousewheel',
      params: {
        mousewheel: {
          enabled: false,
          releaseOnEdges: false,
          invert: false,
          forceToAxis: false,
          sensitivity: 1,
          eventsTarget: 'container',
          thresholdDelta: null,
          thresholdTime: null
        }
      },
      create: function create() {
        var swiper = this;
        bindModuleMethods(swiper, {
          mousewheel: {
            enabled: false,
            lastScrollTime: now(),
            lastEventBeforeSnap: undefined,
            recentWheelEvents: [],
            enable: Mousewheel.enable,
            disable: Mousewheel.disable,
            handle: Mousewheel.handle,
            handleMouseEnter: Mousewheel.handleMouseEnter,
            handleMouseLeave: Mousewheel.handleMouseLeave,
            animateSlider: Mousewheel.animateSlider,
            releaseScroll: Mousewheel.releaseScroll
          }
        });
      },
      on: {
        init: function init(swiper) {
          if (!swiper.params.mousewheel.enabled && swiper.params.cssMode) {
            swiper.mousewheel.disable();
          }

          if (swiper.params.mousewheel.enabled) swiper.mousewheel.enable();
        },
        destroy: function destroy(swiper) {
          if (swiper.params.cssMode) {
            swiper.mousewheel.enable();
          }

          if (swiper.mousewheel.enabled) swiper.mousewheel.disable();
        }
      }
    };

    var Navigation = {
      toggleEl: function toggleEl($el, disabled) {
        $el[disabled ? 'addClass' : 'removeClass'](this.params.navigation.disabledClass);
        if ($el[0] && $el[0].tagName === 'BUTTON') $el[0].disabled = disabled;
      },
      update: function update() {
        // Update Navigation Buttons
        var swiper = this;
        var params = swiper.params.navigation;
        var toggleEl = swiper.navigation.toggleEl;
        if (swiper.params.loop) return;
        var _swiper$navigation = swiper.navigation,
            $nextEl = _swiper$navigation.$nextEl,
            $prevEl = _swiper$navigation.$prevEl;

        if ($prevEl && $prevEl.length > 0) {
          if (swiper.isBeginning) {
```

```
          toggleEl($prevEl, true);
        } else {
          toggleEl($prevEl, false);
        }

        if (swiper.params.watchOverflow && swiper.enabled) {
          $prevEl[swiper.isLocked ? 'addClass' : 'removeClass'](params.lockClass);
        }
      }

      if ($nextEl && $nextEl.length > 0) {
        if (swiper.isEnd) {
          toggleEl($nextEl, true);
        } else {
          toggleEl($nextEl, false);
        }

        if (swiper.params.watchOverflow && swiper.enabled) {
          $nextEl[swiper.isLocked ? 'addClass' : 'removeClass'](params.lockClass);
        }
      }
    },
    onPrevClick: function onPrevClick(e) {
      var swiper = this;
      e.preventDefault();
      if (swiper.isBeginning && !swiper.params.loop) return;
      swiper.slidePrev();
    },
    onNextClick: function onNextClick(e) {
      var swiper = this;
      e.preventDefault();
      if (swiper.isEnd && !swiper.params.loop) return;
      swiper.slideNext();
    },
    init: function init() {
      var swiper = this;
      var params = swiper.params.navigation;
      swiper.params.navigation = createElementIfNotDefined(swiper.$el, swiper.params.navigation,
swiper.params.createElements, {
        nextEl: 'swiper-button-next',
        prevEl: 'swiper-button-prev'
      });
      if (!(params.nextEl || params.prevEl)) return;
      var $nextEl;
      var $prevEl;

      if (params.nextEl) {
        $nextEl = $(params.nextEl);

        if (swiper.params.uniqueNavElements && typeof params.nextEl === 'string' && $nextEl.length >
1 && swiper.$el.find(params.nextEl).length === 1) {
          $nextEl = swiper.$el.find(params.nextEl);
        }
      }

      if (params.prevEl) {
        $prevEl = $(params.prevEl);

        if (swiper.params.uniqueNavElements && typeof params.prevEl === 'string' && $prevEl.length >
1 && swiper.$el.find(params.prevEl).length === 1) {
          $prevEl = swiper.$el.find(params.prevEl);
        }
      }

      if ($nextEl && $nextEl.length > 0) {
        $nextEl.on('click', swiper.navigation.onNextClick);
      }

      if ($prevEl && $prevEl.length > 0) {
        $prevEl.on('click', swiper.navigation.onPrevClick);
      }

      extend(swiper.navigation, {
        $nextEl: $nextEl,
        nextEl: $nextEl && $nextEl[0],
        $prevEl: $prevEl,
        prevEl: $prevEl && $prevEl[0]
      });

      if (!swiper.enabled) {
        if ($nextEl) $nextEl.addClass(params.lockClass);
        if ($prevEl) $prevEl.addClass(params.lockClass);
      }
    },
    destroy: function destroy() {
      var swiper = this;
      var _swiper$navigation2 = swiper.navigation,
          $nextEl = _swiper$navigation2.$nextEl,
          $prevEl = _swiper$navigation2.$prevEl;

      if ($nextEl && $nextEl.length) {
        $nextEl.off('click', swiper.navigation.onNextClick);
        $nextEl.removeClass(swiper.params.navigation.disabledClass);
      }

      if ($prevEl && $prevEl.length) {
        $prevEl.off('click', swiper.navigation.onPrevClick);
        $prevEl.removeClass(swiper.params.navigation.disabledClass);
      }
    }
  };
  var Navigation$1 = {
    name: 'navigation',
    params: {
      navigation: {
        nextEl: null,
        prevEl: null,
        hideOnClick: false,
        disabledClass: 'swiper-button-disabled',
        hiddenClass: 'swiper-button-hidden',
        lockClass: 'swiper-button-lock'
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        navigation: _extends({}, Navigation)
      });
    },
    on: {
      init: function init(swiper) {
        swiper.navigation.init();
        swiper.navigation.update();
      },
      toEdge: function toEdge(swiper) {
        swiper.navigation.update();
      },
      fromEdge: function fromEdge(swiper) {
        swiper.navigation.update();
      },
      destroy: function destroy(swiper) {
        swiper.navigation.destroy();
      },
      'enable disable': function enableDisable(swiper) {
        var _swiper$navigation3 = swiper.navigation,
            $nextEl = _swiper$navigation3.$nextEl,
            $prevEl = _swiper$navigation3.$prevEl;

        if ($nextEl) {
          $nextEl[swiper.enabled ? 'removeClass' : 'addClass'](swiper.params.navigation.lockClass);
        }

        if ($prevEl) {
          $prevEl[swiper.enabled ? 'removeClass' : 'addClass'](swiper.params.navigation.lockClass);
        }
      },
      click: function click(swiper, e) {
        var _swiper$navigation4 = swiper.navigation,
            $nextEl = _swiper$navigation4.$nextEl,
            $prevEl = _swiper$navigation4.$prevEl;
        var targetEl = e.target;

        if (swiper.params.navigation.hideOnClick && !$(targetEl).is($prevEl) &&
!$(targetEl).is($nextEl)) {
          if (swiper.pagination && swiper.params.pagination && swiper.params.pagination.clickable &&
(swiper.pagination.el === targetEl || swiper.pagination.el.contains(targetEl))) return;
          var isHidden;

          if ($nextEl) {
            isHidden = $nextEl.hasClass(swiper.params.navigation.hiddenClass);
          } else if ($prevEl) {
            isHidden = $prevEl.hasClass(swiper.params.navigation.hiddenClass);
          }

          if (isHidden === true) {
            swiper.emit('navigationShow');
          } else {
            swiper.emit('navigationHide');
          }

          if ($nextEl) {
            $nextEl.toggleClass(swiper.params.navigation.hiddenClass);
          }

          if ($prevEl) {
            $prevEl.toggleClass(swiper.params.navigation.hiddenClass);
          }
        }
      }
    }
  };

  var Pagination = {
    update: function update() {
      // Render || Update Pagination bullets/items
      var swiper = this;
      var rtl = swiper.rtl;
      var params = swiper.params.pagination;
      if (!params.el || !swiper.pagination.el || !swiper.pagination.$el ||
swiper.pagination.$el.length === 0) return;
      var slidesLength = swiper.virtual && swiper.params.virtual.enabled ?
swiper.virtual.slides.length : swiper.slides.length;
      var $el = swiper.pagination.$el; // Current/Total

      var current;
      var total = swiper.params.loop ? Math.ceil((slidesLength - swiper.loopedSlides * 2) /
swiper.params.slidesPerGroup) : swiper.snapGrid.length;

      if (swiper.params.loop) {
        current = Math.ceil((swiper.activeIndex - swiper.loopedSlides) /
swiper.params.slidesPerGroup);

        if (current > slidesLength - 1 - swiper.loopedSlides * 2) {
          current -= slidesLength - swiper.loopedSlides * 2;
        }

        if (current > total - 1) current -= total;
        if (current < 0 && swiper.params.paginationType !== 'bullets') current = total + current;
      } else if (typeof swiper.snapIndex !== 'undefined') {
        current = swiper.snapIndex;
      } else {
        current = swiper.activeIndex || 0;
      } // Types


      if (params.type === 'bullets' && swiper.pagination.bullets && swiper.pagination.bullets.length
> 0) {
        var bullets = swiper.pagination.bullets;
        var firstIndex;
        var lastIndex;
        var midIndex;

        if (params.dynamicBullets) {
          swiper.pagination.bulletSize = bullets.eq(0)[swiper.isHorizontal() ? 'outerWidth' :
'outerHeight'](true);
          $el.css(swiper.isHorizontal() ? 'width' : 'height', swiper.pagination.bulletSize *
(params.dynamicMainBullets + 4) + "px");

          if (params.dynamicMainBullets > 1 && swiper.previousIndex !== undefined) {
            swiper.pagination.dynamicBulletIndex += current - swiper.previousIndex;

            if (swiper.pagination.dynamicBulletIndex > params.dynamicMainBullets - 1) {
              swiper.pagination.dynamicBulletIndex = params.dynamicMainBullets - 1;
            } else if (swiper.pagination.dynamicBulletIndex < 0) {
              swiper.pagination.dynamicBulletIndex = 0;
            }
          }

          firstIndex = current - swiper.pagination.dynamicBulletIndex;
          lastIndex = firstIndex + (Math.min(bullets.length, params.dynamicMainBullets) - 1);
          midIndex = (lastIndex + firstIndex) / 2;
        }

        bullets.removeClass(params.bulletActiveClass + " " + params.bulletActiveClass + "-next " +
params.bulletActiveClass + "-next-next " + params.bulletActiveClass + "-prev " +
params.bulletActiveClass + "-prev-prev " + params.bulletActiveClass + "-main");

        if ($el.length > 1) {
          bullets.each(function (bullet) {
            var $bullet = $(bullet);
            var bulletIndex = $bullet.index();

            if (bulletIndex === current) {
              $bullet.addClass(params.bulletActiveClass);
            }

            if (params.dynamicBullets) {
              if (bulletIndex >= firstIndex && bulletIndex <= lastIndex) {
                $bullet.addClass(params.bulletActiveClass + "-main");
              }
              if (bulletIndex === firstIndex) {
                $bullet.prev().addClass(params.bulletActiveClass + "-
prev").prev().addClass(params.bulletActiveClass + "-prev-prev");
              }

              if (bulletIndex === lastIndex) {
                $bullet.next().addClass(params.bulletActiveClass + "-
next").next().addClass(params.bulletActiveClass + "-next-next");
              }
            }
          });
        } else {
          var $bullet = bullets.eq(current);
          var bulletIndex = $bullet.index();
          $bullet.addClass(params.bulletActiveClass);

          if (params.dynamicBullets) {
            var $firstDisplayedBullet = bullets.eq(firstIndex);
            var $lastDisplayedBullet = bullets.eq(lastIndex);

            for (var i = firstIndex; i <= lastIndex; i += 1) {
              bullets.eq(i).addClass(params.bulletActiveClass + "-main");
            }

            if (swiper.params.loop) {
              if (bulletIndex >= bullets.length - params.dynamicMainBullets) {
                for (var _i = params.dynamicMainBullets; _i >= 0; _i -= 1) {
                  bullets.eq(bullets.length - _i).addClass(params.bulletActiveClass + "-main");
                }

                bullets.eq(bullets.length - params.dynamicMainBullets -
1).addClass(params.bulletActiveClass + "-prev");
              } else {
                $firstDisplayedBullet.prev().addClass(params.bulletActiveClass + "-
prev").prev().addClass(params.bulletActiveClass + "-prev-prev");
                $lastDisplayedBullet.next().addClass(params.bulletActiveClass + "-
next").next().addClass(params.bulletActiveClass + "-next-next");
              }
            } else {
              $firstDisplayedBullet.prev().addClass(params.bulletActiveClass + "-
prev").prev().addClass(params.bulletActiveClass + "-prev-prev");
              $lastDisplayedBullet.next().addClass(params.bulletActiveClass + "-
next").next().addClass(params.bulletActiveClass + "-next-next");
            }
          }
        }

        if (params.dynamicBullets) {
          var dynamicBulletsLength = Math.min(bullets.length, params.dynamicMainBullets + 4);
          var bulletsOffset = (swiper.pagination.bulletSize * dynamicBulletsLength -
swiper.pagination.bulletSize) / 2 - midIndex * swiper.pagination.bulletSize;
          var offsetProp = rtl ? 'right' : 'left';
          bullets.css(swiper.isHorizontal() ? offsetProp : 'top', bulletsOffset + "px");
        }
      }

      if (params.type === 'fraction') {
        $el.find(classesToSelector(params.currentClass)).text(params.formatFractionCurrent(current +
1));
        $el.find(classesToSelector(params.totalClass)).text(params.formatFractionTotal(total));
      }

      if (params.type === 'progressbar') {
        var progressbarDirection;
        if (params.progressbarOpposite) {
          progressbarDirection = swiper.isHorizontal() ? 'vertical' : 'horizontal';
        } else {
          progressbarDirection = swiper.isHorizontal() ? 'horizontal' : 'vertical';
        }

        var scale = (current + 1) / total;
        var scaleX = 1;
        var scaleY = 1;

        if (progressbarDirection === 'horizontal') {
          scaleX = scale;
        } else {
          scaleY = scale;
        }

        $el.find(classesToSelector(params.progressbarFillClass)).transform("translate3d(0,0,0)
scaleX(" + scaleX + ") scaleY(" + scaleY + ")").transition(swiper.params.speed);
      }

      if (params.type === 'custom' && params.renderCustom) {
        $el.html(params.renderCustom(swiper, current + 1, total));
        swiper.emit('paginationRender', $el[0]);
      } else {
        swiper.emit('paginationUpdate', $el[0]);
      }

      if (swiper.params.watchOverflow && swiper.enabled) {
        $el[swiper.isLocked ? 'addClass' : 'removeClass'](params.lockClass);
      }
    },
    render: function render() {
      // Render Container
      var swiper = this;
      var params = swiper.params.pagination;
      if (!params.el || !swiper.pagination.el || !swiper.pagination.$el ||
swiper.pagination.$el.length === 0) return;
      var slidesLength = swiper.virtual && swiper.params.virtual.enabled ?
swiper.virtual.slides.length : swiper.slides.length;
      var $el = swiper.pagination.$el;
      var paginationHTML = '';

      if (params.type === 'bullets') {
        var numberOfBullets = swiper.params.loop ? Math.ceil((slidesLength - swiper.loopedSlides * 2)
/ swiper.params.slidesPerGroup) : swiper.snapGrid.length;

        if (swiper.params.freeMode && !swiper.params.loop && numberOfBullets > slidesLength) {
          numberOfBullets = slidesLength;
        }

        for (var i = 0; i < numberOfBullets; i += 1) {
          if (params.renderBullet) {
            paginationHTML += params.renderBullet.call(swiper, i, params.bulletClass);
          } else {
            paginationHTML += "<" + params.bulletElement + " class=\"" + params.bulletClass + "\"></"
+ params.bulletElement + ">";
          }
        }

        $el.html(paginationHTML);
        swiper.pagination.bullets = $el.find(classesToSelector(params.bulletClass));
      }

      if (params.type === 'fraction') {
        if (params.renderFraction) {
          paginationHTML = params.renderFraction.call(swiper, params.currentClass,
params.totalClass);
        } else {
          paginationHTML = "<span class=\"" + params.currentClass + "\"></span>" + ' / ' + ("<span
class=\"" + params.totalClass + "\"></span>");
        }

        $el.html(paginationHTML);
      }

      if (params.type === 'progressbar') {
        if (params.renderProgressbar) {
          paginationHTML = params.renderProgressbar.call(swiper, params.progressbarFillClass);
        } else {
          paginationHTML = "<span class=\"" + params.progressbarFillClass + "\"></span>";
        }

        $el.html(paginationHTML);
      }

      if (params.type !== 'custom') {
        swiper.emit('paginationRender', swiper.pagination.$el[0]);
      }
    },
    init: function init() {
      var swiper = this;
      swiper.params.pagination = createElementIfNotDefined(swiper.$el, swiper.params.pagination,
swiper.params.createElements, {
        el: 'swiper-pagination'
      });
      var params = swiper.params.pagination;
      if (!params.el) return;
      var $el = $(params.el);
      if ($el.length === 0) return;

      if (swiper.params.uniqueNavElements && typeof params.el === 'string' && $el.length > 1) {
        $el = swiper.$el.find(params.el);
      }

      if (params.type === 'bullets' && params.clickable) {
        $el.addClass(params.clickableClass);
      }

      $el.addClass(params.modifierClass + params.type);

      if (params.type === 'bullets' && params.dynamicBullets) {
        $el.addClass("" + params.modifierClass + params.type + "-dynamic");
        swiper.pagination.dynamicBulletIndex = 0;

        if (params.dynamicMainBullets < 1) {
          params.dynamicMainBullets = 1;
        }
      }

      if (params.type === 'progressbar' && params.progressbarOpposite) {
        $el.addClass(params.progressbarOppositeClass);
      }

      if (params.clickable) {
        $el.on('click', classesToSelector(params.bulletClass), function onClick(e) {
          e.preventDefault();
          var index = $(this).index() * swiper.params.slidesPerGroup;
          if (swiper.params.loop) index += swiper.loopedSlides;
          swiper.slideTo(index);
        });
      }

      extend(swiper.pagination, {
        $el: $el,
        el: $el[0]
      });

      if (!swiper.enabled) {
        $el.addClass(params.lockClass);
      }
    },
    destroy: function destroy() {
      var swiper = this;
      var params = swiper.params.pagination;
      if (!params.el || !swiper.pagination.el || !swiper.pagination.$el ||
swiper.pagination.$el.length === 0) return;
      var $el = swiper.pagination.$el;
      $el.removeClass(params.hiddenClass);
      $el.removeClass(params.modifierClass + params.type);
      if (swiper.pagination.bullets) swiper.pagination.bullets.removeClass(params.bulletActiveClass);

      if (params.clickable) {
        $el.off('click', classesToSelector(params.bulletClass));
      }
    }
  };
  var Pagination$1 = {
    name: 'pagination',
    params: {
      pagination: {
        el: null,
        bulletElement: 'span',
        clickable: false,
        hideOnClick: false,
        renderBullet: null,
        renderProgressbar: null,
        renderFraction: null,
        renderCustom: null,
        progressbarOpposite: false,
        type: 'bullets',
        // 'bullets' or 'progressbar' or 'fraction' or 'custom'
        dynamicBullets: false,
        dynamicMainBullets: 1,
        formatFractionCurrent: function formatFractionCurrent(number) {
          return number;
        },
        formatFractionTotal: function formatFractionTotal(number) {
          return number;
        },
        bulletClass: 'swiper-pagination-bullet',
        bulletActiveClass: 'swiper-pagination-bullet-active',
        modifierClass: 'swiper-pagination-',
        // NEW
        currentClass: 'swiper-pagination-current',
        totalClass: 'swiper-pagination-total',
        hiddenClass: 'swiper-pagination-hidden',
        progressbarFillClass: 'swiper-pagination-progressbar-fill',
        progressbarOppositeClass: 'swiper-pagination-progressbar-opposite',
        clickableClass: 'swiper-pagination-clickable',
        // NEW
        lockClass: 'swiper-pagination-lock'
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        pagination: _extends({
          dynamicBulletIndex: 0
        }, Pagination)
      });
    },
    on: {
      init: function init(swiper) {
        swiper.pagination.init();
        swiper.pagination.render();
        swiper.pagination.update();
      },
      activeIndexChange: function activeIndexChange(swiper) {
        if (swiper.params.loop) {
          swiper.pagination.update();
        } else if (typeof swiper.snapIndex === 'undefined') {
          swiper.pagination.update();
        }
      },
      snapIndexChange: function snapIndexChange(swiper) {
        if (!swiper.params.loop) {
          swiper.pagination.update();
        }
      },
      slidesLengthChange: function slidesLengthChange(swiper) {
        if (swiper.params.loop) {
          swiper.pagination.render();
          swiper.pagination.update();
        }
      },
      snapGridLengthChange: function snapGridLengthChange(swiper) {
        if (!swiper.params.loop) {
          swiper.pagination.render();
          swiper.pagination.update();
        }
      },
      destroy: function destroy(swiper) {
        swiper.pagination.destroy();
      },
      'enable disable': function enableDisable(swiper) {
        var $el = swiper.pagination.$el;

        if ($el) {
          $el[swiper.enabled ? 'removeClass' : 'addClass'](swiper.params.pagination.lockClass);
        }
      },
      click: function click(swiper, e) {
        var targetEl = e.target;

        if (swiper.params.pagination.el && swiper.params.pagination.hideOnClick &&
swiper.pagination.$el.length > 0 && !$(targetEl).hasClass(swiper.params.pagination.bulletClass)) {
          if (swiper.navigation && (swiper.navigation.nextEl && targetEl === swiper.navigation.nextEl
|| swiper.navigation.prevEl && targetEl === swiper.navigation.prevEl)) return;
          var isHidden = swiper.pagination.$el.hasClass(swiper.params.pagination.hiddenClass);

          if (isHidden === true) {
            swiper.emit('paginationShow');
          } else {
            swiper.emit('paginationHide');
          }

          swiper.pagination.$el.toggleClass(swiper.params.pagination.hiddenClass);
        }
      }
    }
  };

  var Scrollbar = {
    setTranslate: function setTranslate() {
      var swiper = this;
      if (!swiper.params.scrollbar.el || !swiper.scrollbar.el) return;
      var scrollbar = swiper.scrollbar,
          rtl = swiper.rtlTranslate,
          progress = swiper.progress;
      var dragSize = scrollbar.dragSize,
          trackSize = scrollbar.trackSize,
          $dragEl = scrollbar.$dragEl,
          $el = scrollbar.$el;
      var params = swiper.params.scrollbar;
      var newSize = dragSize;
      var newPos = (trackSize - dragSize) * progress;

      if (rtl) {
        newPos = -newPos;

        if (newPos > 0) {
          newSize = dragSize - newPos;
          newPos = 0;
        } else if (-newPos + dragSize > trackSize) {
          newSize = trackSize + newPos;
        }
      } else if (newPos < 0) {
        newSize = dragSize + newPos;
        newPos = 0;
      } else if (newPos + dragSize > trackSize) {
        newSize = trackSize - newPos;
      }

      if (swiper.isHorizontal()) {
        $dragEl.transform("translate3d(" + newPos + "px, 0, 0)");
        $dragEl[0].style.width = newSize + "px";
      } else {
        $dragEl.transform("translate3d(0px, " + newPos + "px, 0)");
        $dragEl[0].style.height = newSize + "px";
      }

      if (params.hide) {
        clearTimeout(swiper.scrollbar.timeout);
        $el[0].style.opacity = 1;
        swiper.scrollbar.timeout = setTimeout(function () {
          $el[0].style.opacity = 0;
          $el.transition(400);
        }, 1000);
      }
    },
    setTransition: function setTransition(duration) {
      var swiper = this;
      if (!swiper.params.scrollbar.el || !swiper.scrollbar.el) return;
      swiper.scrollbar.$dragEl.transition(duration);
    },
    updateSize: function updateSize() {
      var swiper = this;
      if (!swiper.params.scrollbar.el || !swiper.scrollbar.el) return;
      var scrollbar = swiper.scrollbar;
      var $dragEl = scrollbar.$dragEl,
          $el = scrollbar.$el;
      $dragEl[0].style.width = '';
      $dragEl[0].style.height = '';
      var trackSize = swiper.isHorizontal() ? $el[0].offsetWidth : $el[0].offsetHeight;
      var divider = swiper.size / swiper.virtualSize;
      var moveDivider = divider * (trackSize / swiper.size);
      var dragSize;

      if (swiper.params.scrollbar.dragSize === 'auto') {
        dragSize = trackSize * divider;
      } else {
        dragSize = parseInt(swiper.params.scrollbar.dragSize, 10);
      }

      if (swiper.isHorizontal()) {
        $dragEl[0].style.width = dragSize + "px";
      } else {
        $dragEl[0].style.height = dragSize + "px";
      }

      if (divider >= 1) {
        $el[0].style.display = 'none';
      } else {
        $el[0].style.display = '';
      }

      if (swiper.params.scrollbar.hide) {
        $el[0].style.opacity = 0;
      }

      extend(scrollbar, {
        trackSize: trackSize,
        divider: divider,
        moveDivider: moveDivider,
        dragSize: dragSize
      });

      if (swiper.params.watchOverflow && swiper.enabled) {
        scrollbar.$el[swiper.isLocked ? 'addClass' : 'removeClass']
(swiper.params.scrollbar.lockClass);
      }
    },
    getPointerPosition: function getPointerPosition(e) {
      var swiper = this;

      if (swiper.isHorizontal()) {
        return e.type === 'touchstart' || e.type === 'touchmove' ? e.targetTouches[0].clientX :
e.clientX;
      }

      return e.type === 'touchstart' || e.type === 'touchmove' ? e.targetTouches[0].clientY :
e.clientY;
    },
    setDragPosition: function setDragPosition(e) {
      var swiper = this;
      var scrollbar = swiper.scrollbar,
          rtl = swiper.rtlTranslate;
      var $el = scrollbar.$el,
          dragSize = scrollbar.dragSize,
          trackSize = scrollbar.trackSize,
          dragStartPos = scrollbar.dragStartPos;
      var positionRatio;
      positionRatio = (scrollbar.getPointerPosition(e) - $el.offset()[swiper.isHorizontal() ? 'left'
: 'top'] - (dragStartPos !== null ? dragStartPos : dragSize / 2)) / (trackSize - dragSize);
      positionRatio = Math.max(Math.min(positionRatio, 1), 0);

      if (rtl) {
        positionRatio = 1 - positionRatio;
      }
      var position = swiper.minTranslate() + (swiper.maxTranslate() - swiper.minTranslate()) *
positionRatio;
      swiper.updateProgress(position);
      swiper.setTranslate(position);
      swiper.updateActiveIndex();
      swiper.updateSlidesClasses();
    },
    onDragStart: function onDragStart(e) {
      var swiper = this;
      var params = swiper.params.scrollbar;
      var scrollbar = swiper.scrollbar,
          $wrapperEl = swiper.$wrapperEl;
      var $el = scrollbar.$el,
          $dragEl = scrollbar.$dragEl;
      swiper.scrollbar.isTouched = true;
      swiper.scrollbar.dragStartPos = e.target === $dragEl[0] || e.target === $dragEl ?
scrollbar.getPointerPosition(e) - e.target.getBoundingClientRect()[swiper.isHorizontal() ? 'left' :
'top'] : null;
      e.preventDefault();
      e.stopPropagation();
      $wrapperEl.transition(100);
      $dragEl.transition(100);
      scrollbar.setDragPosition(e);
      clearTimeout(swiper.scrollbar.dragTimeout);
      $el.transition(0);

      if (params.hide) {
        $el.css('opacity', 1);
      }

      if (swiper.params.cssMode) {
        swiper.$wrapperEl.css('scroll-snap-type', 'none');
      }

      swiper.emit('scrollbarDragStart', e);
    },
    onDragMove: function onDragMove(e) {
      var swiper = this;
      var scrollbar = swiper.scrollbar,
          $wrapperEl = swiper.$wrapperEl;
      var $el = scrollbar.$el,
          $dragEl = scrollbar.$dragEl;
      if (!swiper.scrollbar.isTouched) return;
      if (e.preventDefault) e.preventDefault();else e.returnValue = false;
      scrollbar.setDragPosition(e);
      $wrapperEl.transition(0);
      $el.transition(0);
      $dragEl.transition(0);
      swiper.emit('scrollbarDragMove', e);
    },
    onDragEnd: function onDragEnd(e) {
      var swiper = this;
      var params = swiper.params.scrollbar;
      var scrollbar = swiper.scrollbar,
          $wrapperEl = swiper.$wrapperEl;
      var $el = scrollbar.$el;
      if (!swiper.scrollbar.isTouched) return;
      swiper.scrollbar.isTouched = false;

      if (swiper.params.cssMode) {
        swiper.$wrapperEl.css('scroll-snap-type', '');
        $wrapperEl.transition('');
      }
      if (params.hide) {
        clearTimeout(swiper.scrollbar.dragTimeout);
        swiper.scrollbar.dragTimeout = nextTick(function () {
          $el.css('opacity', 0);
          $el.transition(400);
        }, 1000);
      }

      swiper.emit('scrollbarDragEnd', e);

      if (params.snapOnRelease) {
        swiper.slideToClosest();
      }
    },
    enableDraggable: function enableDraggable() {
      var swiper = this;
      if (!swiper.params.scrollbar.el) return;
      var document = getDocument();
      var scrollbar = swiper.scrollbar,
          touchEventsTouch = swiper.touchEventsTouch,
          touchEventsDesktop = swiper.touchEventsDesktop,
          params = swiper.params,
          support = swiper.support;
      var $el = scrollbar.$el;
      var target = $el[0];
      var activeListener = support.passiveListener && params.passiveListeners ? {
        passive: false,
        capture: false
      } : false;
      var passiveListener = support.passiveListener && params.passiveListeners ? {
        passive: true,
        capture: false
      } : false;
      if (!target) return;

      if (!support.touch) {
        target.addEventListener(touchEventsDesktop.start, swiper.scrollbar.onDragStart,
activeListener);
        document.addEventListener(touchEventsDesktop.move, swiper.scrollbar.onDragMove,
activeListener);
        document.addEventListener(touchEventsDesktop.end, swiper.scrollbar.onDragEnd,
passiveListener);
      } else {
        target.addEventListener(touchEventsTouch.start, swiper.scrollbar.onDragStart,
activeListener);
        target.addEventListener(touchEventsTouch.move, swiper.scrollbar.onDragMove, activeListener);
        target.addEventListener(touchEventsTouch.end, swiper.scrollbar.onDragEnd, passiveListener);
      }
    },
    disableDraggable: function disableDraggable() {
      var swiper = this;
      if (!swiper.params.scrollbar.el) return;
      var document = getDocument();
      var scrollbar = swiper.scrollbar,
          touchEventsTouch = swiper.touchEventsTouch,
          touchEventsDesktop = swiper.touchEventsDesktop,
          params = swiper.params,
          support = swiper.support;
      var $el = scrollbar.$el;
      var target = $el[0];
      var activeListener = support.passiveListener && params.passiveListeners ? {
        passive: false,
        capture: false
      } : false;
      var passiveListener = support.passiveListener && params.passiveListeners ? {
        passive: true,
        capture: false
      } : false;
      if (!target) return;

      if (!support.touch) {
        target.removeEventListener(touchEventsDesktop.start, swiper.scrollbar.onDragStart,
activeListener);
        document.removeEventListener(touchEventsDesktop.move, swiper.scrollbar.onDragMove,
activeListener);
        document.removeEventListener(touchEventsDesktop.end, swiper.scrollbar.onDragEnd,
passiveListener);
      } else {
        target.removeEventListener(touchEventsTouch.start, swiper.scrollbar.onDragStart,
activeListener);
        target.removeEventListener(touchEventsTouch.move, swiper.scrollbar.onDragMove,
activeListener);
        target.removeEventListener(touchEventsTouch.end, swiper.scrollbar.onDragEnd,
passiveListener);
      }
    },
    init: function init() {
      var swiper = this;
      var scrollbar = swiper.scrollbar,
          $swiperEl = swiper.$el;
      swiper.params.scrollbar = createElementIfNotDefined($swiperEl, swiper.params.scrollbar,
swiper.params.createElements, {
        el: 'swiper-scrollbar'
      });
      var params = swiper.params.scrollbar;
      if (!params.el) return;
      var $el = $(params.el);

      if (swiper.params.uniqueNavElements && typeof params.el === 'string' && $el.length > 1 &&
$swiperEl.find(params.el).length === 1) {
        $el = $swiperEl.find(params.el);
      }

      var $dragEl = $el.find("." + swiper.params.scrollbar.dragClass);

      if ($dragEl.length === 0) {
        $dragEl = $("<div class=\"" + swiper.params.scrollbar.dragClass + "\"></div>");
        $el.append($dragEl);
      }

      extend(scrollbar, {
        $el: $el,
        el: $el[0],
        $dragEl: $dragEl,
        dragEl: $dragEl[0]
      });

      if (params.draggable) {
        scrollbar.enableDraggable();
      }

      if ($el) {
        $el[swiper.enabled ? 'removeClass' : 'addClass'](swiper.params.scrollbar.lockClass);
      }
    },
    destroy: function destroy() {
      var swiper = this;
      swiper.scrollbar.disableDraggable();
    }
  };
  var Scrollbar$1 = {
    name: 'scrollbar',
    params: {
      scrollbar: {
        el: null,
        dragSize: 'auto',
        hide: false,
        draggable: false,
        snapOnRelease: true,
        lockClass: 'swiper-scrollbar-lock',
        dragClass: 'swiper-scrollbar-drag'
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        scrollbar: _extends({
          isTouched: false,
          timeout: null,
          dragTimeout: null
        }, Scrollbar)
      });
    },
    on: {
      init: function init(swiper) {
        swiper.scrollbar.init();
        swiper.scrollbar.updateSize();
        swiper.scrollbar.setTranslate();
      },
      update: function update(swiper) {
        swiper.scrollbar.updateSize();
      },
      resize: function resize(swiper) {
        swiper.scrollbar.updateSize();
      },
      observerUpdate: function observerUpdate(swiper) {
        swiper.scrollbar.updateSize();
      },
      setTranslate: function setTranslate(swiper) {
        swiper.scrollbar.setTranslate();
      },
      setTransition: function setTransition(swiper, duration) {
        swiper.scrollbar.setTransition(duration);
      },
      'enable disable': function enableDisable(swiper) {
        var $el = swiper.scrollbar.$el;

        if ($el) {
          $el[swiper.enabled ? 'removeClass' : 'addClass'](swiper.params.scrollbar.lockClass);
        }
      },
      destroy: function destroy(swiper) {
        swiper.scrollbar.destroy();
      }
    }
  };

  var Parallax = {
    setTransform: function setTransform(el, progress) {
      var swiper = this;
      var rtl = swiper.rtl;
      var $el = $(el);
      var rtlFactor = rtl ? -1 : 1;
      var p = $el.attr('data-swiper-parallax') || '0';
      var x = $el.attr('data-swiper-parallax-x');
      var y = $el.attr('data-swiper-parallax-y');
      var scale = $el.attr('data-swiper-parallax-scale');
      var opacity = $el.attr('data-swiper-parallax-opacity');

      if (x || y) {
        x = x || '0';
        y = y || '0';
      } else if (swiper.isHorizontal()) {
        x = p;
        y = '0';
      } else {
        y = p;
        x = '0';
      }

      if (x.indexOf('%') >= 0) {
        x = parseInt(x, 10) * progress * rtlFactor + "%";
      } else {
        x = x * progress * rtlFactor + "px";
      }

      if (y.indexOf('%') >= 0) {
        y = parseInt(y, 10) * progress + "%";
      } else {
        y = y * progress + "px";
      }

      if (typeof opacity !== 'undefined' && opacity !== null) {
        var currentOpacity = opacity - (opacity - 1) * (1 - Math.abs(progress));
        $el[0].style.opacity = currentOpacity;
      }

      if (typeof scale === 'undefined' || scale === null) {
        $el.transform("translate3d(" + x + ", " + y + ", 0px)");
      } else {
        var currentScale = scale - (scale - 1) * (1 - Math.abs(progress));
        $el.transform("translate3d(" + x + ", " + y + ", 0px) scale(" + currentScale + ")");
      }
    },
    setTranslate: function setTranslate() {
      var swiper = this;
      var $el = swiper.$el,
          slides = swiper.slides,
          progress = swiper.progress,
          snapGrid = swiper.snapGrid;
      $el.children('[data-swiper-parallax], [data-swiper-parallax-x], [data-swiper-parallax-y],
[data-swiper-parallax-opacity], [data-swiper-parallax-scale]').each(function (el) {
        swiper.parallax.setTransform(el, progress);
      });
      slides.each(function (slideEl, slideIndex) {
        var slideProgress = slideEl.progress;

        if (swiper.params.slidesPerGroup > 1 && swiper.params.slidesPerView !== 'auto') {
          slideProgress += Math.ceil(slideIndex / 2) - progress * (snapGrid.length - 1);
        }

        slideProgress = Math.min(Math.max(slideProgress, -1), 1);
        $(slideEl).find('[data-swiper-parallax], [data-swiper-parallax-x], [data-swiper-parallax-y],
[data-swiper-parallax-opacity], [data-swiper-parallax-scale]').each(function (el) {
          swiper.parallax.setTransform(el, slideProgress);
        });
      });
    },
    setTransition: function setTransition(duration) {
      if (duration === void 0) {
        duration = this.params.speed;
      }

      var swiper = this;
      var $el = swiper.$el;
      $el.find('[data-swiper-parallax], [data-swiper-parallax-x], [data-swiper-parallax-y], [data-
swiper-parallax-opacity], [data-swiper-parallax-scale]').each(function (parallaxEl) {
        var $parallaxEl = $(parallaxEl);
        var parallaxDuration = parseInt($parallaxEl.attr('data-swiper-parallax-duration'), 10) ||
duration;
        if (duration === 0) parallaxDuration = 0;
        $parallaxEl.transition(parallaxDuration);
      });
    }
  };
  var Parallax$1 = {
    name: 'parallax',
    params: {
      parallax: {
        enabled: false
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        parallax: _extends({}, Parallax)
      });
    },
    on: {
      beforeInit: function beforeInit(swiper) {
        if (!swiper.params.parallax.enabled) return;
        swiper.params.watchSlidesProgress = true;
        swiper.originalParams.watchSlidesProgress = true;
      },
      init: function init(swiper) {
        if (!swiper.params.parallax.enabled) return;
        swiper.parallax.setTranslate();
      },
      setTranslate: function setTranslate(swiper) {
        if (!swiper.params.parallax.enabled) return;
        swiper.parallax.setTranslate();
      },
      setTransition: function setTransition(swiper, duration) {
        if (!swiper.params.parallax.enabled) return;
        swiper.parallax.setTransition(duration);
      }
    }
  };

  var Zoom = {
    // Calc Scale From Multi-touches
    getDistanceBetweenTouches: function getDistanceBetweenTouches(e) {
      if (e.targetTouches.length < 2) return 1;
      var x1 = e.targetTouches[0].pageX;
      var y1 = e.targetTouches[0].pageY;
      var x2 = e.targetTouches[1].pageX;
      var y2 = e.targetTouches[1].pageY;
      var distance = Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));
      return distance;
    },
    // Events
    onGestureStart: function onGestureStart(e) {
      var swiper = this;
      var support = swiper.support;
      var params = swiper.params.zoom;
      var zoom = swiper.zoom;
      var gesture = zoom.gesture;
      zoom.fakeGestureTouched = false;
      zoom.fakeGestureMoved = false;

      if (!support.gestures) {
        if (e.type !== 'touchstart' || e.type === 'touchstart' && e.targetTouches.length < 2) {
          return;
        }

        zoom.fakeGestureTouched = true;
        gesture.scaleStart = Zoom.getDistanceBetweenTouches(e);
      }

      if (!gesture.$slideEl || !gesture.$slideEl.length) {
        gesture.$slideEl = $(e.target).closest("." + swiper.params.slideClass);
        if (gesture.$slideEl.length === 0) gesture.$slideEl = swiper.slides.eq(swiper.activeIndex);
        gesture.$imageEl = gesture.$slideEl.find('img, svg, canvas, picture, .swiper-zoom-target');
        gesture.$imageWrapEl = gesture.$imageEl.parent("." + params.containerClass);
        gesture.maxRatio = gesture.$imageWrapEl.attr('data-swiper-zoom') || params.maxRatio;

        if (gesture.$imageWrapEl.length === 0) {
          gesture.$imageEl = undefined;
          return;
        }
      }

      if (gesture.$imageEl) {
        gesture.$imageEl.transition(0);
      }

      swiper.zoom.isScaling = true;
    },
    onGestureChange: function onGestureChange(e) {
      var swiper = this;
      var support = swiper.support;
      var params = swiper.params.zoom;
      var zoom = swiper.zoom;
      var gesture = zoom.gesture;

      if (!support.gestures) {
        if (e.type !== 'touchmove' || e.type === 'touchmove' && e.targetTouches.length < 2) {
          return;
        }

        zoom.fakeGestureMoved = true;
        gesture.scaleMove = Zoom.getDistanceBetweenTouches(e);
      }

      if (!gesture.$imageEl || gesture.$imageEl.length === 0) {
        if (e.type === 'gesturechange') zoom.onGestureStart(e);
        return;
      }

      if (support.gestures) {
        zoom.scale = e.scale * zoom.currentScale;
      } else {
        zoom.scale = gesture.scaleMove / gesture.scaleStart * zoom.currentScale;
      }

      if (zoom.scale > gesture.maxRatio) {
        zoom.scale = gesture.maxRatio - 1 + Math.pow(zoom.scale - gesture.maxRatio + 1, 0.5);
      }

      if (zoom.scale < params.minRatio) {
        zoom.scale = params.minRatio + 1 - Math.pow(params.minRatio - zoom.scale + 1, 0.5);
      }

      gesture.$imageEl.transform("translate3d(0,0,0) scale(" + zoom.scale + ")");
    },
    onGestureEnd: function onGestureEnd(e) {
      var swiper = this;
      var device = swiper.device;
      var support = swiper.support;
      var params = swiper.params.zoom;
      var zoom = swiper.zoom;
      var gesture = zoom.gesture;

      if (!support.gestures) {
        if (!zoom.fakeGestureTouched || !zoom.fakeGestureMoved) {
          return;
        }

        if (e.type !== 'touchend' || e.type === 'touchend' && e.changedTouches.length < 2 &&
!device.android) {
          return;
        }

        zoom.fakeGestureTouched = false;
        zoom.fakeGestureMoved = false;
      }

      if (!gesture.$imageEl || gesture.$imageEl.length === 0) return;
      zoom.scale = Math.max(Math.min(zoom.scale, gesture.maxRatio), params.minRatio);
      gesture.$imageEl.transition(swiper.params.speed).transform("translate3d(0,0,0) scale(" +
zoom.scale + ")");
      zoom.currentScale = zoom.scale;
      zoom.isScaling = false;
      if (zoom.scale === 1) gesture.$slideEl = undefined;
    },
    onTouchStart: function onTouchStart(e) {
      var swiper = this;
      var device = swiper.device;
      var zoom = swiper.zoom;
      var gesture = zoom.gesture,
          image = zoom.image;
      if (!gesture.$imageEl || gesture.$imageEl.length === 0) return;
      if (image.isTouched) return;
      if (device.android && e.cancelable) e.preventDefault();
      image.isTouched = true;
      image.touchesStart.x = e.type === 'touchstart' ? e.targetTouches[0].pageX : e.pageX;
      image.touchesStart.y = e.type === 'touchstart' ? e.targetTouches[0].pageY : e.pageY;
    },
    onTouchMove: function onTouchMove(e) {
      var swiper = this;
      var zoom = swiper.zoom;
      var gesture = zoom.gesture,
          image = zoom.image,
          velocity = zoom.velocity;
      if (!gesture.$imageEl || gesture.$imageEl.length === 0) return;
      swiper.allowClick = false;
      if (!image.isTouched || !gesture.$slideEl) return;

      if (!image.isMoved) {
        image.width = gesture.$imageEl[0].offsetWidth;
        image.height = gesture.$imageEl[0].offsetHeight;
        image.startX = getTranslate(gesture.$imageWrapEl[0], 'x') || 0;
        image.startY = getTranslate(gesture.$imageWrapEl[0], 'y') || 0;
        gesture.slideWidth = gesture.$slideEl[0].offsetWidth;
        gesture.slideHeight = gesture.$slideEl[0].offsetHeight;
        gesture.$imageWrapEl.transition(0);
      } // Define if we need image drag


      var scaledWidth = image.width * zoom.scale;
      var scaledHeight = image.height * zoom.scale;
      if (scaledWidth < gesture.slideWidth && scaledHeight < gesture.slideHeight) return;
      image.minX = Math.min(gesture.slideWidth / 2 - scaledWidth / 2, 0);
      image.maxX = -image.minX;
      image.minY = Math.min(gesture.slideHeight / 2 - scaledHeight / 2, 0);
      image.maxY = -image.minY;
      image.touchesCurrent.x = e.type === 'touchmove' ? e.targetTouches[0].pageX : e.pageX;
      image.touchesCurrent.y = e.type === 'touchmove' ? e.targetTouches[0].pageY : e.pageY;

      if (!image.isMoved && !zoom.isScaling) {
        if (swiper.isHorizontal() && (Math.floor(image.minX) === Math.floor(image.startX) &&
image.touchesCurrent.x < image.touchesStart.x || Math.floor(image.maxX) === Math.floor(image.startX)
&& image.touchesCurrent.x > image.touchesStart.x)) {
          image.isTouched = false;
          return;
        }

        if (!swiper.isHorizontal() && (Math.floor(image.minY) === Math.floor(image.startY) &&
image.touchesCurrent.y < image.touchesStart.y || Math.floor(image.maxY) === Math.floor(image.startY)
&& image.touchesCurrent.y > image.touchesStart.y)) {
          image.isTouched = false;
          return;
        }
      }

      if (e.cancelable) {
        e.preventDefault();
      }

      e.stopPropagation();
      image.isMoved = true;
      image.currentX = image.touchesCurrent.x - image.touchesStart.x + image.startX;
      image.currentY = image.touchesCurrent.y - image.touchesStart.y + image.startY;

      if (image.currentX < image.minX) {
        image.currentX = image.minX + 1 - Math.pow(image.minX - image.currentX + 1, 0.8);
      }

      if (image.currentX > image.maxX) {
        image.currentX = image.maxX - 1 + Math.pow(image.currentX - image.maxX + 1, 0.8);
      }

      if (image.currentY < image.minY) {
        image.currentY = image.minY + 1 - Math.pow(image.minY - image.currentY + 1, 0.8);
      }

      if (image.currentY > image.maxY) {
        image.currentY = image.maxY - 1 + Math.pow(image.currentY - image.maxY + 1, 0.8);
      } // Velocity


      if (!velocity.prevPositionX) velocity.prevPositionX = image.touchesCurrent.x;
      if (!velocity.prevPositionY) velocity.prevPositionY = image.touchesCurrent.y;
      if (!velocity.prevTime) velocity.prevTime = Date.now();
      velocity.x = (image.touchesCurrent.x - velocity.prevPositionX) / (Date.now() -
velocity.prevTime) / 2;
      velocity.y = (image.touchesCurrent.y - velocity.prevPositionY) / (Date.now() -
velocity.prevTime) / 2;
      if (Math.abs(image.touchesCurrent.x - velocity.prevPositionX) < 2) velocity.x = 0;
      if (Math.abs(image.touchesCurrent.y - velocity.prevPositionY) < 2) velocity.y = 0;
      velocity.prevPositionX = image.touchesCurrent.x;
      velocity.prevPositionY = image.touchesCurrent.y;
      velocity.prevTime = Date.now();
      gesture.$imageWrapEl.transform("translate3d(" + image.currentX + "px, " + image.currentY +
"px,0)");
    },
    onTouchEnd: function onTouchEnd() {
      var swiper = this;
      var zoom = swiper.zoom;
      var gesture = zoom.gesture,
          image = zoom.image,
          velocity = zoom.velocity;
      if (!gesture.$imageEl || gesture.$imageEl.length === 0) return;

      if (!image.isTouched || !image.isMoved) {
        image.isTouched = false;
        image.isMoved = false;
        return;
      }

      image.isTouched = false;
      image.isMoved = false;
      var momentumDurationX = 300;
      var momentumDurationY = 300;
      var momentumDistanceX = velocity.x * momentumDurationX;
      var newPositionX = image.currentX + momentumDistanceX;
      var momentumDistanceY = velocity.y * momentumDurationY;
      var newPositionY = image.currentY + momentumDistanceY; // Fix duration

      if (velocity.x !== 0) momentumDurationX = Math.abs((newPositionX - image.currentX) /
velocity.x);
      if (velocity.y !== 0) momentumDurationY = Math.abs((newPositionY - image.currentY) /
velocity.y);
      var momentumDuration = Math.max(momentumDurationX, momentumDurationY);
      image.currentX = newPositionX;
      image.currentY = newPositionY; // Define if we need image drag

      var scaledWidth = image.width * zoom.scale;
      var scaledHeight = image.height * zoom.scale;
      image.minX = Math.min(gesture.slideWidth / 2 - scaledWidth / 2, 0);
      image.maxX = -image.minX;
      image.minY = Math.min(gesture.slideHeight / 2 - scaledHeight / 2, 0);
      image.maxY = -image.minY;
      image.currentX = Math.max(Math.min(image.currentX, image.maxX), image.minX);
      image.currentY = Math.max(Math.min(image.currentY, image.maxY), image.minY);
      gesture.$imageWrapEl.transition(momentumDuration).transform("translate3d(" + image.currentX +
"px, " + image.currentY + "px,0)");
    },
    onTransitionEnd: function onTransitionEnd() {
      var swiper = this;
      var zoom = swiper.zoom;
      var gesture = zoom.gesture;

      if (gesture.$slideEl && swiper.previousIndex !== swiper.activeIndex) {
        if (gesture.$imageEl) {
          gesture.$imageEl.transform('translate3d(0,0,0) scale(1)');
        }

        if (gesture.$imageWrapEl) {
          gesture.$imageWrapEl.transform('translate3d(0,0,0)');
        }

        zoom.scale = 1;
        zoom.currentScale = 1;
        gesture.$slideEl = undefined;
        gesture.$imageEl = undefined;
        gesture.$imageWrapEl = undefined;
      }
    },
    // Toggle Zoom
    toggle: function toggle(e) {
      var swiper = this;
      var zoom = swiper.zoom;

      if (zoom.scale && zoom.scale !== 1) {
        // Zoom Out
        zoom.out();
      } else {
        // Zoom In
        zoom.in(e);
      }
    },
    in: function _in(e) {
      var swiper = this;
      var window = getWindow();
      var zoom = swiper.zoom;
      var params = swiper.params.zoom;
      var gesture = zoom.gesture,
          image = zoom.image;

      if (!gesture.$slideEl) {
        if (e && e.target) {
          gesture.$slideEl = $(e.target).closest("." + swiper.params.slideClass);
        }

        if (!gesture.$slideEl) {
          if (swiper.params.virtual && swiper.params.virtual.enabled && swiper.virtual) {
            gesture.$slideEl = swiper.$wrapperEl.children("." + swiper.params.slideActiveClass);
          } else {
            gesture.$slideEl = swiper.slides.eq(swiper.activeIndex);
          }
        }

        gesture.$imageEl = gesture.$slideEl.find('img, svg, canvas, picture, .swiper-zoom-target');
        gesture.$imageWrapEl = gesture.$imageEl.parent("." + params.containerClass);
      }

      if (!gesture.$imageEl || gesture.$imageEl.length === 0 || !gesture.$imageWrapEl ||
gesture.$imageWrapEl.length === 0) return;
      gesture.$slideEl.addClass("" + params.zoomedSlideClass);
      var touchX;
      var touchY;
      var offsetX;
      var offsetY;
      var diffX;
      var diffY;
      var translateX;
      var translateY;
      var imageWidth;
      var imageHeight;
      var scaledWidth;
      var scaledHeight;
      var translateMinX;
      var translateMinY;
      var translateMaxX;
      var translateMaxY;
      var slideWidth;
      var slideHeight;

      if (typeof image.touchesStart.x === 'undefined' && e) {
        touchX = e.type === 'touchend' ? e.changedTouches[0].pageX : e.pageX;
        touchY = e.type === 'touchend' ? e.changedTouches[0].pageY : e.pageY;
      } else {
        touchX = image.touchesStart.x;
        touchY = image.touchesStart.y;
      }

      zoom.scale = gesture.$imageWrapEl.attr('data-swiper-zoom') || params.maxRatio;
      zoom.currentScale = gesture.$imageWrapEl.attr('data-swiper-zoom') || params.maxRatio;

      if (e) {
        slideWidth = gesture.$slideEl[0].offsetWidth;
        slideHeight = gesture.$slideEl[0].offsetHeight;
        offsetX = gesture.$slideEl.offset().left + window.scrollX;
        offsetY = gesture.$slideEl.offset().top + window.scrollY;
        diffX = offsetX + slideWidth / 2 - touchX;
        diffY = offsetY + slideHeight / 2 - touchY;
        imageWidth = gesture.$imageEl[0].offsetWidth;
        imageHeight = gesture.$imageEl[0].offsetHeight;
        scaledWidth = imageWidth * zoom.scale;
        scaledHeight = imageHeight * zoom.scale;
        translateMinX = Math.min(slideWidth / 2 - scaledWidth / 2, 0);
        translateMinY = Math.min(slideHeight / 2 - scaledHeight / 2, 0);
        translateMaxX = -translateMinX;
        translateMaxY = -translateMinY;
        translateX = diffX * zoom.scale;
        translateY = diffY * zoom.scale;

        if (translateX < translateMinX) {
          translateX = translateMinX;
        }

        if (translateX > translateMaxX) {
          translateX = translateMaxX;
        }

        if (translateY < translateMinY) {
          translateY = translateMinY;
        }

        if (translateY > translateMaxY) {
          translateY = translateMaxY;
        }
      } else {
        translateX = 0;
        translateY = 0;
      }

      gesture.$imageWrapEl.transition(300).transform("translate3d(" + translateX + "px, " +
translateY + "px,0)");
      gesture.$imageEl.transition(300).transform("translate3d(0,0,0) scale(" + zoom.scale + ")");
    },
    out: function out() {
      var swiper = this;
      var zoom = swiper.zoom;
      var params = swiper.params.zoom;
      var gesture = zoom.gesture;

      if (!gesture.$slideEl) {
        if (swiper.params.virtual && swiper.params.virtual.enabled && swiper.virtual) {
          gesture.$slideEl = swiper.$wrapperEl.children("." + swiper.params.slideActiveClass);
        } else {
          gesture.$slideEl = swiper.slides.eq(swiper.activeIndex);
        }
        gesture.$imageEl = gesture.$slideEl.find('img, svg, canvas, picture, .swiper-zoom-target');
        gesture.$imageWrapEl = gesture.$imageEl.parent("." + params.containerClass);
      }

      if (!gesture.$imageEl || gesture.$imageEl.length === 0 || !gesture.$imageWrapEl ||
gesture.$imageWrapEl.length === 0) return;
      zoom.scale = 1;
      zoom.currentScale = 1;
      gesture.$imageWrapEl.transition(300).transform('translate3d(0,0,0)');
      gesture.$imageEl.transition(300).transform('translate3d(0,0,0) scale(1)');
      gesture.$slideEl.removeClass("" + params.zoomedSlideClass);
      gesture.$slideEl = undefined;
    },
    toggleGestures: function toggleGestures(method) {
      var swiper = this;
      var zoom = swiper.zoom;
      var selector = zoom.slideSelector,
          passive = zoom.passiveListener;
      swiper.$wrapperEl[method]('gesturestart', selector, zoom.onGestureStart, passive);
      swiper.$wrapperEl[method]('gesturechange', selector, zoom.onGestureChange, passive);
      swiper.$wrapperEl[method]('gestureend', selector, zoom.onGestureEnd, passive);
    },
    enableGestures: function enableGestures() {
      if (this.zoom.gesturesEnabled) return;
      this.zoom.gesturesEnabled = true;
      this.zoom.toggleGestures('on');
    },
    disableGestures: function disableGestures() {
      if (!this.zoom.gesturesEnabled) return;
      this.zoom.gesturesEnabled = false;
      this.zoom.toggleGestures('off');
    },
    // Attach/Detach Events
    enable: function enable() {
      var swiper = this;
      var support = swiper.support;
      var zoom = swiper.zoom;
      if (zoom.enabled) return;
      zoom.enabled = true;
      var passiveListener = swiper.touchEvents.start === 'touchstart' && support.passiveListener &&
swiper.params.passiveListeners ? {
        passive: true,
        capture: false
      } : false;
      var activeListenerWithCapture = support.passiveListener ? {
        passive: false,
        capture: true
      } : true;
      var slideSelector = "." + swiper.params.slideClass;
      swiper.zoom.passiveListener = passiveListener;
      swiper.zoom.slideSelector = slideSelector; // Scale image

      if (support.gestures) {
        swiper.$wrapperEl.on(swiper.touchEvents.start, swiper.zoom.enableGestures, passiveListener);
        swiper.$wrapperEl.on(swiper.touchEvents.end, swiper.zoom.disableGestures, passiveListener);
      } else if (swiper.touchEvents.start === 'touchstart') {
        swiper.$wrapperEl.on(swiper.touchEvents.start, slideSelector, zoom.onGestureStart,
passiveListener);
        swiper.$wrapperEl.on(swiper.touchEvents.move, slideSelector, zoom.onGestureChange,
activeListenerWithCapture);
        swiper.$wrapperEl.on(swiper.touchEvents.end, slideSelector, zoom.onGestureEnd,
passiveListener);

        if (swiper.touchEvents.cancel) {
          swiper.$wrapperEl.on(swiper.touchEvents.cancel, slideSelector, zoom.onGestureEnd,
passiveListener);
        }
      } // Move image


      swiper.$wrapperEl.on(swiper.touchEvents.move, "." + swiper.params.zoom.containerClass,
zoom.onTouchMove, activeListenerWithCapture);
    },
    disable: function disable() {
      var swiper = this;
      var zoom = swiper.zoom;
      if (!zoom.enabled) return;
      var support = swiper.support;
      swiper.zoom.enabled = false;
      var passiveListener = swiper.touchEvents.start === 'touchstart' && support.passiveListener &&
swiper.params.passiveListeners ? {
        passive: true,
        capture: false
      } : false;
      var activeListenerWithCapture = support.passiveListener ? {
        passive: false,
        capture: true
      } : true;
      var slideSelector = "." + swiper.params.slideClass; // Scale image

      if (support.gestures) {
        swiper.$wrapperEl.off(swiper.touchEvents.start, swiper.zoom.enableGestures, passiveListener);
        swiper.$wrapperEl.off(swiper.touchEvents.end, swiper.zoom.disableGestures, passiveListener);
      } else if (swiper.touchEvents.start === 'touchstart') {
        swiper.$wrapperEl.off(swiper.touchEvents.start, slideSelector, zoom.onGestureStart,
passiveListener);
        swiper.$wrapperEl.off(swiper.touchEvents.move, slideSelector, zoom.onGestureChange,
activeListenerWithCapture);
        swiper.$wrapperEl.off(swiper.touchEvents.end, slideSelector, zoom.onGestureEnd,
passiveListener);

        if (swiper.touchEvents.cancel) {
          swiper.$wrapperEl.off(swiper.touchEvents.cancel, slideSelector, zoom.onGestureEnd,
passiveListener);
        }
      } // Move image


      swiper.$wrapperEl.off(swiper.touchEvents.move, "." + swiper.params.zoom.containerClass,
zoom.onTouchMove, activeListenerWithCapture);
    }
  };
  var Zoom$1 = {
    name: 'zoom',
    params: {
      zoom: {
        enabled: false,
        maxRatio: 3,
        minRatio: 1,
        toggle: true,
        containerClass: 'swiper-zoom-container',
        zoomedSlideClass: 'swiper-slide-zoomed'
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        zoom: _extends({
          enabled: false,
          scale: 1,
          currentScale: 1,
          isScaling: false,
          gesture: {
            $slideEl: undefined,
            slideWidth: undefined,
            slideHeight: undefined,
            $imageEl: undefined,
            $imageWrapEl: undefined,
            maxRatio: 3
          },
          image: {
            isTouched: undefined,
            isMoved: undefined,
            currentX: undefined,
            currentY: undefined,
            minX: undefined,
            minY: undefined,
            maxX: undefined,
            maxY: undefined,
            width: undefined,
            height: undefined,
            startX: undefined,
            startY: undefined,
            touchesStart: {},
            touchesCurrent: {}
          },
          velocity: {
            x: undefined,
            y: undefined,
            prevPositionX: undefined,
            prevPositionY: undefined,
            prevTime: undefined
          }
        }, Zoom)
      });
      var scale = 1;
      Object.defineProperty(swiper.zoom, 'scale', {
        get: function get() {
          return scale;
        },
        set: function set(value) {
          if (scale !== value) {
            var imageEl = swiper.zoom.gesture.$imageEl ? swiper.zoom.gesture.$imageEl[0] : undefined;
            var slideEl = swiper.zoom.gesture.$slideEl ? swiper.zoom.gesture.$slideEl[0] : undefined;
            swiper.emit('zoomChange', value, imageEl, slideEl);
          }

          scale = value;
        }
      });
    },
    on: {
      init: function init(swiper) {
        if (swiper.params.zoom.enabled) {
          swiper.zoom.enable();
        }
      },
      destroy: function destroy(swiper) {
        swiper.zoom.disable();
      },
      touchStart: function touchStart(swiper, e) {
        if (!swiper.zoom.enabled) return;
        swiper.zoom.onTouchStart(e);
      },
      touchEnd: function touchEnd(swiper, e) {
        if (!swiper.zoom.enabled) return;
        swiper.zoom.onTouchEnd(e);
      },
      doubleTap: function doubleTap(swiper, e) {
        if (!swiper.animating && swiper.params.zoom.enabled && swiper.zoom.enabled &&
swiper.params.zoom.toggle) {
          swiper.zoom.toggle(e);
        }
      },
      transitionEnd: function transitionEnd(swiper) {
        if (swiper.zoom.enabled && swiper.params.zoom.enabled) {
          swiper.zoom.onTransitionEnd();
        }
      },
      slideChange: function slideChange(swiper) {
        if (swiper.zoom.enabled && swiper.params.zoom.enabled && swiper.params.cssMode) {
          swiper.zoom.onTransitionEnd();
        }
      }
    }
  };

  var Lazy = {
    loadInSlide: function loadInSlide(index, loadInDuplicate) {
      if (loadInDuplicate === void 0) {
        loadInDuplicate = true;
      }

      var swiper = this;
      var params = swiper.params.lazy;
      if (typeof index === 'undefined') return;
      if (swiper.slides.length === 0) return;
      var isVirtual = swiper.virtual && swiper.params.virtual.enabled;
      var $slideEl = isVirtual ? swiper.$wrapperEl.children("." + swiper.params.slideClass + "[data-
swiper-slide-index=\"" + index + "\"]") : swiper.slides.eq(index);
      var $images = $slideEl.find("." + params.elementClass + ":not(." + params.loadedClass +
"):not(." + params.loadingClass + ")");

      if ($slideEl.hasClass(params.elementClass) && !$slideEl.hasClass(params.loadedClass) &&
!$slideEl.hasClass(params.loadingClass)) {
        $images.push($slideEl[0]);
      }

      if ($images.length === 0) return;
      $images.each(function (imageEl) {
        var $imageEl = $(imageEl);
        $imageEl.addClass(params.loadingClass);
        var background = $imageEl.attr('data-background');
        var src = $imageEl.attr('data-src');
        var srcset = $imageEl.attr('data-srcset');
        var sizes = $imageEl.attr('data-sizes');
        var $pictureEl = $imageEl.parent('picture');
        swiper.loadImage($imageEl[0], src || background, srcset, sizes, false, function () {
          if (typeof swiper === 'undefined' || swiper === null || !swiper || swiper && !swiper.params
|| swiper.destroyed) return;

          if (background) {
            $imageEl.css('background-image', "url(\"" + background + "\")");
            $imageEl.removeAttr('data-background');
          } else {
            if (srcset) {
              $imageEl.attr('srcset', srcset);
              $imageEl.removeAttr('data-srcset');
            }
            if (sizes) {
              $imageEl.attr('sizes', sizes);
              $imageEl.removeAttr('data-sizes');
            }

            if ($pictureEl.length) {
              $pictureEl.children('source').each(function (sourceEl) {
                var $source = $(sourceEl);

                if ($source.attr('data-srcset')) {
                  $source.attr('srcset', $source.attr('data-srcset'));
                  $source.removeAttr('data-srcset');
                }
              });
            }

            if (src) {
              $imageEl.attr('src', src);
              $imageEl.removeAttr('data-src');
            }
          }

          $imageEl.addClass(params.loadedClass).removeClass(params.loadingClass);
          $slideEl.find("." + params.preloaderClass).remove();

          if (swiper.params.loop && loadInDuplicate) {
            var slideOriginalIndex = $slideEl.attr('data-swiper-slide-index');

            if ($slideEl.hasClass(swiper.params.slideDuplicateClass)) {
              var originalSlide = swiper.$wrapperEl.children("[data-swiper-slide-index=\"" +
slideOriginalIndex + "\"]:not(." + swiper.params.slideDuplicateClass + ")");
              swiper.lazy.loadInSlide(originalSlide.index(), false);
            } else {
              var duplicatedSlide = swiper.$wrapperEl.children("." +
swiper.params.slideDuplicateClass + "[data-swiper-slide-index=\"" + slideOriginalIndex + "\"]");
              swiper.lazy.loadInSlide(duplicatedSlide.index(), false);
            }
          }

          swiper.emit('lazyImageReady', $slideEl[0], $imageEl[0]);

          if (swiper.params.autoHeight) {
            swiper.updateAutoHeight();
          }
        });
        swiper.emit('lazyImageLoad', $slideEl[0], $imageEl[0]);
      });
    },
    load: function load() {
      var swiper = this;
      var $wrapperEl = swiper.$wrapperEl,
          swiperParams = swiper.params,
          slides = swiper.slides,
          activeIndex = swiper.activeIndex;
      var isVirtual = swiper.virtual && swiperParams.virtual.enabled;
      var params = swiperParams.lazy;
      var slidesPerView = swiperParams.slidesPerView;

      if (slidesPerView === 'auto') {
        slidesPerView = 0;
      }

      function slideExist(index) {
        if (isVirtual) {
          if ($wrapperEl.children("." + swiperParams.slideClass + "[data-swiper-slide-index=\"" +
index + "\"]").length) {
            return true;
          }
        } else if (slides[index]) return true;

        return false;
      }

      function slideIndex(slideEl) {
        if (isVirtual) {
          return $(slideEl).attr('data-swiper-slide-index');
        }

        return $(slideEl).index();
      }

      if (!swiper.lazy.initialImageLoaded) swiper.lazy.initialImageLoaded = true;

      if (swiper.params.watchSlidesVisibility) {
        $wrapperEl.children("." + swiperParams.slideVisibleClass).each(function (slideEl) {
          var index = isVirtual ? $(slideEl).attr('data-swiper-slide-index') : $(slideEl).index();
          swiper.lazy.loadInSlide(index);
        });
      } else if (slidesPerView > 1) {
        for (var i = activeIndex; i < activeIndex + slidesPerView; i += 1) {
          if (slideExist(i)) swiper.lazy.loadInSlide(i);
        }
      } else {
        swiper.lazy.loadInSlide(activeIndex);
      }

      if (params.loadPrevNext) {
        if (slidesPerView > 1 || params.loadPrevNextAmount && params.loadPrevNextAmount > 1) {
          var amount = params.loadPrevNextAmount;
          var spv = slidesPerView;
          var maxIndex = Math.min(activeIndex + spv + Math.max(amount, spv), slides.length);
          var minIndex = Math.max(activeIndex - Math.max(spv, amount), 0); // Next Slides

          for (var _i = activeIndex + slidesPerView; _i < maxIndex; _i += 1) {
            if (slideExist(_i)) swiper.lazy.loadInSlide(_i);
          } // Prev Slides


          for (var _i2 = minIndex; _i2 < activeIndex; _i2 += 1) {
            if (slideExist(_i2)) swiper.lazy.loadInSlide(_i2);
          }
        } else {
          var nextSlide = $wrapperEl.children("." + swiperParams.slideNextClass);
          if (nextSlide.length > 0) swiper.lazy.loadInSlide(slideIndex(nextSlide));
          var prevSlide = $wrapperEl.children("." + swiperParams.slidePrevClass);
          if (prevSlide.length > 0) swiper.lazy.loadInSlide(slideIndex(prevSlide));
        }
      }
    },
    checkInViewOnLoad: function checkInViewOnLoad() {
      var window = getWindow();
      var swiper = this;
      if (!swiper || swiper.destroyed) return;
      var $scrollElement = swiper.params.lazy.scrollingElement ?
$(swiper.params.lazy.scrollingElement) : $(window);
      var isWindow = $scrollElement[0] === window;
      var scrollElementWidth = isWindow ? window.innerWidth : $scrollElement[0].offsetWidth;
      var scrollElementHeight = isWindow ? window.innerHeight : $scrollElement[0].offsetHeight;
      var swiperOffset = swiper.$el.offset();
      var rtl = swiper.rtlTranslate;
      var inView = false;
      if (rtl) swiperOffset.left -= swiper.$el[0].scrollLeft;
      var swiperCoord = [[swiperOffset.left, swiperOffset.top], [swiperOffset.left + swiper.width,
swiperOffset.top], [swiperOffset.left, swiperOffset.top + swiper.height], [swiperOffset.left +
swiper.width, swiperOffset.top + swiper.height]];

      for (var i = 0; i < swiperCoord.length; i += 1) {
        var point = swiperCoord[i];

        if (point[0] >= 0 && point[0] <= scrollElementWidth && point[1] >= 0 && point[1] <=
scrollElementHeight) {
          if (point[0] === 0 && point[1] === 0) continue; // eslint-disable-line

          inView = true;
        }
      }

      var passiveListener = swiper.touchEvents.start === 'touchstart' &&
swiper.support.passiveListener && swiper.params.passiveListeners ? {
        passive: true,
        capture: false
      } : false;

      if (inView) {
        swiper.lazy.load();
        $scrollElement.off('scroll', swiper.lazy.checkInViewOnLoad, passiveListener);
      } else if (!swiper.lazy.scrollHandlerAttached) {
        swiper.lazy.scrollHandlerAttached = true;
        $scrollElement.on('scroll', swiper.lazy.checkInViewOnLoad, passiveListener);
      }
    }
  };
  var Lazy$1 = {
    name: 'lazy',
    params: {
      lazy: {
        checkInView: false,
        enabled: false,
        loadPrevNext: false,
        loadPrevNextAmount: 1,
        loadOnTransitionStart: false,
        scrollingElement: '',
        elementClass: 'swiper-lazy',
        loadingClass: 'swiper-lazy-loading',
        loadedClass: 'swiper-lazy-loaded',
        preloaderClass: 'swiper-lazy-preloader'
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        lazy: _extends({
          initialImageLoaded: false
        }, Lazy)
      });
    },
    on: {
      beforeInit: function beforeInit(swiper) {
        if (swiper.params.lazy.enabled && swiper.params.preloadImages) {
          swiper.params.preloadImages = false;
        }
      },
      init: function init(swiper) {
        if (swiper.params.lazy.enabled && !swiper.params.loop && swiper.params.initialSlide === 0) {
          if (swiper.params.lazy.checkInView) {
            swiper.lazy.checkInViewOnLoad();
          } else {
            swiper.lazy.load();
          }
        }
      },
      scroll: function scroll(swiper) {
        if (swiper.params.freeMode && !swiper.params.freeModeSticky) {
          swiper.lazy.load();
        }
      },
      'scrollbarDragMove resize _freeModeNoMomentumRelease': function lazyLoad(swiper) {
        if (swiper.params.lazy.enabled) {
          swiper.lazy.load();
        }
      },
      transitionStart: function transitionStart(swiper) {
        if (swiper.params.lazy.enabled) {
          if (swiper.params.lazy.loadOnTransitionStart || !swiper.params.lazy.loadOnTransitionStart
&& !swiper.lazy.initialImageLoaded) {
            swiper.lazy.load();
          }
        }
      },
      transitionEnd: function transitionEnd(swiper) {
        if (swiper.params.lazy.enabled && !swiper.params.lazy.loadOnTransitionStart) {
          swiper.lazy.load();
        }
      },
      slideChange: function slideChange(swiper) {
        var _swiper$params = swiper.params,
            lazy = _swiper$params.lazy,
            cssMode = _swiper$params.cssMode,
            watchSlidesVisibility = _swiper$params.watchSlidesVisibility,
            watchSlidesProgress = _swiper$params.watchSlidesProgress,
            touchReleaseOnEdges = _swiper$params.touchReleaseOnEdges,
            resistanceRatio = _swiper$params.resistanceRatio;

        if (lazy.enabled && (cssMode || (watchSlidesVisibility || watchSlidesProgress) &&
(touchReleaseOnEdges || resistanceRatio === 0))) {
          swiper.lazy.load();
        }
      }
    }
  };

  var Controller = {
    LinearSpline: function LinearSpline(x, y) {
      var binarySearch = function search() {
        var maxIndex;
        var minIndex;
        var guess;
        return function (array, val) {
          minIndex = -1;
          maxIndex = array.length;

          while (maxIndex - minIndex > 1) {
            guess = maxIndex + minIndex >> 1;

            if (array[guess] <= val) {
              minIndex = guess;
            } else {
              maxIndex = guess;
            }
          }
          return maxIndex;
        };
      }();

      this.x = x;
      this.y = y;
      this.lastIndex = x.length - 1; // Given an x value (x2), return the expected y2 value:
      // (x1,y1) is the known point before given value,
      // (x3,y3) is the known point after given value.

      var i1;
      var i3;

      this.interpolate = function interpolate(x2) {
        if (!x2) return 0; // Get the indexes of x1 and x3 (the array indexes before and after given
x2):

        i3 = binarySearch(this.x, x2);
        i1 = i3 - 1; // We have our indexes i1 & i3, so we can calculate already:
        // y2 := ((x2−x1) × (y3−y1)) ÷ (x3−x1) + y1

        return (x2 - this.x[i1]) * (this.y[i3] - this.y[i1]) / (this.x[i3] - this.x[i1]) +
this.y[i1];
      };

      return this;
    },
    // xxx: for now i will just save one spline function to to
    getInterpolateFunction: function getInterpolateFunction(c) {
      var swiper = this;

      if (!swiper.controller.spline) {
        swiper.controller.spline = swiper.params.loop ? new
Controller.LinearSpline(swiper.slidesGrid, c.slidesGrid) : new
Controller.LinearSpline(swiper.snapGrid, c.snapGrid);
      }
    },
    setTranslate: function setTranslate(_setTranslate, byController) {
      var swiper = this;
      var controlled = swiper.controller.control;
      var multiplier;
      var controlledTranslate;
      var Swiper = swiper.constructor;

      function setControlledTranslate(c) {
        // this will create an Interpolate function based on the snapGrids
        // x is the Grid of the scrolled scroller and y will be the controlled scroller
        // it makes sense to create this only once and recall it for the interpolation
        // the function does a lot of value caching for performance
        var translate = swiper.rtlTranslate ? -swiper.translate : swiper.translate;

        if (swiper.params.controller.by === 'slide') {
          swiper.controller.getInterpolateFunction(c); // i am not sure why the values have to be
multiplicated this way, tried to invert the snapGrid
          // but it did not work out

          controlledTranslate = -swiper.controller.spline.interpolate(-translate);
        }

        if (!controlledTranslate || swiper.params.controller.by === 'container') {
          multiplier = (c.maxTranslate() - c.minTranslate()) / (swiper.maxTranslate() -
swiper.minTranslate());
          controlledTranslate = (translate - swiper.minTranslate()) * multiplier + c.minTranslate();
        }
```

```
          if (swiper.params.controller.inverse) {
            controlledTranslate = c.maxTranslate() - controlledTranslate;
          }

          c.updateProgress(controlledTranslate);
          c.setTranslate(controlledTranslate, swiper);
          c.updateActiveIndex();
          c.updateSlidesClasses();
        }

        if (Array.isArray(controlled)) {
          for (var i = 0; i < controlled.length; i += 1) {
            if (controlled[i] !== byController && controlled[i] instanceof Swiper) {
              setControlledTranslate(controlled[i]);
            }
          }
        } else if (controlled instanceof Swiper && byController !== controlled) {
          setControlledTranslate(controlled);
        }
      },
      setTransition: function setTransition(duration, byController) {
        var swiper = this;
        var Swiper = swiper.constructor;
        var controlled = swiper.controller.control;
        var i;

        function setControlledTransition(c) {
          c.setTransition(duration, swiper);

          if (duration !== 0) {
            c.transitionStart();

            if (c.params.autoHeight) {
              nextTick(function () {
                c.updateAutoHeight();
              });
            }

            c.$wrapperEl.transitionEnd(function () {
              if (!controlled) return;

              if (c.params.loop && swiper.params.controller.by === 'slide') {
                c.loopFix();
              }

              c.transitionEnd();
            });
          }
        }

        if (Array.isArray(controlled)) {
          for (i = 0; i < controlled.length; i += 1) {
            if (controlled[i] !== byController && controlled[i] instanceof Swiper) {
              setControlledTransition(controlled[i]);
            }
          }
        } else if (controlled instanceof Swiper && byController !== controlled) {
          setControlledTransition(controlled);
        }
      }
    };
    var Controller$1 = {
      name: 'controller',
      params: {
```

```
      controller: {
        control: undefined,
        inverse: false,
        by: 'slide' // or 'container'

      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        controller: _extends({
          control: swiper.params.controller.control
        }, Controller)
      });
    },
    on: {
      update: function update(swiper) {
        if (!swiper.controller.control) return;

        if (swiper.controller.spline) {
          swiper.controller.spline = undefined;
          delete swiper.controller.spline;
        }
      },
      resize: function resize(swiper) {
        if (!swiper.controller.control) return;

        if (swiper.controller.spline) {
          swiper.controller.spline = undefined;
          delete swiper.controller.spline;
        }
      },
      observerUpdate: function observerUpdate(swiper) {
        if (!swiper.controller.control) return;

        if (swiper.controller.spline) {
          swiper.controller.spline = undefined;
          delete swiper.controller.spline;
        }
      },
      setTranslate: function setTranslate(swiper, translate, byController) {
        if (!swiper.controller.control) return;
        swiper.controller.setTranslate(translate, byController);
      },
      setTransition: function setTransition(swiper, duration, byController) {
        if (!swiper.controller.control) return;
        swiper.controller.setTransition(duration, byController);
      }
    }
  };

  var A11y = {
    getRandomNumber: function getRandomNumber(size) {
      if (size === void 0) {
        size = 16;
      }

      var randomChar = function randomChar() {
        return Math.round(16 * Math.random()).toString(16);
      };

      return 'x'.repeat(size).replace(/x/g, randomChar);
    },
    makeElFocusable: function makeElFocusable($el) {
      $el.attr('tabIndex', '0');
      return $el;
    },
    makeElNotFocusable: function makeElNotFocusable($el) {
      $el.attr('tabIndex', '-1');
      return $el;
    },
    addElRole: function addElRole($el, role) {
      $el.attr('role', role);
      return $el;
    },
    addElRoleDescription: function addElRoleDescription($el, description) {
      $el.attr('aria-roledescription', description);
      return $el;
    },
    addElControls: function addElControls($el, controls) {
      $el.attr('aria-controls', controls);
      return $el;
    },
    addElLabel: function addElLabel($el, label) {
      $el.attr('aria-label', label);
      return $el;
    },
    addElId: function addElId($el, id) {
      $el.attr('id', id);
      return $el;
    },
    addElLive: function addElLive($el, live) {
      $el.attr('aria-live', live);
      return $el;
    },
    disableEl: function disableEl($el) {
      $el.attr('aria-disabled', true);
      return $el;
    },
    enableEl: function enableEl($el) {
      $el.attr('aria-disabled', false);
      return $el;
    },
    onEnterOrSpaceKey: function onEnterOrSpaceKey(e) {
      if (e.keyCode !== 13 && e.keyCode !== 32) return;
      var swiper = this;
      var params = swiper.params.a11y;
      var $targetEl = $(e.target);

      if (swiper.navigation && swiper.navigation.$nextEl && $targetEl.is(swiper.navigation.$nextEl))
{
        if (!(swiper.isEnd && !swiper.params.loop)) {
          swiper.slideNext();
        }

        if (swiper.isEnd) {
          swiper.a11y.notify(params.lastSlideMessage);
        } else {
          swiper.a11y.notify(params.nextSlideMessage);
        }
      }

      if (swiper.navigation && swiper.navigation.$prevEl && $targetEl.is(swiper.navigation.$prevEl))
{
        if (!(swiper.isBeginning && !swiper.params.loop)) {
          swiper.slidePrev();
        }

        if (swiper.isBeginning) {
          swiper.a11y.notify(params.firstSlideMessage);
        } else {
          swiper.a11y.notify(params.prevSlideMessage);
        }
      }

      if (swiper.pagination && $targetEl.is(classesToSelector(swiper.params.pagination.bulletClass)))
{
        $targetEl[0].click();
      }
    },
    notify: function notify(message) {
      var swiper = this;
      var notification = swiper.a11y.liveRegion;
      if (notification.length === 0) return;
      notification.html('');
      notification.html(message);
    },
    updateNavigation: function updateNavigation() {
      var swiper = this;
      if (swiper.params.loop || !swiper.navigation) return;
      var _swiper$navigation = swiper.navigation,
          $nextEl = _swiper$navigation.$nextEl,
          $prevEl = _swiper$navigation.$prevEl;

      if ($prevEl && $prevEl.length > 0) {
        if (swiper.isBeginning) {
          swiper.a11y.disableEl($prevEl);
          swiper.a11y.makeElNotFocusable($prevEl);
        } else {
          swiper.a11y.enableEl($prevEl);
          swiper.a11y.makeElFocusable($prevEl);
        }
      }

      if ($nextEl && $nextEl.length > 0) {
        if (swiper.isEnd) {
          swiper.a11y.disableEl($nextEl);
          swiper.a11y.makeElNotFocusable($nextEl);
        } else {
          swiper.a11y.enableEl($nextEl);
          swiper.a11y.makeElFocusable($nextEl);
        }
      }
    },
    updatePagination: function updatePagination() {
      var swiper = this;
      var params = swiper.params.a11y;

      if (swiper.pagination && swiper.params.pagination.clickable && swiper.pagination.bullets &&
swiper.pagination.bullets.length) {
        swiper.pagination.bullets.each(function (bulletEl) {
          var $bulletEl = $(bulletEl);
          swiper.a11y.makeElFocusable($bulletEl);

          if (!swiper.params.pagination.renderBullet) {
            swiper.a11y.addElRole($bulletEl, 'button');
            swiper.a11y.addElLabel($bulletEl, params.paginationBulletMessage.replace(/\{\{index\}\}/,
$bulletEl.index() + 1));
          }
        });
      }
    },
    init: function init() {
      var swiper = this;
      var params = swiper.params.a11y;
      swiper.$el.append(swiper.a11y.liveRegion); // Container

      var $containerEl = swiper.$el;

      if (params.containerRoleDescriptionMessage) {
        swiper.a11y.addElRoleDescription($containerEl, params.containerRoleDescriptionMessage);
      }

      if (params.containerMessage) {
        swiper.a11y.addElLabel($containerEl, params.containerMessage);
      } // Wrapper


      var $wrapperEl = swiper.$wrapperEl;
      var wrapperId = $wrapperEl.attr('id') || "swiper-wrapper-" + swiper.a11y.getRandomNumber(16);
      var live = swiper.params.autoplay && swiper.params.autoplay.enabled ? 'off' : 'polite';
      swiper.a11y.addElId($wrapperEl, wrapperId);
      swiper.a11y.addElLive($wrapperEl, live); // Slide

      if (params.itemRoleDescriptionMessage) {
        swiper.a11y.addElRoleDescription($(swiper.slides), params.itemRoleDescriptionMessage);
      }

      swiper.a11y.addElRole($(swiper.slides), params.slideRole);
      var slidesLength = swiper.params.loop ? swiper.slides.filter(function (el) {
        return !el.classList.contains(swiper.params.slideDuplicateClass);
      }).length : swiper.slides.length;
      swiper.slides.each(function (slideEl, index) {
        var $slideEl = $(slideEl);
        var slideIndex = swiper.params.loop ? parseInt($slideEl.attr('data-swiper-slide-index'), 10)
: index;
        var ariaLabelMessage = params.slideLabelMessage.replace(/\{\{index\}\}/, slideIndex +
1).replace(/\{\{slidesLength\}\}/, slidesLength);
        swiper.a11y.addElLabel($slideEl, ariaLabelMessage);
      }); // Navigation

      var $nextEl;
      var $prevEl;

      if (swiper.navigation && swiper.navigation.$nextEl) {
        $nextEl = swiper.navigation.$nextEl;
      }

      if (swiper.navigation && swiper.navigation.$prevEl) {
        $prevEl = swiper.navigation.$prevEl;
      }

      if ($nextEl && $nextEl.length) {
        swiper.a11y.makeElFocusable($nextEl);

        if ($nextEl[0].tagName !== 'BUTTON') {
          swiper.a11y.addElRole($nextEl, 'button');
          $nextEl.on('keydown', swiper.a11y.onEnterOrSpaceKey);
        }

        swiper.a11y.addElLabel($nextEl, params.nextSlideMessage);
        swiper.a11y.addElControls($nextEl, wrapperId);
      }

      if ($prevEl && $prevEl.length) {
        swiper.a11y.makeElFocusable($prevEl);

        if ($prevEl[0].tagName !== 'BUTTON') {
          swiper.a11y.addElRole($prevEl, 'button');
          $prevEl.on('keydown', swiper.a11y.onEnterOrSpaceKey);
        }

        swiper.a11y.addElLabel($prevEl, params.prevSlideMessage);
        swiper.a11y.addElControls($prevEl, wrapperId);
      } // Pagination


      if (swiper.pagination && swiper.params.pagination.clickable && swiper.pagination.bullets &&
swiper.pagination.bullets.length) {
        swiper.pagination.$el.on('keydown', classesToSelector(swiper.params.pagination.bulletClass),
swiper.a11y.onEnterOrSpaceKey);
      }
    },
    destroy: function destroy() {
      var swiper = this;
      if (swiper.a11y.liveRegion && swiper.a11y.liveRegion.length > 0)
swiper.a11y.liveRegion.remove();
      var $nextEl;
      var $prevEl;

      if (swiper.navigation && swiper.navigation.$nextEl) {
        $nextEl = swiper.navigation.$nextEl;
      }

      if (swiper.navigation && swiper.navigation.$prevEl) {
        $prevEl = swiper.navigation.$prevEl;
      }

      if ($nextEl) {
        $nextEl.off('keydown', swiper.a11y.onEnterOrSpaceKey);
      }

      if ($prevEl) {
        $prevEl.off('keydown', swiper.a11y.onEnterOrSpaceKey);
      } // Pagination


      if (swiper.pagination && swiper.params.pagination.clickable && swiper.pagination.bullets &&
swiper.pagination.bullets.length) {
        swiper.pagination.$el.off('keydown', classesToSelector(swiper.params.pagination.bulletClass),
swiper.a11y.onEnterOrSpaceKey);
      }
    }
  };
  var A11y$1 = {
    name: 'a11y',
    params: {
      a11y: {
        enabled: true,
        notificationClass: 'swiper-notification',
        prevSlideMessage: 'Previous slide',
        nextSlideMessage: 'Next slide',
        firstSlideMessage: 'This is the first slide',
        lastSlideMessage: 'This is the last slide',
        paginationBulletMessage: 'Go to slide {{index}}',
        slideLabelMessage: '{{index}} / {{slidesLength}}',
        containerMessage: null,
        containerRoleDescriptionMessage: null,
        itemRoleDescriptionMessage: null,
        slideRole: 'group'
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        a11y: _extends({}, A11y, {
          liveRegion: $("<span class=\"" + swiper.params.a11y.notificationClass + "\" aria-
live=\"assertive\" aria-atomic=\"true\"></span>")
        })
      });
    },
    on: {
      afterInit: function afterInit(swiper) {
        if (!swiper.params.a11y.enabled) return;
        swiper.a11y.init();
        swiper.a11y.updateNavigation();
      },
      toEdge: function toEdge(swiper) {
        if (!swiper.params.a11y.enabled) return;
        swiper.a11y.updateNavigation();
      },
      fromEdge: function fromEdge(swiper) {
        if (!swiper.params.a11y.enabled) return;
        swiper.a11y.updateNavigation();
      },
      paginationUpdate: function paginationUpdate(swiper) {
        if (!swiper.params.a11y.enabled) return;
        swiper.a11y.updatePagination();
      },
      destroy: function destroy(swiper) {
        if (!swiper.params.a11y.enabled) return;
        swiper.a11y.destroy();
      }
    }
  };

  var History = {
    init: function init() {
      var swiper = this;
      var window = getWindow();
      if (!swiper.params.history) return;

      if (!window.history || !window.history.pushState) {
        swiper.params.history.enabled = false;
        swiper.params.hashNavigation.enabled = true;
        return;
      }

      var history = swiper.history;
      history.initialized = true;
      history.paths = History.getPathValues(swiper.params.url);
      if (!history.paths.key && !history.paths.value) return;
      history.scrollToSlide(0, history.paths.value, swiper.params.runCallbacksOnInit);

      if (!swiper.params.history.replaceState) {
        window.addEventListener('popstate', swiper.history.setHistoryPopState);
      }
    },
    destroy: function destroy() {
      var swiper = this;
      var window = getWindow();

      if (!swiper.params.history.replaceState) {
        window.removeEventListener('popstate', swiper.history.setHistoryPopState);
      }
    },
    setHistoryPopState: function setHistoryPopState() {
      var swiper = this;
      swiper.history.paths = History.getPathValues(swiper.params.url);
      swiper.history.scrollToSlide(swiper.params.speed, swiper.history.paths.value, false);
    },
    getPathValues: function getPathValues(urlOverride) {
      var window = getWindow();
      var location;

      if (urlOverride) {
        location = new URL(urlOverride);
      } else {
        location = window.location;
      }

      var pathArray = location.pathname.slice(1).split('/').filter(function (part) {
        return part !== '';
      });
      var total = pathArray.length;
      var key = pathArray[total - 2];
      var value = pathArray[total - 1];
      return {
        key: key,
        value: value
      };
    },
    setHistory: function setHistory(key, index) {
      var swiper = this;
      var window = getWindow();
      if (!swiper.history.initialized || !swiper.params.history.enabled) return;
      var location;

      if (swiper.params.url) {
        location = new URL(swiper.params.url);
      } else {
        location = window.location;
      }

      var slide = swiper.slides.eq(index);
      var value = History.slugify(slide.attr('data-history'));

      if (swiper.params.history.root.length > 0) {
        var root = swiper.params.history.root;
        if (root[root.length - 1] === '/') root = root.slice(0, root.length - 1);
        value = root + "/" + key + "/" + value;
      } else if (!location.pathname.includes(key)) {
        value = key + "/" + value;
      }

      var currentState = window.history.state;

      if (currentState && currentState.value === value) {
        return;
      }

      if (swiper.params.history.replaceState) {
        window.history.replaceState({
          value: value
        }, null, value);
      } else {
        window.history.pushState({
          value: value
        }, null, value);
      }
    },
    slugify: function slugify(text) {
      return text.toString().replace(/\s+/g, '-').replace(/[^\w-]+/g, '').replace(/--+/g, '-
').replace(/^-+/, '').replace(/-+$/, '');
    },
    scrollToSlide: function scrollToSlide(speed, value, runCallbacks) {
      var swiper = this;

      if (value) {
        for (var i = 0, length = swiper.slides.length; i < length; i += 1) {
          var slide = swiper.slides.eq(i);
          var slideHistory = History.slugify(slide.attr('data-history'));

          if (slideHistory === value && !slide.hasClass(swiper.params.slideDuplicateClass)) {
            var index = slide.index();
            swiper.slideTo(index, speed, runCallbacks);
          }
        }
      } else {
        swiper.slideTo(0, speed, runCallbacks);
      }
    }
  };
  var History$1 = {
    name: 'history',
    params: {
      history: {
        enabled: false,
        root: '',
        replaceState: false,
        key: 'slides'
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        history: _extends({}, History)
      });
    },
    on: {
      init: function init(swiper) {
        if (swiper.params.history.enabled) {
          swiper.history.init();
        }
      },
      destroy: function destroy(swiper) {
        if (swiper.params.history.enabled) {
          swiper.history.destroy();
        }
      },
      'transitionEnd _freeModeNoMomentumRelease': function
transitionEnd_freeModeNoMomentumRelease(swiper) {
        if (swiper.history.initialized) {
          swiper.history.setHistory(swiper.params.history.key, swiper.activeIndex);
        }
      },
      slideChange: function slideChange(swiper) {
        if (swiper.history.initialized && swiper.params.cssMode) {
          swiper.history.setHistory(swiper.params.history.key, swiper.activeIndex);
        }
      }
    }
  };

  var HashNavigation = {
    onHashChange: function onHashChange() {
      var swiper = this;
      var document = getDocument();
      swiper.emit('hashChange');
      var newHash = document.location.hash.replace('#', '');
      var activeSlideHash = swiper.slides.eq(swiper.activeIndex).attr('data-hash');

      if (newHash !== activeSlideHash) {
        var newIndex = swiper.$wrapperEl.children("." + swiper.params.slideClass + "[data-hash=\"" +
newHash + "\"]").index();
        if (typeof newIndex === 'undefined') return;
        swiper.slideTo(newIndex);
      }
    },
    setHash: function setHash() {
      var swiper = this;
      var window = getWindow();
      var document = getDocument();
      if (!swiper.hashNavigation.initialized || !swiper.params.hashNavigation.enabled) return;

      if (swiper.params.hashNavigation.replaceState && window.history && window.history.replaceState)
{
        window.history.replaceState(null, null, "#" +
swiper.slides.eq(swiper.activeIndex).attr('data-hash') || '');
        swiper.emit('hashSet');
      } else {
        var slide = swiper.slides.eq(swiper.activeIndex);
        var hash = slide.attr('data-hash') || slide.attr('data-history');
        document.location.hash = hash || '';
        swiper.emit('hashSet');
      }
    },
    init: function init() {
      var swiper = this;
      var document = getDocument();
      var window = getWindow();
      if (!swiper.params.hashNavigation.enabled || swiper.params.history &&
swiper.params.history.enabled) return;
      swiper.hashNavigation.initialized = true;
      var hash = document.location.hash.replace('#', '');

      if (hash) {
        var speed = 0;

        for (var i = 0, length = swiper.slides.length; i < length; i += 1) {
          var slide = swiper.slides.eq(i);
          var slideHash = slide.attr('data-hash') || slide.attr('data-history');

          if (slideHash === hash && !slide.hasClass(swiper.params.slideDuplicateClass)) {
            var index = slide.index();
            swiper.slideTo(index, speed, swiper.params.runCallbacksOnInit, true);
          }
        }
      }

      if (swiper.params.hashNavigation.watchState) {
        $(window).on('hashchange', swiper.hashNavigation.onHashChange);
      }
    },
    destroy: function destroy() {
      var swiper = this;
      var window = getWindow();

      if (swiper.params.hashNavigation.watchState) {
        $(window).off('hashchange', swiper.hashNavigation.onHashChange);
      }
    }
  };
  var HashNavigation$1 = {
    name: 'hash-navigation',
    params: {
      hashNavigation: {
        enabled: false,
        replaceState: false,
        watchState: false
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        hashNavigation: _extends({
          initialized: false
        }, HashNavigation)
      });
    },
    on: {
      init: function init(swiper) {
        if (swiper.params.hashNavigation.enabled) {
          swiper.hashNavigation.init();
        }
      },
      destroy: function destroy(swiper) {
        if (swiper.params.hashNavigation.enabled) {
          swiper.hashNavigation.destroy();
        }
      },
      'transitionEnd _freeModeNoMomentumRelease': function
transitionEnd_freeModeNoMomentumRelease(swiper) {
        if (swiper.hashNavigation.initialized) {
          swiper.hashNavigation.setHash();
        }
      },
      slideChange: function slideChange(swiper) {
        if (swiper.hashNavigation.initialized && swiper.params.cssMode) {
          swiper.hashNavigation.setHash();
        }
      }
    }
  };

  var Autoplay = {
    run: function run() {
      var swiper = this;
      var $activeSlideEl = swiper.slides.eq(swiper.activeIndex);
      var delay = swiper.params.autoplay.delay;

      if ($activeSlideEl.attr('data-swiper-autoplay')) {
        delay = $activeSlideEl.attr('data-swiper-autoplay') || swiper.params.autoplay.delay;
      }

      clearTimeout(swiper.autoplay.timeout);
      swiper.autoplay.timeout = nextTick(function () {
        var autoplayResult;

        if (swiper.params.autoplay.reverseDirection) {
          if (swiper.params.loop) {
            swiper.loopFix();
            autoplayResult = swiper.slidePrev(swiper.params.speed, true, true);
            swiper.emit('autoplay');
          } else if (!swiper.isBeginning) {
            autoplayResult = swiper.slidePrev(swiper.params.speed, true, true);
            swiper.emit('autoplay');
          } else if (!swiper.params.autoplay.stopOnLastSlide) {
            autoplayResult = swiper.slideTo(swiper.slides.length - 1, swiper.params.speed, true,
true);
```

```
            swiper.emit('autoplay');
          } else {
            swiper.autoplay.stop();
          }
        } else if (swiper.params.loop) {
          swiper.loopFix();
          autoplayResult = swiper.slideNext(swiper.params.speed, true, true);
          swiper.emit('autoplay');
        } else if (!swiper.isEnd) {
          autoplayResult = swiper.slideNext(swiper.params.speed, true, true);
          swiper.emit('autoplay');
        } else if (!swiper.params.autoplay.stopOnLastSlide) {
          autoplayResult = swiper.slideTo(0, swiper.params.speed, true, true);
          swiper.emit('autoplay');
        } else {
          swiper.autoplay.stop();
        }

        if (swiper.params.cssMode && swiper.autoplay.running) swiper.autoplay.run();else if
(autoplayResult === false) {
          swiper.autoplay.run();
        }
      }, delay);
    },
    start: function start() {
      var swiper = this;
      if (typeof swiper.autoplay.timeout !== 'undefined') return false;
      if (swiper.autoplay.running) return false;
      swiper.autoplay.running = true;
      swiper.emit('autoplayStart');
      swiper.autoplay.run();
      return true;
    },
    stop: function stop() {
      var swiper = this;
      if (!swiper.autoplay.running) return false;
      if (typeof swiper.autoplay.timeout === 'undefined') return false;

      if (swiper.autoplay.timeout) {
        clearTimeout(swiper.autoplay.timeout);
        swiper.autoplay.timeout = undefined;
      }

      swiper.autoplay.running = false;
      swiper.emit('autoplayStop');
      return true;
    },
    pause: function pause(speed) {
      var swiper = this;
      if (!swiper.autoplay.running) return;
      if (swiper.autoplay.paused) return;
      if (swiper.autoplay.timeout) clearTimeout(swiper.autoplay.timeout);
      swiper.autoplay.paused = true;

      if (speed === 0 || !swiper.params.autoplay.waitForTransition) {
        swiper.autoplay.paused = false;
        swiper.autoplay.run();
      } else {
        ['transitionend', 'webkitTransitionEnd'].forEach(function (event) {
          swiper.$wrapperEl[0].addEventListener(event, swiper.autoplay.onTransitionEnd);
        });
      }
    },
    onVisibilityChange: function onVisibilityChange() {
      var swiper = this;
```

```
        var document = getDocument();

        if (document.visibilityState === 'hidden' && swiper.autoplay.running) {
          swiper.autoplay.pause();
        }

        if (document.visibilityState === 'visible' && swiper.autoplay.paused) {
          swiper.autoplay.run();
          swiper.autoplay.paused = false;
        }
      },
      onTransitionEnd: function onTransitionEnd(e) {
        var swiper = this;
        if (!swiper || swiper.destroyed || !swiper.$wrapperEl) return;
        if (e.target !== swiper.$wrapperEl[0]) return;
        ['transitionend', 'webkitTransitionEnd'].forEach(function (event) {
          swiper.$wrapperEl[0].removeEventListener(event, swiper.autoplay.onTransitionEnd);
        });
        swiper.autoplay.paused = false;

        if (!swiper.autoplay.running) {
          swiper.autoplay.stop();
        } else {
          swiper.autoplay.run();
        }
      },
      onMouseEnter: function onMouseEnter() {
        var swiper = this;

        if (swiper.params.autoplay.disableOnInteraction) {
          swiper.autoplay.stop();
        } else {
          swiper.autoplay.pause();
        }

        ['transitionend', 'webkitTransitionEnd'].forEach(function (event) {
          swiper.$wrapperEl[0].removeEventListener(event, swiper.autoplay.onTransitionEnd);
        });
      },
      onMouseLeave: function onMouseLeave() {
        var swiper = this;

        if (swiper.params.autoplay.disableOnInteraction) {
          return;
        }

        swiper.autoplay.paused = false;
        swiper.autoplay.run();
      },
      attachMouseEvents: function attachMouseEvents() {
        var swiper = this;

        if (swiper.params.autoplay.pauseOnMouseEnter) {
          swiper.$el.on('mouseenter', swiper.autoplay.onMouseEnter);
          swiper.$el.on('mouseleave', swiper.autoplay.onMouseLeave);
        }
      },
      detachMouseEvents: function detachMouseEvents() {
        var swiper = this;
        swiper.$el.off('mouseenter', swiper.autoplay.onMouseEnter);
        swiper.$el.off('mouseleave', swiper.autoplay.onMouseLeave);
      }
    };
    var Autoplay$1 = {
      name: 'autoplay',
      params: {
        autoplay: {
          enabled: false,
          delay: 3000,
          waitForTransition: true,
          disableOnInteraction: true,
          stopOnLastSlide: false,
          reverseDirection: false,
          pauseOnMouseEnter: false
        }
      },
      create: function create() {
        var swiper = this;
        bindModuleMethods(swiper, {
          autoplay: _extends({}, Autoplay, {
            running: false,
            paused: false
          })
        });
      },
      on: {
        init: function init(swiper) {
          if (swiper.params.autoplay.enabled) {
            swiper.autoplay.start();
            var document = getDocument();
            document.addEventListener('visibilitychange', swiper.autoplay.onVisibilityChange);
            swiper.autoplay.attachMouseEvents();
          }
        },
        beforeTransitionStart: function beforeTransitionStart(swiper, speed, internal) {
          if (swiper.autoplay.running) {
            if (internal || !swiper.params.autoplay.disableOnInteraction) {
              swiper.autoplay.pause(speed);
            } else {
              swiper.autoplay.stop();
            }
          }
        },
        sliderFirstMove: function sliderFirstMove(swiper) {
          if (swiper.autoplay.running) {
            if (swiper.params.autoplay.disableOnInteraction) {
              swiper.autoplay.stop();
            } else {
              swiper.autoplay.pause();
            }
          }
        },
        touchEnd: function touchEnd(swiper) {
          if (swiper.params.cssMode && swiper.autoplay.paused &&
  !swiper.params.autoplay.disableOnInteraction) {
            swiper.autoplay.run();
          }
        },
        destroy: function destroy(swiper) {
          swiper.autoplay.detachMouseEvents();

          if (swiper.autoplay.running) {
            swiper.autoplay.stop();
          }

          var document = getDocument();
          document.removeEventListener('visibilitychange', swiper.autoplay.onVisibilityChange);
        }
      }
    };
    var Fade = {
      setTranslate: function setTranslate() {
        var swiper = this;
        var slides = swiper.slides;

        for (var i = 0; i < slides.length; i += 1) {
          var $slideEl = swiper.slides.eq(i);
          var offset = $slideEl[0].swiperSlideOffset;
          var tx = -offset;
          if (!swiper.params.virtualTranslate) tx -= swiper.translate;
          var ty = 0;

          if (!swiper.isHorizontal()) {
            ty = tx;
            tx = 0;
          }

          var slideOpacity = swiper.params.fadeEffect.crossFade ? Math.max(1 -
  Math.abs($slideEl[0].progress), 0) : 1 + Math.min(Math.max($slideEl[0].progress, -1), 0);
          $slideEl.css({
            opacity: slideOpacity
          }).transform("translate3d(" + tx + "px, " + ty + "px, 0px)");
        }
      },
      setTransition: function setTransition(duration) {
        var swiper = this;
        var slides = swiper.slides,
            $wrapperEl = swiper.$wrapperEl;
        slides.transition(duration);

        if (swiper.params.virtualTranslate && duration !== 0) {
          var eventTriggered = false;
          slides.transitionEnd(function () {
            if (eventTriggered) return;
            if (!swiper || swiper.destroyed) return;
            eventTriggered = true;
            swiper.animating = false;
            var triggerEvents = ['webkitTransitionEnd', 'transitionend'];

            for (var i = 0; i < triggerEvents.length; i += 1) {
              $wrapperEl.trigger(triggerEvents[i]);
            }
          });
        }
      }
    };
    var EffectFade = {
      name: 'effect-fade',
      params: {
        fadeEffect: {
          crossFade: false
        }
      },
      create: function create() {
        var swiper = this;
        bindModuleMethods(swiper, {
          fadeEffect: _extends({}, Fade)
        });
      },
      on: {
        beforeInit: function beforeInit(swiper) {
          if (swiper.params.effect !== 'fade') return;
          swiper.classNames.push(swiper.params.containerModifierClass + "fade");
          var overwriteParams = {
            slidesPerView: 1,
            slidesPerColumn: 1,
            slidesPerGroup: 1,
            watchSlidesProgress: true,
            spaceBetween: 0,
            virtualTranslate: true
          };
          extend(swiper.params, overwriteParams);
          extend(swiper.originalParams, overwriteParams);
        },
        setTranslate: function setTranslate(swiper) {
          if (swiper.params.effect !== 'fade') return;
          swiper.fadeEffect.setTranslate();
        },
        setTransition: function setTransition(swiper, duration) {
          if (swiper.params.effect !== 'fade') return;
          swiper.fadeEffect.setTransition(duration);
        }
      }
    };

    var Cube = {
      setTranslate: function setTranslate() {
        var swiper = this;
        var $el = swiper.$el,
            $wrapperEl = swiper.$wrapperEl,
            slides = swiper.slides,
            swiperWidth = swiper.width,
            swiperHeight = swiper.height,
            rtl = swiper.rtlTranslate,
            swiperSize = swiper.size,
            browser = swiper.browser;
        var params = swiper.params.cubeEffect;
        var isHorizontal = swiper.isHorizontal();
        var isVirtual = swiper.virtual && swiper.params.virtual.enabled;
        var wrapperRotate = 0;
        var $cubeShadowEl;

        if (params.shadow) {
          if (isHorizontal) {
            $cubeShadowEl = $wrapperEl.find('.swiper-cube-shadow');

            if ($cubeShadowEl.length === 0) {
              $cubeShadowEl = $('<div class="swiper-cube-shadow"></div>');
              $wrapperEl.append($cubeShadowEl);
            }

            $cubeShadowEl.css({
              height: swiperWidth + "px"
            });
          } else {
            $cubeShadowEl = $el.find('.swiper-cube-shadow');

            if ($cubeShadowEl.length === 0) {
              $cubeShadowEl = $('<div class="swiper-cube-shadow"></div>');
              $el.append($cubeShadowEl);
            }
          }
        }

        for (var i = 0; i < slides.length; i += 1) {
          var $slideEl = slides.eq(i);
          var slideIndex = i;

          if (isVirtual) {
            slideIndex = parseInt($slideEl.attr('data-swiper-slide-index'), 10);
          }

          var slideAngle = slideIndex * 90;
          var round = Math.floor(slideAngle / 360);

          if (rtl) {
            slideAngle = -slideAngle;
            round = Math.floor(-slideAngle / 360);
          }

          var progress = Math.max(Math.min($slideEl[0].progress, 1), -1);
          var tx = 0;
          var ty = 0;
          var tz = 0;

          if (slideIndex % 4 === 0) {
            tx = -round * 4 * swiperSize;
            tz = 0;
          } else if ((slideIndex - 1) % 4 === 0) {
            tx = 0;
            tz = -round * 4 * swiperSize;
          } else if ((slideIndex - 2) % 4 === 0) {
            tx = swiperSize + round * 4 * swiperSize;
            tz = swiperSize;
          } else if ((slideIndex - 3) % 4 === 0) {
            tx = -swiperSize;
            tz = 3 * swiperSize + swiperSize * 4 * round;
          }

          if (rtl) {
            tx = -tx;
          }

          if (!isHorizontal) {
            ty = tx;
            tx = 0;
          }

          var transform = "rotateX(" + (isHorizontal ? 0 : -slideAngle) + "deg) rotateY(" +
  (isHorizontal ? slideAngle : 0) + "deg) translate3d(" + tx + "px, " + ty + "px, " + tz + "px)";

          if (progress <= 1 && progress > -1) {
            wrapperRotate = slideIndex * 90 + progress * 90;
            if (rtl) wrapperRotate = -slideIndex * 90 - progress * 90;
          }

          $slideEl.transform(transform);

          if (params.slideShadows) {
            // Set shadows
            var shadowBefore = isHorizontal ? $slideEl.find('.swiper-slide-shadow-left') :
  $slideEl.find('.swiper-slide-shadow-top');
            var shadowAfter = isHorizontal ? $slideEl.find('.swiper-slide-shadow-right') :
  $slideEl.find('.swiper-slide-shadow-bottom');

            if (shadowBefore.length === 0) {
              shadowBefore = $("<div class=\"swiper-slide-shadow-" + (isHorizontal ? 'left' : 'top') +
  "\"></div>");
              $slideEl.append(shadowBefore);
            }

            if (shadowAfter.length === 0) {
              shadowAfter = $("<div class=\"swiper-slide-shadow-" + (isHorizontal ? 'right' : 'bottom')
  + "\"></div>");
```

```
            $slideEl.append(shadowAfter);
          }

          if (shadowBefore.length) shadowBefore[0].style.opacity = Math.max(-progress, 0);
          if (shadowAfter.length) shadowAfter[0].style.opacity = Math.max(progress, 0);
        }
      }

      $wrapperEl.css({
        '-webkit-transform-origin': "50% 50% -" + swiperSize / 2 + "px",
        '-moz-transform-origin': "50% 50% -" + swiperSize / 2 + "px",
        '-ms-transform-origin': "50% 50% -" + swiperSize / 2 + "px",
        'transform-origin': "50% 50% -" + swiperSize / 2 + "px"
      });

      if (params.shadow) {
        if (isHorizontal) {
          $cubeShadowEl.transform("translate3d(0px, " + (swiperWidth / 2 + params.shadowOffset) +
"px, " + -swiperWidth / 2 + "px) rotateX(90deg) rotateZ(0deg) scale(" + params.shadowScale + ")");
        } else {
          var shadowAngle = Math.abs(wrapperRotate) - Math.floor(Math.abs(wrapperRotate) / 90) * 90;
          var multiplier = 1.5 - (Math.sin(shadowAngle * 2 * Math.PI / 360) / 2 +
Math.cos(shadowAngle * 2 * Math.PI / 360) / 2);
          var scale1 = params.shadowScale;
          var scale2 = params.shadowScale / multiplier;
          var offset = params.shadowOffset;
          $cubeShadowEl.transform("scale3d(" + scale1 + ", 1, " + scale2 + ") translate3d(0px, " +
(swiperHeight / 2 + offset) + "px, " + -swiperHeight / 2 / scale2 + "px) rotateX(-90deg)");
        }
      }

      var zFactor = browser.isSafari || browser.isWebView ? -swiperSize / 2 : 0;
      $wrapperEl.transform("translate3d(0px,0," + zFactor + "px) rotateX(" + (swiper.isHorizontal() ?
0 : wrapperRotate) + "deg) rotateY(" + (swiper.isHorizontal() ? -wrapperRotate : 0) + "deg)");
    },
    setTransition: function setTransition(duration) {
      var swiper = this;
      var $el = swiper.$el,
          slides = swiper.slides;
      slides.transition(duration).find('.swiper-slide-shadow-top, .swiper-slide-shadow-right,
.swiper-slide-shadow-bottom, .swiper-slide-shadow-left').transition(duration);

      if (swiper.params.cubeEffect.shadow && !swiper.isHorizontal()) {
        $el.find('.swiper-cube-shadow').transition(duration);
      }
    }
  };
  var EffectCube = {
    name: 'effect-cube',
    params: {
      cubeEffect: {
        slideShadows: true,
        shadow: true,
        shadowOffset: 20,
        shadowScale: 0.94
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        cubeEffect: _extends({}, Cube)
      });
    },
    on: {
      beforeInit: function beforeInit(swiper) {
        if (swiper.params.effect !== 'cube') return;
        swiper.classNames.push(swiper.params.containerModifierClass + "cube");
        swiper.classNames.push(swiper.params.containerModifierClass + "3d");
        var overwriteParams = {
          slidesPerView: 1,
          slidesPerColumn: 1,
          slidesPerGroup: 1,
          watchSlidesProgress: true,
          resistanceRatio: 0,
          spaceBetween: 0,
          centeredSlides: false,
          virtualTranslate: true
        };
        extend(swiper.params, overwriteParams);
        extend(swiper.originalParams, overwriteParams);
      },
      setTranslate: function setTranslate(swiper) {
        if (swiper.params.effect !== 'cube') return;
        swiper.cubeEffect.setTranslate();
      },
      setTransition: function setTransition(swiper, duration) {
        if (swiper.params.effect !== 'cube') return;
        swiper.cubeEffect.setTransition(duration);
      }
    }
  };

  var Flip = {
    setTranslate: function setTranslate() {
      var swiper = this;
      var slides = swiper.slides,
          rtl = swiper.rtlTranslate;

      for (var i = 0; i < slides.length; i += 1) {
        var $slideEl = slides.eq(i);
        var progress = $slideEl[0].progress;

        if (swiper.params.flipEffect.limitRotation) {
          progress = Math.max(Math.min($slideEl[0].progress, 1), -1);
        }

        var offset = $slideEl[0].swiperSlideOffset;
        var rotate = -180 * progress;
        var rotateY = rotate;
        var rotateX = 0;
        var tx = -offset;
        var ty = 0;

        if (!swiper.isHorizontal()) {
          ty = tx;
          tx = 0;
          rotateX = -rotateY;
          rotateY = 0;
        } else if (rtl) {
          rotateY = -rotateY;
        }

        $slideEl[0].style.zIndex = -Math.abs(Math.round(progress)) + slides.length;

        if (swiper.params.flipEffect.slideShadows) {
          // Set shadows
          var shadowBefore = swiper.isHorizontal() ? $slideEl.find('.swiper-slide-shadow-left') :
$slideEl.find('.swiper-slide-shadow-top');
          var shadowAfter = swiper.isHorizontal() ? $slideEl.find('.swiper-slide-shadow-right') :
$slideEl.find('.swiper-slide-shadow-bottom');
```

```
          if (shadowBefore.length === 0) {
            shadowBefore = $("<div class=\"swiper-slide-shadow-" + (swiper.isHorizontal() ? 'left' :
'top') + "\"></div>");
            $slideEl.append(shadowBefore);
          }

          if (shadowAfter.length === 0) {
            shadowAfter = $("<div class=\"swiper-slide-shadow-" + (swiper.isHorizontal() ? 'right' :
'bottom') + "\"></div>");
            $slideEl.append(shadowAfter);
          }

          if (shadowBefore.length) shadowBefore[0].style.opacity = Math.max(-progress, 0);
          if (shadowAfter.length) shadowAfter[0].style.opacity = Math.max(progress, 0);
        }

        $slideEl.transform("translate3d(" + tx + "px, " + ty + "px, 0px) rotateX(" + rotateX + "deg)
rotateY(" + rotateY + "deg)");
      }
    },
    setTransition: function setTransition(duration) {
      var swiper = this;
      var slides = swiper.slides,
          activeIndex = swiper.activeIndex,
          $wrapperEl = swiper.$wrapperEl;
      slides.transition(duration).find('.swiper-slide-shadow-top, .swiper-slide-shadow-right,
.swiper-slide-shadow-bottom, .swiper-slide-shadow-left').transition(duration);

      if (swiper.params.virtualTranslate && duration !== 0) {
        var eventTriggered = false; // eslint-disable-next-line

        slides.eq(activeIndex).transitionEnd(function onTransitionEnd() {
          if (eventTriggered) return;
          if (!swiper || swiper.destroyed) return; // if
(!$(this).hasClass(swiper.params.slideActiveClass)) return;

          eventTriggered = true;
          swiper.animating = false;
          var triggerEvents = ['webkitTransitionEnd', 'transitionend'];

          for (var i = 0; i < triggerEvents.length; i += 1) {
            $wrapperEl.trigger(triggerEvents[i]);
          }
        });
      }
    }
  };
  var EffectFlip = {
    name: 'effect-flip',
    params: {
      flipEffect: {
        slideShadows: true,
        limitRotation: true
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        flipEffect: _extends({}, Flip)
      });
    },
    on: {
      beforeInit: function beforeInit(swiper) {
        if (swiper.params.effect !== 'flip') return;
        swiper.classNames.push(swiper.params.containerModifierClass + "flip");
        swiper.classNames.push(swiper.params.containerModifierClass + "3d");
        var overwriteParams = {
          slidesPerView: 1,
          slidesPerColumn: 1,
          slidesPerGroup: 1,
          watchSlidesProgress: true,
          spaceBetween: 0,
          virtualTranslate: true
        };
        extend(swiper.params, overwriteParams);
        extend(swiper.originalParams, overwriteParams);
      },
      setTranslate: function setTranslate(swiper) {
        if (swiper.params.effect !== 'flip') return;
        swiper.flipEffect.setTranslate();
      },
      setTransition: function setTransition(swiper, duration) {
        if (swiper.params.effect !== 'flip') return;
        swiper.flipEffect.setTransition(duration);
      }
    }
  };

  var Coverflow = {
    setTranslate: function setTranslate() {
      var swiper = this;
      var swiperWidth = swiper.width,
          swiperHeight = swiper.height,
          slides = swiper.slides,
          slidesSizesGrid = swiper.slidesSizesGrid;
      var params = swiper.params.coverflowEffect;
      var isHorizontal = swiper.isHorizontal();
      var transform = swiper.translate;
      var center = isHorizontal ? -transform + swiperWidth / 2 : -transform + swiperHeight / 2;
      var rotate = isHorizontal ? params.rotate : -params.rotate;
      var translate = params.depth; // Each slide offset from center

      for (var i = 0, length = slides.length; i < length; i += 1) {
        var $slideEl = slides.eq(i);
        var slideSize = slidesSizesGrid[i];
        var slideOffset = $slideEl[0].swiperSlideOffset;
        var offsetMultiplier = (center - slideOffset - slideSize / 2) / slideSize * params.modifier;
        var rotateY = isHorizontal ? rotate * offsetMultiplier : 0;
        var rotateX = isHorizontal ? 0 : rotate * offsetMultiplier; // var rotateZ = 0

        var translateZ = -translate * Math.abs(offsetMultiplier);
        var stretch = params.stretch; // Allow percentage to make a relative stretch for responsive
sliders

        if (typeof stretch === 'string' && stretch.indexOf('%') !== -1) {
          stretch = parseFloat(params.stretch) / 100 * slideSize;
        }

        var translateY = isHorizontal ? 0 : stretch * offsetMultiplier;
        var translateX = isHorizontal ? stretch * offsetMultiplier : 0;
        var scale = 1 - (1 - params.scale) * Math.abs(offsetMultiplier); // Fix for ultra small
values

        if (Math.abs(translateX) < 0.001) translateX = 0;
        if (Math.abs(translateY) < 0.001) translateY = 0;
        if (Math.abs(translateZ) < 0.001) translateZ = 0;
        if (Math.abs(rotateY) < 0.001) rotateY = 0;
        if (Math.abs(rotateX) < 0.001) rotateX = 0;
        if (Math.abs(scale) < 0.001) scale = 0;
        var slideTransform = "translate3d(" + translateX + "px," + translateY + "px," + translateZ +
"px)  rotateX(" + rotateX + "deg) rotateY(" + rotateY + "deg) scale(" + scale + ")";
        $slideEl.transform(slideTransform);
        $slideEl[0].style.zIndex = -Math.abs(Math.round(offsetMultiplier)) + 1;

        if (params.slideShadows) {
          // Set shadows
          var $shadowBeforeEl = isHorizontal ? $slideEl.find('.swiper-slide-shadow-left') :
$slideEl.find('.swiper-slide-shadow-top');
          var $shadowAfterEl = isHorizontal ? $slideEl.find('.swiper-slide-shadow-right') :
$slideEl.find('.swiper-slide-shadow-bottom');

          if ($shadowBeforeEl.length === 0) {
            $shadowBeforeEl = $("<div class=\"swiper-slide-shadow-" + (isHorizontal ? 'left' : 'top')
+ "\"></div>");
            $slideEl.append($shadowBeforeEl);
          }

          if ($shadowAfterEl.length === 0) {
            $shadowAfterEl = $("<div class=\"swiper-slide-shadow-" + (isHorizontal ? 'right' :
'bottom') + "\"></div>");
            $slideEl.append($shadowAfterEl);
          }

          if ($shadowBeforeEl.length) $shadowBeforeEl[0].style.opacity = offsetMultiplier > 0 ?
offsetMultiplier : 0;
          if ($shadowAfterEl.length) $shadowAfterEl[0].style.opacity = -offsetMultiplier > 0 ? -
offsetMultiplier : 0;
        }
      }
    },
    setTransition: function setTransition(duration) {
      var swiper = this;
      swiper.slides.transition(duration).find('.swiper-slide-shadow-top, .swiper-slide-shadow-right,
.swiper-slide-shadow-bottom, .swiper-slide-shadow-left').transition(duration);
    }
  };
  var EffectCoverflow = {
    name: 'effect-coverflow',
    params: {
      coverflowEffect: {
        rotate: 50,
        stretch: 0,
        depth: 100,
        scale: 1,
        modifier: 1,
        slideShadows: true
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        coverflowEffect: _extends({}, Coverflow)
      });
    },
    on: {
      beforeInit: function beforeInit(swiper) {
        if (swiper.params.effect !== 'coverflow') return;
        swiper.classNames.push(swiper.params.containerModifierClass + "coverflow");
        swiper.classNames.push(swiper.params.containerModifierClass + "3d");
        swiper.params.watchSlidesProgress = true;
        swiper.originalParams.watchSlidesProgress = true;
      },
      setTranslate: function setTranslate(swiper) {
        if (swiper.params.effect !== 'coverflow') return;
        swiper.coverflowEffect.setTranslate();
      },
      setTransition: function setTransition(swiper, duration) {
        if (swiper.params.effect !== 'coverflow') return;
        swiper.coverflowEffect.setTransition(duration);
      }
    }
  };

  var Thumbs = {
    init: function init() {
      var swiper = this;
      var thumbsParams = swiper.params.thumbs;
      if (swiper.thumbs.initialized) return false;
      swiper.thumbs.initialized = true;
      var SwiperClass = swiper.constructor;

      if (thumbsParams.swiper instanceof SwiperClass) {
        swiper.thumbs.swiper = thumbsParams.swiper;
        extend(swiper.thumbs.swiper.originalParams, {
          watchSlidesProgress: true,
          slideToClickedSlide: false
        });
        extend(swiper.thumbs.swiper.params, {
          watchSlidesProgress: true,
          slideToClickedSlide: false
        });
      } else if (isObject(thumbsParams.swiper)) {
        swiper.thumbs.swiper = new SwiperClass(extend({}, thumbsParams.swiper, {
          watchSlidesVisibility: true,
          watchSlidesProgress: true,
          slideToClickedSlide: false
        }));
        swiper.thumbs.swiperCreated = true;
      }

      swiper.thumbs.swiper.$el.addClass(swiper.params.thumbs.thumbsContainerClass);
      swiper.thumbs.swiper.on('tap', swiper.thumbs.onThumbClick);
      return true;
    },
    onThumbClick: function onThumbClick() {
      var swiper = this;
      var thumbsSwiper = swiper.thumbs.swiper;
      if (!thumbsSwiper) return;
      var clickedIndex = thumbsSwiper.clickedIndex;
      var clickedSlide = thumbsSwiper.clickedSlide;
      if (clickedSlide && $(clickedSlide).hasClass(swiper.params.thumbs.slideThumbActiveClass))
return;
      if (typeof clickedIndex === 'undefined' || clickedIndex === null) return;
      var slideToIndex;

      if (thumbsSwiper.params.loop) {
        slideToIndex = parseInt($(thumbsSwiper.clickedSlide).attr('data-swiper-slide-index'), 10);
      } else {
        slideToIndex = clickedIndex;
      }

      if (swiper.params.loop) {
        var currentIndex = swiper.activeIndex;

        if (swiper.slides.eq(currentIndex).hasClass(swiper.params.slideDuplicateClass)) {
          swiper.loopFix(); // eslint-disable-next-line

          swiper._clientLeft = swiper.$wrapperEl[0].clientLeft;
          currentIndex = swiper.activeIndex;
        }

        var prevIndex = swiper.slides.eq(currentIndex).prevAll("[data-swiper-slide-index=\"" +
slideToIndex + "\"]").eq(0).index();
        var nextIndex = swiper.slides.eq(currentIndex).nextAll("[data-swiper-slide-index=\"" +
slideToIndex + "\"]").eq(0).index();
        if (typeof prevIndex === 'undefined') slideToIndex = nextIndex;else if (typeof nextIndex ===
'undefined') slideToIndex = prevIndex;else if (nextIndex - currentIndex < currentIndex - prevIndex)
slideToIndex = nextIndex;else slideToIndex = prevIndex;
      }

      swiper.slideTo(slideToIndex);
    },
    update: function update(initial) {
      var swiper = this;
      var thumbsSwiper = swiper.thumbs.swiper;
      if (!thumbsSwiper) return;
      var slidesPerView = thumbsSwiper.params.slidesPerView === 'auto' ?
thumbsSwiper.slidesPerViewDynamic() : thumbsSwiper.params.slidesPerView;
      var autoScrollOffset = swiper.params.thumbs.autoScrollOffset;
      var useOffset = autoScrollOffset && !thumbsSwiper.params.loop;

      if (swiper.realIndex !== thumbsSwiper.realIndex || useOffset) {
        var currentThumbsIndex = thumbsSwiper.activeIndex;
        var newThumbsIndex;
        var direction;

        if (thumbsSwiper.params.loop) {
          if
(thumbsSwiper.slides.eq(currentThumbsIndex).hasClass(thumbsSwiper.params.slideDuplicateClass)) {
            thumbsSwiper.loopFix(); // eslint-disable-next-line

            thumbsSwiper._clientLeft = thumbsSwiper.$wrapperEl[0].clientLeft;
            currentThumbsIndex = thumbsSwiper.activeIndex;
          } // Find actual thumbs index to slide to


          var prevThumbsIndex = thumbsSwiper.slides.eq(currentThumbsIndex).prevAll("[data-swiper-
slide-index=\"" + swiper.realIndex + "\"]").eq(0).index();
          var nextThumbsIndex = thumbsSwiper.slides.eq(currentThumbsIndex).nextAll("[data-swiper-
slide-index=\"" + swiper.realIndex + "\"]").eq(0).index();

          if (typeof prevThumbsIndex === 'undefined') {
            newThumbsIndex = nextThumbsIndex;
          } else if (typeof nextThumbsIndex === 'undefined') {
            newThumbsIndex = prevThumbsIndex;
          } else if (nextThumbsIndex - currentThumbsIndex === currentThumbsIndex - prevThumbsIndex) {
            newThumbsIndex = thumbsSwiper.params.slidesPerGroup > 1 ? nextThumbsIndex :
currentThumbsIndex;
          } else if (nextThumbsIndex - currentThumbsIndex < currentThumbsIndex - prevThumbsIndex) {
            newThumbsIndex = nextThumbsIndex;
          } else {
            newThumbsIndex = prevThumbsIndex;
          }

          direction = swiper.activeIndex > swiper.previousIndex ? 'next' : 'prev';
        } else {
          newThumbsIndex = swiper.realIndex;
          direction = newThumbsIndex > swiper.previousIndex ? 'next' : 'prev';
        }

        if (useOffset) {
          newThumbsIndex += direction === 'next' ? autoScrollOffset : -1 * autoScrollOffset;
        }
        if (thumbsSwiper.visibleSlidesIndexes &&
thumbsSwiper.visibleSlidesIndexes.indexOf(newThumbsIndex) < 0) {
          if (thumbsSwiper.params.centeredSlides) {
            if (newThumbsIndex > currentThumbsIndex) {
              newThumbsIndex = newThumbsIndex - Math.floor(slidesPerView / 2) + 1;
            } else {
              newThumbsIndex = newThumbsIndex + Math.floor(slidesPerView / 2) - 1;
            }
          } else if (newThumbsIndex > currentThumbsIndex && thumbsSwiper.params.slidesPerGroup === 1)
;

          thumbsSwiper.slideTo(newThumbsIndex, initial ? 0 : undefined);
        }
      } // Activate thumbs


      var thumbsToActivate = 1;
      var thumbActiveClass = swiper.params.thumbs.slideThumbActiveClass;

      if (swiper.params.slidesPerView > 1 && !swiper.params.centeredSlides) {
        thumbsToActivate = swiper.params.slidesPerView;
      }

      if (!swiper.params.thumbs.multipleActiveThumbs) {
        thumbsToActivate = 1;
      }

      thumbsToActivate = Math.floor(thumbsToActivate);
      thumbsSwiper.slides.removeClass(thumbActiveClass);

      if (thumbsSwiper.params.loop || thumbsSwiper.params.virtual &&
thumbsSwiper.params.virtual.enabled) {
        for (var i = 0; i < thumbsToActivate; i += 1) {
          thumbsSwiper.$wrapperEl.children("[data-swiper-slide-index=\"" + (swiper.realIndex + i) +
"\"]").addClass(thumbActiveClass);
        }
      } else {
        for (var _i = 0; _i < thumbsToActivate; _i += 1) {
          thumbsSwiper.slides.eq(swiper.realIndex + _i).addClass(thumbActiveClass);
        }
      }
    }
  };
  var Thumbs$1 = {
    name: 'thumbs',
    params: {
      thumbs: {
        swiper: null,
        multipleActiveThumbs: true,
        autoScrollOffset: 0,
        slideThumbActiveClass: 'swiper-slide-thumb-active',
        thumbsContainerClass: 'swiper-container-thumbs'
      }
    },
    create: function create() {
      var swiper = this;
      bindModuleMethods(swiper, {
        thumbs: _extends({
          swiper: null,
          initialized: false
        }, Thumbs)
      });
    },
    on: {
      beforeInit: function beforeInit(swiper) {
        var thumbs = swiper.params.thumbs;
        if (!thumbs || !thumbs.swiper) return;
        swiper.thumbs.init();
        swiper.thumbs.update(true);
      },
      slideChange: function slideChange(swiper) {
        if (!swiper.thumbs.swiper) return;
        swiper.thumbs.update();
      },
      update: function update(swiper) {
        if (!swiper.thumbs.swiper) return;
        swiper.thumbs.update();
      },
      resize: function resize(swiper) {
        if (!swiper.thumbs.swiper) return;
        swiper.thumbs.update();
      },
      observerUpdate: function observerUpdate(swiper) {
        if (!swiper.thumbs.swiper) return;
        swiper.thumbs.update();
      },
      setTransition: function setTransition(swiper, duration) {
        var thumbsSwiper = swiper.thumbs.swiper;
        if (!thumbsSwiper) return;
        thumbsSwiper.setTransition(duration);
      },
      beforeDestroy: function beforeDestroy(swiper) {
        var thumbsSwiper = swiper.thumbs.swiper;
        if (!thumbsSwiper) return;

        if (swiper.thumbs.swiperCreated && thumbsSwiper) {
          thumbsSwiper.destroy();
        }
      }
    }
  };

  // Swiper Class
  var components = [Virtual$1, Keyboard$1, Mousewheel$1, Navigation$1, Pagination$1, Scrollbar$1,
Parallax$1, Zoom$1, Lazy$1, Controller$1, A11y$1, History$1, HashNavigation$1, Autoplay$1,
EffectFade, EffectCube, EffectFlip, EffectCoverflow, Thumbs$1];
  Swiper.use(components);

  return Swiper;

})));
//# sourceMappingURL=swiper-bundle.js.map
```