```
var sort = require('./modules/sort')
var id = require('./modules/id')
var format = require('./modules/format')
var filter = require('./modules/filter')
var domain = require('./modules/domain')
var range = require('./modules/range')

/* global d3, alert, $, THREE, TWEEN, requestAnimationFrame */

/*
 *
 *
 * Initialisierung Visualisation
 *
 *
 */

// Für die Visualisation benötigte Variablen

var config,           // Config-Array für _alle_ Elemente
  datasetsMeta,       // Das 'datasets'-Attribut von meta.json
  index,              // Config-Objekt für die Index-Spalte (X-Wert)
  values,             // Config-Array für Werte-Spalten (Y-Werte)
  v_accessor,         // Funktion, die den Werteaccessor zurückgibt
  v_accessor_cord,    // Funktion, die den Koordinatenaccessor zurückgibt
  v_accessor_scaled,  // Funktion, die den skalierten Wert zurückgibt.
  v_bundle,           // Objekt, das die drei v-Funktionen enthält.

  xScale,             // x-Skala
  yScale,             // y-Skala
  zScale,             // z-Skala
  xWertebereich,      // Bereich der x-Werte
  yWertebereich,      // Bereich der y-Werte
  zWertebereich       // Bereich der z-Werte

/**
 * Laden der Konfigurationsdatei
 * @param  {[String]} "meta.json"             Der Dateiname für die
 *                                            Konfigurationsdatei
 * @param  {[Function]} function(err, config) Callback
 */
d3.json('meta.json', function (err, res) {
  if (err) {
    console.log(err)
    alert(err)
    return
  }

  config = []
  datasetsMeta = res.datasets

  index = {}
  values = []

  var colors = d3.scale.category20()

  for (var i = 0; i < datasetsMeta.length; i++) {
    var dataset = datasetsMeta[i]
    var url = dataset.url

    for (var j = 0; j < dataset.config.length; j++) {
      var c = dataset.config[j]
      c.url = url

      // ID generieren
      c.rowId = id.get(c)

      config.push(c)

      // Einfügen der Config in index oder values
      if (c.type === 'index') {
        index = c
      } else if (c.type === 'value') {
```

```
        // Spaltenspezifische Farbe generieren
        c.color = colors(values.length + 1)

        // Wenn das Attribut activated nicht gesetzt ist, setze es auf true.
        if (typeof c.activated === 'undefined') {
          c.activated = true
        }
        values.push(c)
      }
    }
  // Bei unbekannten Typen: nicht in values oder index einfügen.
  }

  // Datentyp der Skalen festlegen
  if (index.data_type === 'Number') {
    xScale = d3.scale.linear()
  } else if (index.data_type === 'Date') {
    xScale = d3.time.scale()
  }

  if (values[0].data_type === 'Number') {
    yScale = d3.scale.linear()
  } else if (values[0].data_type === 'Date') {
    yScale = d3.time.scale()
  }

  if (values[1].data_type === 'Number') {
    zScale = d3.scale.linear()
  } else if (values[1].data_type === 'Date') {
    zScale = d3.time.scale()
  }

  // Wertebereich der Achsenskalierungen definieren. Hier ist die Anzahl der Pixel
  // gemeint, über die sich die Achsen erstrecken. Die x-Achse und die y-Achse
  // verschieben wir um 50 nach rechts, damit man die y-Achse beschriften kann.
  xScale.range([0, 100])
  yScale.range([0, 100])
  zScale.range([0, 100])

  /*
   *
   *
   * Accessors für die Daten
   *
   *
   */

  // Index-Accessor-Funktion: Gibt für eine bestimmte Datenreihe den Wert der
    // Index-Spalte zurück.

  index.accessor = function (d) {
    return d[index.row]
  }

  // ..._scaled: Gibt den skalierten Wert von accessor zurück.
  index.accessor_scaled = function (d) {
    return xScale(d[index.row])
  }

  // Funktion, welche die Werte-Accessor-Funktion zurückgibt. Da sich die Werte-
  // Accessor-Funktionen im Gegensatz zum statischen Index-Accessor unterschei-
  // den, müssen sie für jede Spalte neu generiert werden. Diese Funktion ist
  // dafür zuständig.

  v_accessor = function (entry) {
    return function (d) {
      return d[entry.rowId]
    }
  }

  v_accessor_scaled = function (entry) {
    return function (d) {
      return yScale(d[entry.rowId])
```

```
    }
  }

  // Funktion, die den Koordinatenaccessor für die in entry angegebene Spalte
  // zurückgibt.
  v_accessor_cord = function (index, entry) {
    return function (d) {
      return [index.accessor_scaled(d), v_accessor_scaled(entry)(d)]
    }
  }

  v_bundle = {
    'raw': v_accessor,
    'scaled': v_accessor_scaled,
    'cord': v_accessor_cord
  }

  // Die Daten laden
  loadFiles()
})

/*
 *
 *
 * Laden der Daten
 *
 *
 */

/**
 * Die Funktion, die den Datensatz lädt und vorbereitet
 *
 * Vorgehen:  1. Laden der Daten
 *                                                    2. Formatieren des Datensatzes (data
_types und id)
 *                                                    3. 'Mergen' mit den anderen Datensät
zen, d. h. zusammenfügen
 *                                                    4. Sortieren
 *                                                    5. Die gemergten Datensätze weiterge
ben
 */
function loadFiles () {
  // Anzahl von Dateien, die schon heruntergeladen wurde
  var loaded = 0

  // Die Variable für die gemergten Datensätze
  var data = []

  // Jedes einzelne File herunterladen (1)
  for (var i = 0; i < datasetsMeta.length; i++) {
    d3.csv(datasetsMeta[i].url, mkcb(i))
  }

  /**
   * Funktion, die die Callback-Funktion für einen bestimmten Datensatz-Meta-
   * daten-Objekt mit Index i zurückgibt. Siehe auch: MKCB-Problem
   * @param  {[Number]} i   Index des Datensatz-Metadaten-Objekts aus
   *                        datasetsMeta
   * @return {[Function]}   Das generierte Callback, das nach dem Laden der
   *                        Datei ausgeführt wird
   */
  function mkcb (i) {
    return function (err, resp) {
      if (err) {
        alert(err)
        console.log(err)
        return
      }

      // Formatieren (2)
      resp = format.data_types(resp, datasetsMeta[i].config)
      resp = format.ids(resp, datasetsMeta[i].config)
```

```
      // Merge (3)
      for (var j = 0; j < resp.length; j++) {
        data.push(resp[j])
      }

      if (++loaded === datasetsMeta.length) {
        // Alle Datein sind heruntergeladen worden und gemergt.

        // Sortieren (4)
        data = sort(data, index)

        // Weitergeben (5)
        loadVisualization(data)
      }
    }
  }
}

/*
 *
 *
 * Laden der Visualisation
 *
 *
 */

/**
 * Lädt die Visualisation
 * @param  {[Array]} data Die gemergten Datensätze
 */
function loadVisualization (data) {
  $('#xtext').html((index.name ? index.name : index.row))
  $('#ytext').html((values[0].name ? values[0].name : values[0].row) + ' in ' + valu
es[0].unit)
  $('#ztext').html((values[1].name ? values[1].name : values[1].row) + ' in ' + valu
es[1].unit)

  xWertebereich = domain.overflowX(data, index, 1.1)

  yWertebereich = []
  zWertebereich = []

  yWertebereich[0] = range.min(data, v_bundle.raw(values[0]))
  yWertebereich[1] = range.max(data, v_bundle.raw(values[0]))
  yWertebereich[1] = range.applyOverflow(yWertebereich[0], yWertebereich[1],
    1.1, values[0].data_type)

  zWertebereich[0] = range.min(data, v_bundle.raw(values[1]))
  zWertebereich[1] = range.max(data, v_bundle.raw(values[1]))
  zWertebereich[1] = range.applyOverflow(zWertebereich[0], zWertebereich[1],
    1.1, values[1].data_type)

  xScale.domain(xWertebereich)
  yScale.domain(yWertebereich)
  zScale.domain(zWertebereich)

  var camera, controls, scene, renderer, material
  init()
  render()
  function animate () {
    requestAnimationFrame(animate)
    controls.update()
    TWEEN.update()
  }
  function init () {
    scene = new THREE.Scene()

    var w = window.innerWidth * 0.8
    var h = window.innerHeight * 0.8

    // var cameraP = new THREE.PerspectiveCamera(45, w / h, 1, 10000)
    // var cameraO = new THREE.OrthographicCamera(w / -2, w / 2, h / 2, h / -2, 1, 1
000)
```

```
    camera = new THREE.CombinedCamera(w / 2, h / 2, 70, 1, 1000, -500, 1000)

    // override.
    camera.setZoom(1)
    camera.toPerspective()

    camera.position.x = 80
    camera.position.y = 70
    camera.position.z = 150

    material = new THREE.MeshBasicMaterial({ color: 0x000000, wireframe: false })

    renderer = new THREE.WebGLRenderer({ alpha: true, antialias: true })
    renderer.setSize(w, h)

    controls = new THREE.OrbitControls(camera, renderer.domElement)
    controls.damping = 0.2
    controls.addEventListener('change', render)

    axis()

    document.getElementById('visualization-wrap').appendChild(renderer.domElement)

    animate()

    xScale.range([0, 100])
    yScale.range([0, 100])

    points()
  }

  function render () {
    renderer.render(scene, camera)
    console.log('cords: ', camera.position.x, camera.position.y, camera.position.z)
    console.log('in scene: ', camera.position.x + 50, camera.position.y + 50, camera
.position.z + 50)
  // console.log(camera.rotation.x*180/Math.PI, camera.rotation.y*180/Math.PI, camer
a.rotation.z*180/Math.PI)
  }

  function toScene (x, y, z) {
    return [x - 50, y - 50, z - 50]
  }

  function axis () {
    var origin = new THREE.Vector3(-50, -50, -50)
    var length = 100

    scene.add(new THREE.ArrowHelper(new THREE.Vector3(1, 0, 0), origin, length, 0xff
0000))
    scene.add(new THREE.ArrowHelper(new THREE.Vector3(0, 1, 0), origin, length, 0x00
ff00))
    scene.add(new THREE.ArrowHelper(new THREE.Vector3(0, 0, 1), origin, length, 0x00
00ff))

    var dashed = new THREE.LineDashedMaterial({
      color: 0xdedede,
      dashSize: 3,
      gapSize: 2,
      scale: 1
    })

    // Box zeichnen
    var xy1 = [toScene(100, 100, 0), toScene(100, 0, 0)]
    var xy2 = [toScene(100, 100, 0), toScene(0, 100, 0)]
    var yz1 = [toScene(0, 100, 100), toScene(0, 100, 0)]
    var yz2 = [toScene(0, 100, 100), toScene(0, 0, 100)]
    var xz1 = [toScene(100, 0, 100), toScene(100, 0, 0)]
    var xz2 = [toScene(100, 0, 100), toScene(0, 0, 100)]
    var xyz1 = [toScene(100, 100, 100), toScene(100, 100, 0)]
    var xyz2 = [toScene(100, 100, 100), toScene(100, 0, 100)]
    var xyz3 = [toScene(100, 100, 100), toScene(0, 100, 100)]
```

```
    for (var i = 0; i < 9; i++) {
      var a = [xy1, xy2, xz1, xz2, yz1, yz2, xyz1, xyz2, xyz3][i]
      var lg = new THREE.Geometry()
      lg.vertices.push(new THREE.Vector3(a[0][0], a[0][1], a[0][2]))
      lg.vertices.push(new THREE.Vector3(a[1][0], a[1][1], a[1][2]))
      lg.computeLineDistances()
      var line = new THREE.Line(lg, dashed)
      scene.add(line)
    }
  }

  function points () {
    var dataY = filter.row(data, values[0].rowId)
    var dataZ = filter.row(data, values[1].rowId)
    for (var i = 0; i < dataY.length; i++) {
      var x = xScale(index.accessor(dataY[i]))
      var y = yScale(dataY[i][values[0].rowId])
      var z = zScale(dataZ[i][values[1].rowId])

      var sphere = new THREE.SphereGeometry(0.5, 8, 6)
      var smesh = new THREE.Mesh(sphere, material)
      var arr = toScene(x, y, z)
      smesh.translateX(arr[0])
      smesh.translateY(arr[1])
      smesh.translateZ(arr[2])
      scene.add(smesh)
    }
  }

  $('#toPerspective').click(function () {
    camera.toPerspective()
    camera.setZoom(1)
    camera.updateProjectionMatrix()
    render()
  })

  $('#toOrtho').click(function () {
    camera.toOrthographic()
    camera.setZoom(5)
    camera.updateProjectionMatrix()
    render()
  })

  $('#xy').click(function () {
    ortho('xy')
  })

  $('#xz').click(function () {
    ortho('xz')
  })

  $('#zy').click(function () {
    ortho('zy')
  })

  function ortho (mode) {
    if (camera.inPerspectiveMode) {
      camera.setZoom(1)
    } else {
      camera.setZoom(5)
    }
    var cc
    if (mode === 'xy') {
      cc = toScene(50, 50, 200)
    } else if (mode === 'xz') {
      cc = toScene(50, -100, 50)
    } else if (mode === 'zy') {
      cc = toScene(-100, 50, 50)
    }

    var posx = { x: camera.position.x }
    var posy = { x: camera.position.y }
    var posz = { x: camera.position.z }
```

```
    var tarx = { x: cc[0] }
    var tary = { x: cc[1] }
    var tarz = { x: cc[2] }

    var tx = new TWEEN.Tween(posx).to(tarx, 1400)
    var ty = new TWEEN.Tween(posy).to(tary, 1400)
    var tz = new TWEEN.Tween(posz).to(tarz, 1400)

    tx.easing(TWEEN.Easing.Cubic.InOut)
    ty.easing(TWEEN.Easing.Cubic.InOut)
    tz.easing(TWEEN.Easing.Cubic.InOut)

    tx.onUpdate(function () {
      camera.position.x = posx.x
    })
    ty.onUpdate(function () {
      camera.position.y = posy.x
    })
    tz.onUpdate(function () {
      camera.position.z = posz.x
    })

    tx.start()
    ty.start()
    tz.start()
  }
}
```