JSDoc Documentation

# app.js.html

# Source: app.js

import { showTests } from "./tests/tests.js";  
import { showAreas } from "./areas/areas.js";  
  
import { handleTimer } from "./timer/timer.js";  
  
import { getData } from "./data/get-data.js";  
import { getFromLocalStorage, saveToLocalStorage } from "./utils/localStorage.js";  
  
let counter = 0  
  
const handleInterface = () => {  
 counter += 1  
 handleTimer()  
  
 if (counter % 3 === 0) {  
 counter = 0  
 fetchZones()  
 }  
}  
/\*\*  
 \* Асинхронно получает данные, сохраняет их в локальное хранилище и отображает тесты и области  
 \* @returns {Promise<number>} Число 0 после успешного выполнения  
 \*/  
async function fetchZones() {  
   
 let data = await getData()  
 saveToLocalStorage(data)  
  
 showTests(data)  
 showAreas(data)  
  
 window.addEventListener('resize', () => {  
 showAreas(data)  
 })  
  
 return 0;  
}  
  
fetchZones()

* [fetchZones](global.html#fetchZones)

# areas.js.html

# Source: areas.js

import { makeJSONUnique } from "../utils/makeJSONUnique.js"  
import { shuffle } from "../utils/shuffle.js"  
  
let flag = false  
  
/\*\*  
 \* Отображает области в HTML-контейнере с идентификатором 'areas-container' и активирует обработчики кликов  
 \* @param {Object} blocks - Объект с данными, содержащий MainBlock с секциями  
 \* @returns {boolean} Возвращает true после успешного выполнения  
 \*/  
const showAreas = (blocks) => {  
 const areasContainer = document.getElementById('areas-container')  
 areasContainer.innerHTML = ''  
  
 for (let i = 0; i < blocks.MainBlock.Sections.length; i++) {  
 const area = blocks.MainBlock.Sections[i]  
  
 const areaDiv = document.createElement('div')  
 areaDiv.className = 'area-div'  
  
 if (i === 0) {  
 areaDiv.id = 'area-active'  
 }  
  
 if (i === blocks.MainBlock.Sections.length - 1 && i !== 0) {  
 areaDiv.id = 'area-last'  
 }  
  
 const areaHeading = document.createElement('h2')  
 areaHeading.textContent = area.GeneralName  
  
 areaDiv.appendChild(areaHeading)  
  
 areaDiv.addEventListener('click', () => {  
 const active = document.getElementById('area-active')  
 active.removeAttribute('id')  
  
 areaDiv.id = 'area-active'  
  
 localStorage.setItem('active-zone', areaDiv.textContent)  
  
 showActiveArea(blocks)  
 })  
  
 areasContainer.appendChild(areaDiv)  
 }  
 showActiveArea(blocks)  
  
 return true  
}  
  
/\*\*  
 \* Находит информацию об области по её имени  
 \* @param {string} name - Имя области  
 \* @param {Object} blocks - Объект с данными, содержащий MainBlock с секциями  
 \* @returns {Object|undefined} Объект области или undefined, если область не найдена  
 \*/  
const getAreaInfoByName = (name, blocks) => {  
 for (let i = 0; i < blocks.MainBlock.Sections.length; i++) {  
 if (name === blocks.MainBlock.Sections[i].GeneralName) {  
 return blocks.MainBlock.Sections[i]  
 }  
 }  
}  
  
/\*\*  
 \* Отображает информацию об активной области в HTML-контейнере с идентификатором 'active-area-wrapper'  
 \* @param {Object} blocks - Объект с данными, содержащий MainBlock с секциями  
 \* @returns {void} Ничего не возвращает  
 \*/  
const showActiveArea = (blocks) => {  
 const activeAreaDiv = document.getElementById('area-active')  
  
 const activeAreaInfo = getAreaInfoByName(activeAreaDiv.textContent, blocks)  
  
 const areaInfoContainer = document.getElementById('active-area-wrapper')  
 areaInfoContainer.innerHTML = ''  
  
 let numOfCols = 5  
 const screenWidth = window.innerWidth  
  
 if (screenWidth <= 1024) {  
 numOfCols = 2  
 } else if (screenWidth < 1750) {  
 numOfCols = 3  
 } else if (screenWidth <= 1920) {  
 numOfCols = 4  
 }  
  
 for (let i = 0; i < activeAreaInfo.Parameters.length; i += numOfCols) {  
  
 const servers = activeAreaInfo.Parameters.slice(i, i + numOfCols)  
  
 const row = document.createElement("div")  
 row.className = 'active-area-row'  
  
 servers.forEach(server => {  
 const activeAreaContaier = document.createElement('div')  
 activeAreaContaier.className = 'active-area-container'  
  
 const activeAreaHeading = document.createElement('h1')  
 activeAreaHeading.textContent = server.IP + " " + server.Role  
  
 const recordsContainer = document.createElement('div')  
 recordsContainer.className = 'records-container'  
  
 server.Probes = shuffle(makeJSONUnique(server.Probes)).slice(0, 15)  
  
 for (let j = 0; j < server.Probes.length; j++) {  
 const record = server.Probes[j]  
  
 const recordDiv = document.createElement('div')  
 recordDiv.className = 'record-div'  
  
 const recordText = document.createElement('p')  
  
 if (record.Name === "SOA" && !flag) {  
 flag = true  
 } else {  
 recordText.textContent = record.Name  
  
 recordDiv.appendChild(recordText)  
  
 recordsContainer.append(recordDiv)  
 }  
 }  
  
 activeAreaContaier.appendChild(activeAreaHeading)  
 activeAreaContaier.appendChild(recordsContainer)  
  
 row.appendChild(activeAreaContaier)  
 })  
  
 areaInfoContainer.appendChild(row)  
 }  
  
}  
  
export { showAreas }

* [getAreaInfoByName](global.html#getAreaInfoByName)
* [showActiveArea](global.html#showActiveArea)
* [showAreas](global.html#showAreas)

# global.html

### Methods

#### (async) fetchZones() → {Promise.<number>}

Асинхронно получает данные, сохраняет их в локальное хранилище и отображает тесты и области

Source:

* [app.js](app.js.html), [line 24](app.js.html#line24)

##### Returns:

Число 0 после успешного выполнения

Type

Promise.<number>

* [fetchZones](global.html#fetchZones)

# index.html

# Home

* [fetchZones](global.html#fetchZones)

# localStorage.js.html

# Source: localStorage.js

/\*\*  
 \* Сохраняет объект данных в локальное хранилище под ключом "DATA"  
 \* @param {Object} blocks - Объект данных для сохранения  
 \* @returns {void} Ничего не возвращает  
 \*/  
function saveToLocalStorage(blocks) {  
 localStorage.setItem("DATA", JSON.stringify(blocks))  
}  
  
/\*\*  
 \* Получает данные из локального хранилища по ключу "DATA"  
 \* @returns {string|null} Строку JSON с данными или null, если данные отсутствуют  
 \*/  
function getFromLocalStorage() {  
 return localStorage.getItem("DATA")  
}  
  
export { saveToLocalStorage, getFromLocalStorage }

* [getFromLocalStorage](global.html#getFromLocalStorage)
* [saveToLocalStorage](global.html#saveToLocalStorage)

Documentation generated by [JSDoc 4.0.4](https://github.com/jsdoc/jsdoc) on Thu Jun 05 2025 15:50:19 GMT+0300 (Москва, стандартное время)

# makeJSONUnique.js.html

# Source: makeJSONUnique.js

/\*\*  
 \* Удаляет дубликаты из массива объектов, преобразуя их в уникальный массив  
 \* @param {Array<Object>} sections - Массив объектов для обработки  
 \* @returns {Array<Object>} Массив уникальных объектов  
 \*/  
  
export const makeJSONUnique = (sections) => {  
 return [...new Set(  
 sections.map(item => JSON.stringify(item))  
 )].map(item => JSON.parse(item));  
}

* [makeJSONUnique](global.html#makeJSONUnique)

Documentation generated by [JSDoc 4.0.4](https://github.com/jsdoc/jsdoc) on Thu Jun 05 2025 15:52:37 GMT+0300 (Москва, стандартное время)

# shuffle.js.html

# Source: shuffle.js

/\*\*  
 \* Перемешивает элементы массива случайным образом с использованием алгоритма Фишера-Йетса  
 \* @param {Array} array - Массив для перемешивания  
 \* @returns {Array} Перемешанный массив  
 \*/  
export function shuffle(array) {  
 let currentIndex = array.length;  
  
 while (currentIndex != 0) {  
  
 let randomIndex = Math.floor(Math.random() \* currentIndex);  
 currentIndex--;  
  
 [array[currentIndex], array[randomIndex]] = [  
 array[randomIndex], array[currentIndex]];  
 }  
  
 return array  
}

* [shuffle](global.html#shuffle)

Documentation generated by [JSDoc 4.0.4](https://github.com/jsdoc/jsdoc) on Thu Jun 05 2025 15:50:36 GMT+0300 (Москва, стандартное время)

# tests.js.html

# Source: tests.js

import { shuffle } from "../utils/shuffle.js";  
  
/\*\*  
 \* Отображает до 10 тестов в HTML-контейнере с идентификатором 'tests-container', группируя их по 5 в строке  
 \* @param {Object} blocks - Объект с данными, содержащий TopBlock с параметрами тестов  
 \* @returns {void} Ничего не возвращает  
 \*/  
const showTests = (blocks) => {  
 const testsContainer = document.getElementById('tests-container')  
 testsContainer.innerHTML = '';  
 const tests = shuffle(blocks.TopBlock.Parameters).slice(0, 10)  
  
 for (let i = 0; i < tests.length; i += 5) {  
 const row = document.createElement('div')  
 row.className = 'row'  
   
 let rowOfTests = tests.slice(i, i + 5)  
   
 rowOfTests.forEach(test => {  
 const testDiv = document.createElement('div');  
 testDiv.className = 'test-div';  
  
 if (test.success === true || test.success === "true") {  
 testDiv.style.backgroundColor = '#08fc64';  
 } else {  
 testDiv.style.backgroundColor = '#ff6464';  
 }  
  
 const headingContainer = document.createElement('div')  
 headingContainer.className = 'heading-container'  
  
 const nsName = document.createElement('h2')  
 nsName.textContent = test.GeneralName || "N/A"  
  
 const typeName = document.createElement('h3')  
 typeName.textContent = test.ProbeName;  
  
 headingContainer.appendChild(nsName)  
 headingContainer.appendChild(typeName)  
  
 const footerContainer = document.createElement('div')  
  
 footerContainer.className = "footer-container"  
  
 const ipName = document.createElement('p')  
 ipName.textContent = test.ip;  
  
 const lamp = document.createElement('p')  
 lamp.className = "lamp"  
 lamp.id = "lamp" + test.ip + test.GeneralName + test.GeneralName  
 lamp.innerHTML = "💡"  
  
 footerContainer.appendChild(ipName)  
 footerContainer.appendChild(lamp)  
  
 testDiv.appendChild(headingContainer)  
 testDiv.appendChild(footerContainer)  
 row.appendChild(testDiv)  
 })  
  
 testsContainer.appendChild(row)  
  
 }  
}  
  
export { showTests }

* [showTests](global.html#showTests)

# truncateWord.js.html

# Source: truncateWord.js

### /\*\* \* Обрезает строку до указанной длины, добавляя "..." в конец, если строка превышает максимальную длину \* @param {string} word - Строка для обрезки \* @param {number} maxLength - Максимальная длина строки \* @returns {string} Обрезанная строка или исходная строка, если её длина меньше или равна maxLength \*/ export const truncateWord = (word, maxLength) => { if (word.length > maxLength) { return word.slice(0, maxLength) + "..."; } return word; }

* [truncateWord](global.html#truncateWord)

Documentation generated by [JSDoc 4.0.4](https://github.com/jsdoc/jsdoc) on Thu Jun 05 2025 15:48:40 GMT+0300 (Москва, стандартное время)